Studienarbeit Informatik

Virtual IP strongSwan IKEv2

Fabian Hartmann
Noah Heusser

Wintersemester 2006/2007
9. Februar 2007

Betreuer:
Martin Willi
Prof. Dr. Andreas Steffen
Institut für Internet-Technologien und Anwendungen
Hochschule für Technik Rapperswil

Abstract

strongSwan

strongSwan ist eine der wenigen Open Source Internet Protocol Security (IPSec)-Produkte unter Linux. Es besitzt als einzige, der bestehenden Lösungen sowohl Internet Key Exchange Protocol (IKE)v1 als auch IKEv2 in seinem Repertoire. Die Unterstützung der neuen Version des Schlüsselaustauschprotokolls wurde im Rahmen der Diplomarbeit von Jan Hutter und Martin Willi entwickelt. Zwischenzeitlich wurde strongSwan II um einige wichtige Features wie z. Bsp. Network Address Translation (NAT)-Traversal erweitert und steht nun kurz vor dessen Komplettierung.

Zielsetzung

Ziel der Studienarbeit war es, die IKEv2-Komponente um Virtual Internet Protocol (IP) zu ergänzen. Eine virtuelle IP-Adresse ist die innere Adresse eines Virtual Private Network (VPN)-Endknotens. Das bedeutet, dass der Verkehr im Tunnel zwar an die Virtual IP adressiert wird, aber sämtliche Pakete über ein reales Netzwerkinterface verschickt werden.

Der Configuration-Payload von IKEv2 soll zum Aushandeln, neben der Virtual IP, weiterer Konfigurationsparameter wie Subnetzmaske, Domain Name System (DNS)-Server und Windows Internet Name Service (WINS)-Server sowie IPv6 genutzt werden. Ein VPN-Client kann somit eine Virtual IP vom Gateway anfordern und diese dann auf dessen System registrieren.

Ergebnis

Mit dem Abschluss dieser Studienarbeit ist strongSwan II nun in der Lage virtuelle IP-Adressen statisch im Configfile des Clients oder dynamisch via Configuration Payload einem Verbindungspartner zuzuweisen. Die Adressen und Routen werden im Linux-Kernel eingetragen und bei Terminierung der Verbindung wieder entfernt.

Als Erweiterung dieser Arbeit köennen nun Plugins füer die Verwendung von DNS- oder WINS-Server und das Adresspool-Management entwickelt werden.

Inhaltsverzeichnis

Kapitel 1

Aufgabenstellung

Einführung

strongSwan ist eine IPSec-basierte VPN Lösung für Linux. Das Institute für Internet Technologien und Anwendungen (ITA) treibt das Open Source Projekt voran.
Das neue Key Exchange Protocol IKEv2 (Request for Comment (RFC)4306) wird das Protokoll der Version 1 in Zukunft ablösen. Ein IKEv2 Daemon wurde bereits zu grossen Teilen durch das ITA unter Mithilfe von Hochschule für Technik Rapperswil (HSR) Studierenden implementiert und soll durch diese Arbeit um eine weiteres wichtiges Feature erweitert werden. Damit sich ein Roadwarrior mit seinem Heimatnetzwerk verbinden kann, ist es sinnvoll, diesem eine virtuelle IP Adresse zuzuordnen, damit die Antwortpakete den Weg zurück aus dem geschützten Netz über den Tunnel zum Remote Access Client finden.

Aufgabenstellung

- Einarbeitung in den strongSwan Source Code und in den IKEv2 Standard (RFC 4306).
- Erweitern der IKEv2-Implementierung um die Configuration Payload, welche das Aushandeln von virtuellen IP Adressen und anderen Parametern erlaubt.
- Konfiguration der zugeteilten Adresse auf der Client-Seite. Dabei soll die Kernel-Schnittstelle erweitert werden, so dass darüber Adressen mittels Netlink konfiguriert werden können.
- Bereitstellen von IP Adressen auf der Server-Seite wahlweise durch feste Zuordnung auf Benutzer oder über einen Address Pool, der durch den VPN Gateway verwaltet wird.
- Optional kann eine Anbindung an einen Remote Authentication Dial-In User Service (RADIUS) Server in betracht gezogen werden, welcher die Verwaltung der IP Adressen übernimmt.

Kapitel 2

Einleitung

2.1 Ziele

Das Hauptziel der Studienarbeit bestand darin, strongSwan's IKEv2[CK05] Daemon charon um die Fähigkeit zu erweitern, virtuelle IP-Adressen an Verbindungspartnern zuweisen zu können. Dies geschieht durch den in IKEv2 definierten Configuration Payload, welcher es ermöglicht diverse Netzwerkkonfigurationsparameter während dem Verbindungsaufbau zu übertragen. Namentlich sind dies IP-Adresse, Subnetzmaske, DNS-, WINS-Server und weitere, welche detailiert im IKEv2-Standard beschrieben sind.

2.2 strongSwan II

strongSwan ist eine komplette Open Source IPSec-Implementierung für Linux. Das Projekt wurde von Prof. Dr. Andreas Steffen ins Leben gerufen und fokussiert die starke Authentifizierung mittels X.509 Zertifikaten. Ursprünglich ist strongSwan aus dem Vorgängerprojekt FreeS/WAN entstanden, welches 2003 eingestellt wurde. strongSwan II[JH05], welches die IKEv2-Unterstützung brachte, wurde im Rahmen einer Diplomarbeit von Jan Hutter und Martin Willi entwickelt und während einigen Studienarbeiten um zahlreiche Funktionen ergänzt.

Kapitel 3

Grundlagen

Dieses Kapitel beschreibt die technischen Grundlagen, welche zum Verstaïdnis der nachfolgenden Kapitel notwendig sind. Wie der Name des Kapitels schon aussagt, werden lediglich grundlegende Eigenschaften beschrieben und keine Details behandelt. Für tiefere Informationen zu den einzelnen Themen sind die jeweiligen Standards oder Originaldokumente zu konsultieren.

3.1 IPSec

IPSec wurde als sichere Variante des Internet Protokolls entwickelt, um dessen Schwachstellen zu beseitigen. Detailierte Informationen über IPSec befinden sich im Bericht der Studienarbeit über NAT-Traversal[TB06] oder im entsprechenden RFC[SK98].

3.1.1 Internet Key Exchnge Protocol 2 (IKEv2)

IKE regelt die automatische Schlüsselverwaltung für IPSec. Dessen Version 2 wurde im Gegensatz zum Vorgänger komplett in einem RFC[CK05] verfasst und hat dadurch einiges an Komplexität verloren und dafür an Flexibilität dazu gewonnen. Für eine umfassende Beschreibung des Dokuments verweisen wir neben dem RFC[CK05] auf die Diplomarbeit von Jan Hutter und Martin Willi[JH05].

3.1.2 Configuration Payload (CP)

Der Configuration Payload, ferner Configuration Payload (CP) genannt, dient dem Austausch von IP-Konfigurationsparametern zwischen IKE-Verbindungspartnern. Dieser Austausch erinnert stark an das Dynamic Host Configuration Protocol (DHCP), welches sehr oft in lokalen Netzen zur automatischen Adresskonfiguration von einzelnen Netzwerkknoten eingesetzt wird.
CP-Transaktionen können auf zwei verschiedene Arten abgehandelt werden. Dazu werden die verschiedenen Typen verwendet, welche nun vorgestellt werden.

CFG_REQUEST/CFG_REPLY

Diese Transaktionsart ermöglicht es einem IKE-Endknoten Informationen von seinem Verbindungspartner anzufordern. Der Initiator kann sogar selbst schon Daten liefern, welche von der Gegenstelle dann als Vorschlag angesehen werden. Die Antwort beinhaltet schliesslich die geforderten Daten. Falls genannte Daten nicht vorhanden sind oder deren Attribute nicht unterstützt werden, wird ein Antwortpaket mit den Headerinformation der verlangten Attribute ohne Nutzdaten zurückgeschickt. Im Falle, dass kein einziges Attribut angenommen wurde, kann in der Antwort der CP weggelassen werden.

CFG_SET/CFG_ACK

Mit dieser Methode kann ein IKE-Endknoten seine Konfigurationsdaten an seinen Verbindungspartner übertragen. Der Initiator zwingt seinen Partner somit die im CP enthaltenen Attribute entsprechend anzupassen. In der Antwort sind alle Attribute enthalten, welche vom Verbindungspartner übernommen wurden. Momentan gibt es noch keine definerten Nutzungszwecke für diese Transaktionsmethode und wurde daher in strongSwan ignoriert.

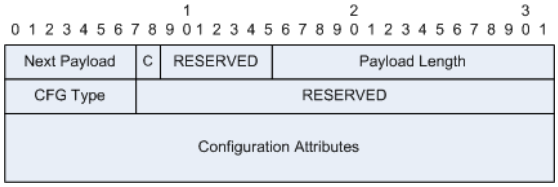

Abbildung 3.1: Configuration Payload

3.1.3 Configuration Attributes

Der CP enthält meistens ein oder mehrere Konfigurationsparameter, welche `Configuration Attributes` genannt werden. Sie enthalten die eigentlichen Adressdaten, welche die IKE-Endpunkte zur Systemkonfiguration verwenden.

Die wichtigsten Attribute werden in folgender Tabelle dargestellt:

3.2 RTNetlink

RTNetlink ist die Linux-Kernel Schnittstelle, welche Routing-Tabellen und Netzwerkinterfaces manipulieren kann. Benutzerprozesse sind durch Netlink in der

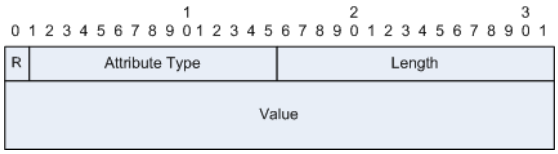

Abbildung 3.2: Configuration Attributes

Attribute Type	**Grösse**	**Beschreibung**
INTERNAL_IP4_ADDRESS	4 Bytes	Virtuelle IPv4-Adresse
INTERNAL_IP6_ADDRESS	17 Bytes	Virtuelle IPv6-Adresse
INTERNAL_IP4_NETMASK	4 Bytes	IPv4-Subnetzmaske
INTERNAL_IP6_NETMASK	17 Bytes	IPv6-Subnetzmaske
INTERNAL_IP4_DNS	4 Bytes	DNS-Server-Adresse
INTERNAL_IP6_DNS	16 Bytes	DNS-Server-Adresse
INTERNAL_IP4_NBNS	4 Bytes	NetBIOS-Name Server-Adresse
INTERNAL_IP6_NBNS	16 Bytes	NetBIOS-Server-Adresse

Tabelle 3.1: Configuration Attributes

Lage mit dem Kernel über einen speziellen Socket zu kommunizieren. In diesem Kapitel werden grundlegende Eigenschaften wie das Erstellen, Senden und Empfangen solcher Nachrichten behandelt.

Die Kommunikation mit dem Kernel ist datagrammbasiert und somit verbindungslos, was bedeutet, dass die sichere Übertragung von Paketen via Netlink nicht garantiert ist. Der Kernel ist also nicht verpflichtet auf eine Anfrage zu antworten. Netlink kann aber mittels Flag dazu gezwungen werden, eine gewisse Zuverlässigkeit zu erreichen. Das Senden und Empfangen von Nachrichten wird folglich mittels Socketfunktion `sendto` und `recvfrom` umgesetzt.

Nachfolgend ist der soeben beschriebene Netlink Socket dargestellt:

Listing 3.1: rtnetlink Socket

```
rt_sock = socket(PF_NETLINK,SOCK_DGRAM,NETLINK_ROUTE);
```

3.2.1 Nachrichtenformat

Netlink propagiert einen Request-Reply-Mechanismus, um Informationen des Kernels zu senden oder zu erhalten. Eine Nachricht besteht aus einem Stream aufeinanderfolgender Strukturen, welcher der Sender/Empfänger abfüllen muss. Das folgende Listing zeigt den internen Aufbau der Paketheaderstruktur:

Listing 3.2: Netlink Nachrichtenformat

```
struct nlmsghdr
{
        __u32   nlmsg_len;
        __u16   nlmsg_type;
        __u16   nlmsg_flags;
        __u32   nlmsg_seq;
        __u32   nlmsg_pid;
}
```

nlmsg_len Enthält die Länge der gesamten RTNETLINK-Nachricht inklusive der nlmsghdr-Struktur. Dieses Feld kann mittels der Makrofunktion NLMSG_ALIGN(len[1]) gesetzt werden.

nlmsg_type 16-Bit grosser Nachrichtentyp, welcher im folgenden Unterkapitel genauer beschrieben wird.

nlmsg_flags 16-Bit grosses Flag, das die Zuverlässigkeit von Netlink während dessen Kommunikation konfiguriert. Mögliche Werte sind z. Bsp.:

- NLM_F_REQUEST: Nachricht ist ein Request.
- NLM_F_ACK: Antwort enthält Bestätigung, falls erfolgreich oder entsprechenden Fehlercode.
- NLM_F_CREATE: Adresse hinzufügen, falls sie noch nicht existiert. Wird bei NEW-Requests benötigt.

[1] Länge der Netlink-Nachricht abgefüllt mit Daten

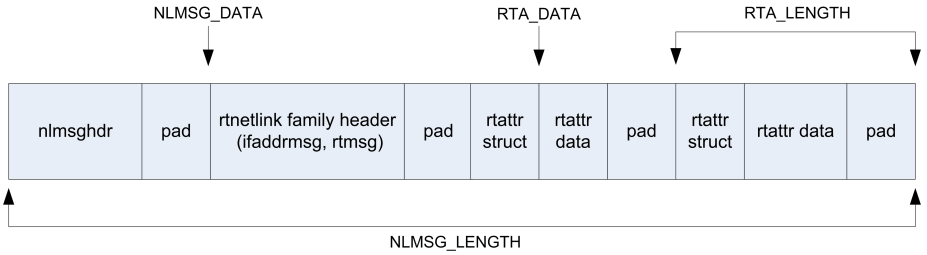

Abbildung 3.3: Aufbau einer RTNetlink-Nachricht

3.2.2 Nachrichtenmakros

NETLINK_ROUTE stellt diverse Makrofunktionen zur Verfügung, welche das Zusammenstellen von Nachrichten etwas erleichtern. Die Wichtigsten werden hier vorgestellt und beschrieben. Um diese Makros erfolgreich nutzen zu können, benötigt man folgende rtattr-Struktur:

Listing 3.3: struct rtattr

```
struct rtattr
{
        unsigned short rta_len;
        unsigned short rta_type;
}
```

```
void* RTA_DATA(struct rtattr *rta);
```

Gibt einen Zeiger auf die Stelle in der rtattr-Struktur zurück, an welche die Nutzdaten kopiert werden müssen.

```
unsigned int RTA_LENGTH(unsigned int length);
```

Gibt die korrekte Länge für die übergebene Nutzlastgrösse zurück. Dieser Wert wird dem Feld rta_len zugewiesen.

3.2.3 Nachrichtenfamilien und -typen

In folgendem Abschnitt werden die verschiedenen Nachrichtenfamilien und dessen Typen von RTNetlink behandelt. Es wird nur auf die beiden, in dieser Arbeit verwendeten, Familien eingegangen. Für ein Studium der weiteren Familien sei auf das Manual[Hor04] von Neil Horman verwiesen. Die Manpage von rtnetlink - RTNETLINK(7)[rtn99] - sei hier nur am Rande vermerkt, da diese nach dessen eigener Aussage mangelhaft und demnach nicht sehr empfehlenswert zum vertieften Verständnis von netlink ist.

3.2.3.1 ADDR-Family

ADDR-Nachrichten werden verwendet, um Layer 3 (v.a. IPv4)-Adressen auf Netzwerkinterfaces zu manipulieren. Es stehen drei verschiedene Nachrichten zur Verfügung:

- RTM_NEWADDR: Fügt eine neue Adresse einem Interface hinzu.

- RTM_DELADDR: Löscht eine bestehende Adresse von einem Interface.
- RTM_GETADDR: Liefert sämtliche Informationen zu einer bestehenden Adresse.

Sämtliche Nachrichten verfügen als zusätzliche Headerinformationen über einen struct ifaddrmsg, welcher die nötigen Attribute für die Adresskonfiguration enthält:

Listing 3.4: struct ifaddrmsg

```
struct ifaddrmsg
{
        unsigned char ifa_family;
        unsigned char ifa_prefixlen;
        unsigned char ifa_flags;
        unsigned char ifa_scope;
        int ifa_index;
}
```

ifa_family Adressfamilie. Meistens AF_INET (IPv4) oder AF_INET6 (IPv6).

ifa_prefixlen Bitlänge der Subnetzmaske.

ifa_scope Gültigkeitsbereich der Adresse. IFA_SCOPE_HOST gilt nur lokal, während IFA_SCOPE_UNIVERSE globale gültig ist.

ifa_index Indexnummer[2]des Netzwerkinterfaces.

Die eigentlichen Nutzdaten müssen jeweils in einer separaten rtattr-Struktur untergebracht werden. Folgende Tabelle zeigt die wichtigsten Routing-Attribute, welche in dieser Family gesetzt werden können.

rtattr-Typ	**Beschreibung**	**Datentyp**
IFA_ADDRESS	Interface-Adresse	Adresswert
IFA_LOCAL	Lokale Adresse	Adresswert
IFA_LABEL	Interface-Name	String

Tabelle 3.2: RTNetlink Interface Address (IFA)-Typen

3.2.3.2 ROUTE-Family

ROUTE-Nachrichten sind in der Lage die IPv4-Routingtabelle zu manipulieren. Ähnlich der ADDR-Famlie können innerhalb dieser RTNetlink-Familie folgende Nachrichten versendet werden:

- RTM_NEWROUTE: Fügt eine neue Route der Routingtabelle hinzu. Benötigt bereits erwähntes NLM_F_ACK-Flag aus Abschnitt 3.2.1.
- RTM_DELROUTE: Äquivalent zu RTM_DELADDR.

[2]Die Indizes können mit dem Kommando 'ip addr' abgefragt werden. Interfaces sind aus uns unbekannten Gründen selten schön aufsteigend nummeriert!

- RTM_GETROUTE: Äquivalent zu RTM_GETADDR.

Obige Nachrichten verwenden stets die Struktur rtmsg, welche die Headerinformationen zu einem Routing-Eintrag enthalten:

Listing 3.5: struct rtmsg

```
struct rtmsg
{
        unsigned char rtm_family;
        unsigned char rtm_dst_len;
        unsigned char rtm_src_len;
        unsigned char rtm_tos;

        unsigned char rtm_table;
        unsigned char rtm_protocol;
        unsigned char rtm_scope;
        unsigned char rtm_type;

        unsigned int  rtm_flags;
}
```

rtm_family Routing-Adressfamilie. Siehe ifa_family in Abschnitt 3.2.3.1.

rtm_dst_len Länge der Zieladresse des Routingeintrags. Z. Bsp. 16 für eine Class-B Adresse.

rtm_src_len Länge der Quelladresse.

rtm_tos Type of Service der Route.

rtm_rtm_table Routingtabelle, welche manipuliert werden soll. Empfohlen wird entweder die Haupttabelle oder sämtliche Tabellen.

- RT_TABLE_UNSPEC: Alle Tabellen
- RT_TABLE_DEFAULT: Standardtabelle
- RT_TABLE_MAIN: Haupttabelle
- RT_TABLE_LOCAL: Lokale Tabelle

rtm_protocol Routing-Protokoll, welches benutzt wird. Mögliche Werte sind hier:

- RTPROT_UNSPEC: Alle Routing-Protokolle
- RTPROT_REDIRECT: Internet Control Message Protocol (ICMP) Redirect-Route
- RTPROT_KERNEL: Vom Kernel gesetzte Route
- RTPROT_BOOT: Während Bootvorgang gesetzte Route
- RTPROT_STATIC: Vom Userspace gesetzte Route (D. h. entweder per Netlink-Application Programming Interface (API) oder ip-Shellkommando)

rtm_scope Gültigkeitsbereich der Route. RT_SCOPE_UNIVERSE gilt als globale Route.

rtm_type Typ der Route. Z. Bsp. Unicast (RTN_UNICAST) oder Broadcast (RTN_BROADCAST).

Wie in der ADDR-Family werden auch hier, wie bereits erwähnt, die Nutzdaten mittels Makros an rtattr-Strukturen angehängt (Siehe Abschnitt 3.2.2). Die wichtigsten Payload-Typen werden in der folgenden Tabelle dargestellt und erläutert:

rtattr-Typ	**Beschreibung**	**Datentyp**
RTA_DST	Zieladresse	Adresswert
RTA_SRC	Quelladresse	Adresswert
RTA_GATEWAY	Next-Hop Router	Adresswert
RTA_IIF	Index des eingehenden Interface	Integer
RTA_OIF	Index des ausgehenden Interface	Integer
RTA_PREFSRC	Alternative Quelladresse[3]	Adresswert

Tabelle 3.3: RTNetlink Netlink Routing Attribute (RTA)-Typen

[3] Um eine Source-Route zu setzen, muss dieses Attribut und nicht RTA_SRC verwendet werden.

3.3 User Mode Linux (UML)

UML ermöglicht die Ausführung mehrere virtuellen Linux-Kernel auf einem einzelnen Linuxsystem. Jede virtuelle Maschine wird als eigener Benutzerprozess ausgeführt und teilt somit sämtliche Hardwareressourcen sowohl mit dem Hostsystem als auch mit den anderen virtuellen Maschinen. Somit kann auf den virtuellen Maschinen gearbeitet werden, ohne das Wirtsystem zu verändern oder zu gefährden.

3.3.1 strongSwan UML Testsuite

Eine lauffähige Testumgebung wurde von zwei Studenten der Zürcher Fachhochschule Winterthur während ihrer Diplomarbeit entwickelt. Diese Testsuite ermöglicht automatisiertes Testen mittels spezifischer UML-Testfälle in einer virtuellen Netzwerkumgebung. Weiterführende Informationen sind oben genannter Diplomarbeit[EM04] zu entnehmen.

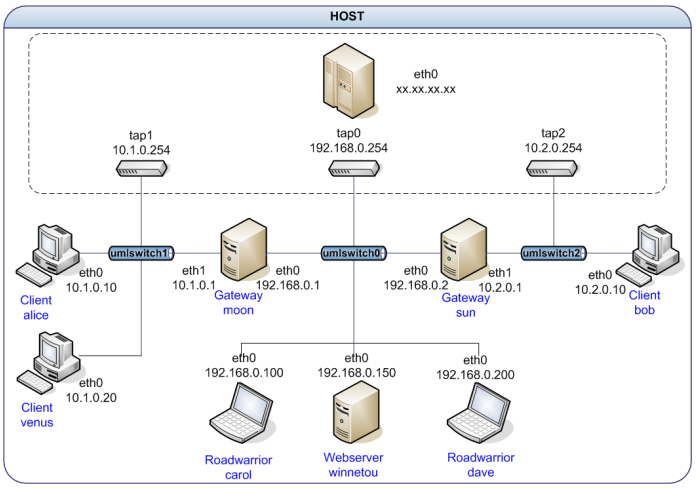

Abbildung 3.4: strongSwan UML Testsuite Architektur

Kapitel 4

Design

4.1 Methodik

Da der Quellcode von strongSwan II bereits schon einen grossen Umfang angenommen hatte, mussten wir unsere Erweiterungen komplett an die bereits bestehenden Codestrukturen anpassen. Bei Designproblemen stand somit vor allem eine möglichst optimale Integration in die bereits bestehenden Klassen und der Nutzung, der zur Verfügung gestellten Funktionalitäten.

4.1.1 Programmierrichtlinien

Die Programmierrichtlinien waren grösstenteils von Jan Hutter und Martin Willi bereits vorgegeben. Zum Studium dieser sei auf [JH05] verwiesen. Erwähnenswert ist allerdings der, in strongSwan II verwendete, objektorientierte Ansatz mit der Programmiersprache C. Durch `structs` (Klassen) voller Funktionszeiger (Methoden) wurde dies erreicht. Daraus folgt die explizite Paramaterübergabe von `this` beim Methodenaufruf. Auf Vererbung wurde dabei verzichtet.

4.1.2 Source Code Management

Der Source Code für Virtual IP wurde als eigenständiger Branch auf dem strongSwan-eigenen Subversion-Repositoryserver verwaltet. Das Backup wurde von M. Willi übernommen und der Zugriff via Secure Shell (SSH)-Keys auf die beiden Autoren dieser Arbeit beschränkt. Der Funktionsumfang von Subversion (SVN) ermöglichte ein angenehmes Merging und parallele Entwicklung von Sourcecode und Dokumentation. Nicht zu vernachlässigen dabei ist natürlich der Aspekt der Datensicherung, welche ebenfalls durch das Repository übernommen wurde.

4.2 Verbindungsaufbau (IKE-AUTH)

Innerhalb der IKE-AUTH-Transaktion, welche bereits verschlüsselt abläuft, wird der CP vom Roadwarrior mitgeliefert, um anzuzeigen, dass er eine Virtual IP erhalten möchte. IKE-AUTH ist dafür zuständig die Authentisierung der beiden Endpunkte sicherzustellen und eine mögliche erste Child Security Association

(SA) zu konfigurieren. Für detailierte Informationen über IKE-AUTH sei hier auf die StrongSwan2 Arbeit[JH05] verwiesen.

4.2.1 Client-Request

In der Methode `get_request` wird die CP mit den Attributen IP-Adresse und Subnetzmaske dem IKE-AUTH-Paket hinzugefügt. Da der Roadwarrior seine Virtual IP noch nicht kennt, werden Traffic-Selector mit der Netzadresse 0.0.0.0 mitgeschickt, damit der Server den Request bei der Aushandlung der Traffic-Selector akzeptiert.

4.2.2 Server-Response

Der VPN-Gateway beantwort den erhaltenen Request in der Methode `get_response` und liefert Daten für die angefordeten Attribute, falls diese vom Server unterstützt werden. Momentan wird lediglich die Virtual IP und die Subnetzmaske unterstützt. Die Traffic-Selector werden direkt auf die Sourceadresse des Roadwarriors eingetragen und im Antwortpaket mitgeschickt. Falls Attribute verlangt werden, welche der Server nicht kennt, werden diese der CP_REPLY nicht angehängt, was ganz dem Sinne des RFCs[CK05] entspricht.

4.2.3 Client-Conclude

Der Client reagiert auf die Antwort des Servers mit der Registrierung der IP-Adresse sowohl in der Child SA als auch im Kernel via RTNetlink. Durch den Aufruf des Kernel-Interfaces wird Letzteres bewerkstelligt. Dieser wurde in der Methode `install_child_sa` untergebracht, damit sichergestellt wurde, dass die Virtual IP erst nach vollständiger Erzeugung der Child SA gesetzt wird.

Virtual IP und Subnetzmaske werden jeweils in der entsprechenden Child SA gespeichert, damit diese Informationen beim Löschen, welches während dem Verbindungsabbau geschieht, verfügbar sind.

4.3 Verbindungsabbau (Delete IKE-SA)

Das Entfernen einer IKE-SA wird durch den Delete-Payload bewerkstelligt. Falls ein bestehender Tunnel heruntergefahren werden soll, wird genannter Payload an dessen Peer gesendet, welcher eine allfällig vorhandene Virtual IP aus dem Kernel löscht. Der Aufruf des Kernel-Interfaces befindet sich dabei in der neu erzeugten Methode `delete_virtualip`.

4.4 Kernel-Interface

Das Kernel Interface ist zuständig (wie der Name schon sagt) für sämtliche Kommunikation mit dem Kernel.

4.4.1 API-Erweiterungen

In diesem Abschnitt wird erklärt, was die neuen Methoden der Klasse `kernel_interface` an Funktionalität bieten und wie sie zu verwenden sind.

4.4.1.1 IP-Adresse hinzufügen

Listing 4.1: Methode configure_new_virtualip

```
status_t (*configure_new_virtualip)(
                        kernel_interface_t* this,
                        host_t* virtual_ip,
                        host_t* virtual_ip_subnet,
                        linked_list_t* other_ts,
                        host_t* vpn_gateway
                        );
```

`this`	Zeiger auf das aktuelle Objekt.
`virtual_ip`	Die neu hinzuzufügende Adresse.
`virtual_ip_subnet`	Subnetgrösse der neu hinzugefügten Adresse.
`other_ts`	Liste aller Zielnetze für die diese Adresse verwendet werden soll.
`vpn_gateway`	VPN Gateway hinter dem sich die angegebenen Zielnetze befinden.
return	Der retournierte Status ist im Erfolgsfall SUCCESS. Alles andere sind Fehler.

Tabelle 4.1: Signaturbeschreibung der Methode configure_new_virtualip

4.4.1.2 IP-Adresse löschen

Listing 4.2: Methode del_ipaddr

```
status_t (*del_ipaddr)(
                        kernel_interface_t* this,
                        host_t* dst_gateway,
                        host_t* ip_address,
                        host_t* subnet_mask
                        );
```

Die Methode `del_ipaddr` löscht eine hinzugefügte IP-Adresse. Durch den Kernel werden automatisch auch alle Routen gelöscht, welche diese IP als Source gesetzt hatten. Wurde die Adresse mehrfach mit `configure_new_virtualip` hinzugefügt, so muss sie auch mehrfach wieder gelöscht werden.

`this`	Zeiger auf das aktuelle Objekt.
`dst_gateway`	Next-Hop Gateway. (Wird verwendet, um das Interface ermitteln zu können.)
`ip_address`	Adresse welche gelöscht werden soll.
`subnet_mask`	Subnetgrösse der zu löschenden Adresse.
return	Retournierte Status ist im Erfolgsfall SUCCESS. Alles andere sind Fehler.

Tabelle 4.2: Signaturbeschreibung der Methode del_ipaddr

4.4.2 Interna

Falls Sie das `kernel_interface` nur verwenden, nicht aber daran weiterentwickeln wollen, so können sie diesen Abschnitt problemlos überspringen. Andernfalls enthält folgender Abschnitt wichtige Infos über die Funktionsweise unserer Erweiterungen.

4.4.2.1 RTNetlink Socket

Neuer Socket

Bisher verfügte das Kernel-Interface über einen XFRM-Netlink-Socket zur Policy-Konfiguration der IPSec-Tunnels. Neu haben wir einen RTNetlink-Socket der uns erlaubt IP-Adressen und Routen zu konfigurieren. Um das zu erreichen, haben wir die alte Objektvariable `socket` nach `xfrm_socket` umbenannt. Zusätzlich besteht nun aber noch die Variable `rt_socket`, die den neuen Socket enthält.

Pakete empfangen

Damit auch von beiden Sockets Pakete entgegengenommen werden können, haben wir die Methode `receive_messages` nach `receive_xfrm_messages` umbenannt und einen neue Methode `receive_rt_messages` realisiert. Beide Methoden werden im Konstruktor in einem eigenen Thread gestartet und terminieren bei Programmende. Um mehrfachen Code so stark wie möglich zu vermeiden, haben wir die Methode `netlink_package_receiver` erstellt, welche von oben beschriebenen Methoden aufgerufen wird.

Pakete senden

Um das Senden von Paketen erlauben zu können, haben wir in der Methode `send_message` einen neuen Parameter für den Socket eingeführt.

4.4.2.2 Nexthop-Adresse abfragen

Wir haben eine private Methode `get_nexthop` eingeführt. Diese gibt aufgrund einer Ziel-Adresse das Interface aus über welches ein Paket den Rechner verlassen würde, sowie den ersten Router via, zu dem das Paket geschickt würde.

Verwendung

`this`	Zeiger auf das aktuelle Objekt.
`dst_ip`	Adresse die überprüft werden soll.
`gwip`	Der Inhalt dieser Variable ist egal. Die Methode braucht diesen Parameter blos als zusätzlichen Rückgabewert. Die IP des Next-Hops wird an die übergebene Stelle geschrieben. Ist das Ziel im selben Subnet, so wird das Ziel als Next-Hop ausgegeben.
`outif`	Auch `outif` ist ein zusätzlicher Rückgabewert. In ihm steht der Interface-Index des Greäts überwelches ein Paket zu `dst_ip` den Rechner verlassen würde. Gehört die Adresse dem eigenen Rechner, so wir das lo Interface zurückgegeben.
return	Der retournierte Status ist im Erfolgsfall SUCCESS. Alles andere sind Fehler.

Tabelle 4.3: Signaturbeschreibung der Methode get_nexthop

Weshalb wird dies benötigt?

Die Methode wird aus zwei Gründen gebraucht: Der Erste ist, dass wir beim Eintragen einer neuen Adresse wissen müssen, an welches Interface wir sie binden. Wir binden sie immer an aktuelle ausgehende Interface. Der zweite Grund sind die Source Routen. Diese werden benötigt um festzustellen, für welche Zielhosts, welche Absender-Adresse verwendet wird. Da diese Einträge aber normale

Routen sind, muss immer ein `via` mitgegeben werden. Dieses `via` wird auch von `get_nexthop` mitgeliefert.

4.4.2.3 Mehrfach eingetragene Adressen

Eine Adresse kann einmal pro Gerät eingetragen werden. Würden nun zwei Verbindungen die selbe Adresse verwenden, so würde die Verbindung die zuerst heruntergefahren wird der anderen die Adresse löschen. Deshalb haben wir einen Referencecounter eingeführt. Das heisst, wenn zweimal `configure_new_virtualip` aufgerufen wird, muss danach auch zweimal `del_ipaddr` aufgerufen werden, damit die Virtual IP auch entfernt wird.

4.4.2.4 IPv6-Unterstützung

Das Kernelinterface bietet vollen Support sowohl für IPv4 als auch IPv6. Die Kapselung der IP-Adressen in `host_t`-Objekte erleichterte die Programmierung zumindest ausserhalb des Kernelinterfaces. Aufgrund der Tatsache, dass momentan nur gerade ein Testfall existiert, welcher IPv6-Kompatiblität überprüft, konnte implementierter IPv6-Code nicht detailiert auf dessen Funktionalität getestet werden.

4.5 Konfiguration und Nutzung

Um Virtual IP mit IKEv2 unter charon einsetzen zu können, muss man in der Konfigurationsdatei von strongSwan, in `ipsec.conf`, bestimmte Einstellungen vornehmen. Hier sind die zwei verschiedenen Szenarien besprochen, in welchen Virtual IP eingesetzt wird. Dazu werden die jeweiligen Beispielkonfigurationen mitgeliefert. Für mehr Informationen über die Konfiguration von strongSwan sei auf dessen Webseite verwiesen.

4.5.1 Statische Virtual IP

Wie in pluto, dem IKEv1-Daemon, ist es möglich auf dem Client eine fixe Virtual IP zu definieren. Mittels Parameter `leftsourceip` und der Zuweisung einer eindeutigen, virtuellen Adresse kann Virtual IP aktiviert werden. Auf dem VPN-Gateway muss `leftsubnetwithin` auf das virtuelle Netz zeigen, in welchem sich die virtuellen IP-Adressen der Roadwarriors befinden. Auf dem VPN-Gateway muss `leftsubnetwithin` auf das virtuelle Netz zeigen, in welchem sich die virtuellen IP-Adressen der Roadwarriors befinden.

Folgendes Listing zeigt eine entsprechende Beispielkonfiguration:

Listing 4.3: Statische Virtual IP Konfiguration

```
# roadwarrior ipsec.conf

conn home
        leftsourceip=10.3.0.1

# vpn-gateway

conn rw
        leftsubnetwithin=10.3.0.0/16
```

4.5.2 Dynamische Virtual IP via CP

Durch die Configuration Payload3.1 kann der VPN-Gateway seinen Roadwarriors eine IP-Adresse aus dem Configfile übermitteln. Diese wird auf dem Gateway durch den Configparameter `rightsourceip` gesetzt. Damit der Client eine Virtual IP anfordert, muss in dessen Konfiguration `leftsourceip` auf %pool[1]

Listing 4.4: Dynamische Virtual IP Konfiguration

```
# roadwarrior ipsec.conf

conn home
        leftsourceip=%pool

# vpn-gateway

conn rw
        rightsourceip=10.3.0.1
        rightid=rw@strongswan.org
```

[1] Wird nur von charon, sprich IKEv2 unterstützt.

Kapitel 5

Tests

Wir haben primär von Hand getestet. Das heisst wir haben übersetzt, installiert und gestartet. Wenn es funktioniert hat, waren wir zufrieden. Um genügend Rechner zu haben, haben wir die UML-Testumgebung[EM04] verwendet. Genauere Informationen zu dieser Testsuite findet man auf strongswan.org. Hier sei nur so viel gesagt, es handelt sich um eine Simulation eines ganzen Netzwerkes indem VPN-Verbindungen aufgebaut werden können. Martin Willi hat ein Parameter im Configfile der Testsuite eingeführt, mitdessen Hilfe man Verzeichnisbäume aus dem Hostsystem in die virtualisierten Systeme mounten kann. Dadurch konnten wir unseren Code in der virtuellen Maschine übersetzen.

Wir wollten unseren Code auch gegen das OpenIKEv2 Projekt testen. Da wir das aber nicht in vernünftiger Zeit zum Laufen gebracht haben, liessen wir das aus Zeitgründen aus.

Auch für einen eigenen automatischen UML-Test reichte die Zeit nicht.

Kapitel 6

Projektfortsetzung

6.1 Erweiterungsmöglichkeiten

Im Anschluss an diese Studienarbeit können folgende Erweiterungen in Angriff genommen werden:

- Implementierung eines IP-Pools auf Serverseite.
- Unterstützung von DNS und WINS-Serveradressen.
- Delegation des IP-Pools an einen LDAP/RADIUS-Server.

6.2 Verbesserungsvorschläge

Das strongSwan Projekt hat mit Charon ein ziemlich spezielles Programmiermodell gewählt. Ein paar Ideen zur Verbesserung der Produktivität und des Programmierkomforts werden hier näher ausgeführt.

6.2.1 Lesbarkeit des Quellcodes

Die Lesbarkeit ist für wohl eine der zentralsten Qualitätsmerkmale einer Software. Wir sind der Meinung, dass der Code um Einiges lesefreundlicher und verständlicher gestaltet werden könnte.

Methodengrösse

Als ersten Verbesserungsvorschlag könnte man die Länge einer einzelnen Methode auf z. Bsp. 50 Zeilen beschränken. Zusätzlich sollten die Methoden über einen Namen verfügen, welcher ganz klar aussagt, was deren Inhalt ist.

Kohäsion

Charon besteht aus wenigen sehr mächtigen Klassen. So verwaltet das Kernelinterface 2 Referencecounter, 2 Netlink Sockets, baut Tunnels auf und ab, fügt IP Adressen und Routen hinzu usw. Jede dieser Aufgaben müsste laut Design-Prinzipien in eine eigene Klasse ausgelagert werden. So wäre es z.B. sinnvoll eine Klasse Netlink zu machen, von der die 2 Klassen RTNetlink und XFRMNetlink erben (Vererbung wäre implementierbar). Das Kernelinterface könnte Instanzen der Beiden verwenden. Klassen mit nur einer wohldefinierten Aufgabe würden das Verständnis fördern.

6.2.2 Programmierunterstützung

Code Generierung

Ein möglicher Grund dafür, weshalb viele Klassen extrem gross sind, ist wohl die aufwendige Erzeugung einer solchen. Ideal wäre, wenn z.B. ein neues Codewort (public) eingeführt werden würde, mithilfe dessen ein kleines Perl Programm den Konstruktor und das Headerfile selber schreiben könnte. Es würden weniger Fehler entstehen, und man müsste sich weniger vor neuen Klassen fürchten.

Mehr Stack

Zur Zeit ist es nicht möglich ein Objekt auf den Stack zu legen. Der Stack hat aber klare Vorteile: Er wird zuverlässig abgeräumt. Deshalb sollte es möglich sein, z.B. indem man dem Konstruktor einen Zeiger mitgibt, ein Objekt auf dem Stack zu halten. Wie man allerdings garantiert, dass der Destruktor ausgeführt wird, ist auch uns nicht bekannt.

Autovervollständigung

Als IDE wurde KDevelop verwendet. Leider unterstützt KDevelop die OO-C-Programmierung nur teilweise und bietet daher auch nur sehr wenig bis gar keine Autovervollständigung von Programmoperationen. Vielleicht ist es möglich mittels KDevelop-Plugin Genanntes zu erreichen.

6.2.3 Automatisierte Tests

Die UML-Tests stellen sicher, dass ein Release fehlerfrei publiziert werden kann. Während der Entwicklungsarbeit, sind sie aber zu aufwendig. Darum wäre es sinnvoll, wenn einige einfache Tests, die oft Fehler verursachen, schnell durchgeführt werden könnten.

Redundanz

Ein Beispiel dafür ist dass der Methodenheader, der einmal im Konstruktor bei casten, einmal im Headerfile und ein weiteres Mal in der Implementierung vorkommt. Ein Test könnte überprüfen, ob die Parameter dabei immer übereinstimmen und diese möglicherweise gleich synchronisieren.

Schnelle Tests

Zusätzlich wären auch kleinere, in kurzer Zeit ausführbare Unit-Tests von Vorteil. Damit könnten Fortschritte beim Programmieren einiges schneller und leichter auf dessen Funktionsweise überprüfen..

Autotests

Bereits verfügbar sind die UML-Tests. Diese sollten jede Nacht für jeden Branch automatisch ausgeführt werden, und bei Fehlern umgehend die betroffenen Entwickler z.B. per Mail informieren.

6.2.4 Installationsskripts der UML-Testumgebung

Die zweifellos komfortable UML-Testumgebung[EM04] unterstützt den Entwickler enorm bei Testen neuer Softwarekomponenten. Innerhalb der Installationsroutinen haben sich dann aber einige Fehler eingeschlichen, welche durch die etwas optimistische Skriptingweise nur schwer erkannt werden konnten. Weil die Installationsskripts nach jeder Ausführung eines Programms jeweils mit der Ausgabe `Done!` abschliessen (auch im Fehlerfall!), erfährt der Benutzer nur selten, ob und vor allem was schief gegangen ist.

Anhang A

Projektmanagement

A.1 Arbeitspakete

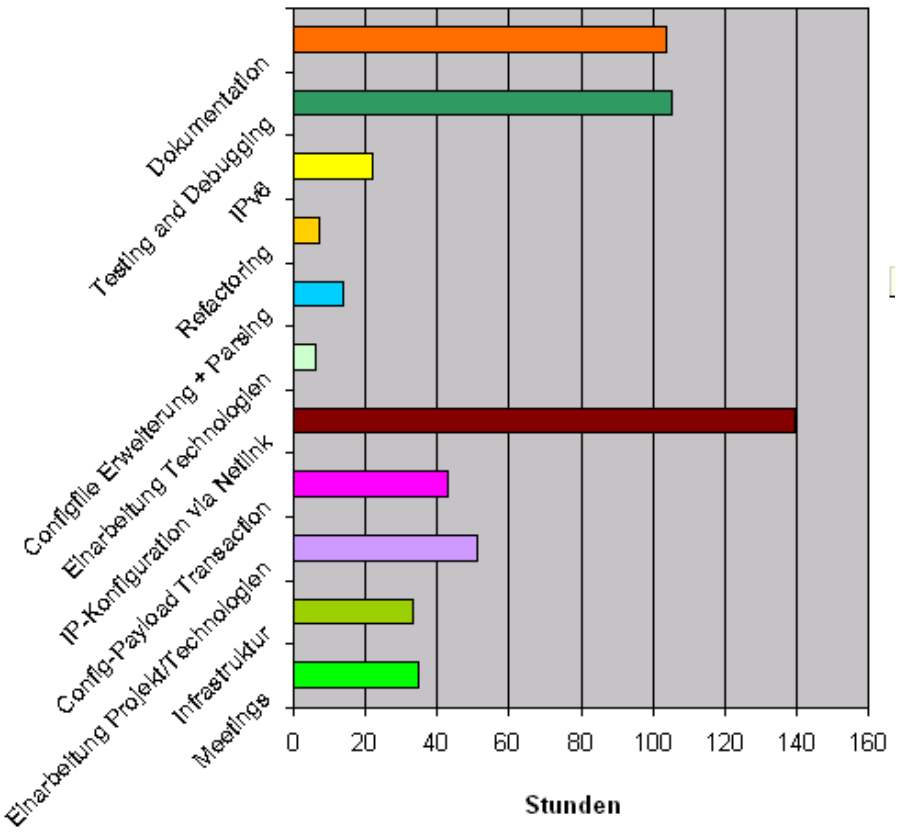

Abbildung A.1: Stunden nach Paketen

A.2 Zeitabrechnung

A.2.1 Gesamtaufwand

Wer	$\sum$	$\varnothing$
Fabian Hartmann	290	20.7
Noah Heusser	270	19.3
Min. Modulbeschreibung	196	14.0

Tabelle A.1: Aufwand

A.2.2 Aufwand nach Wochen

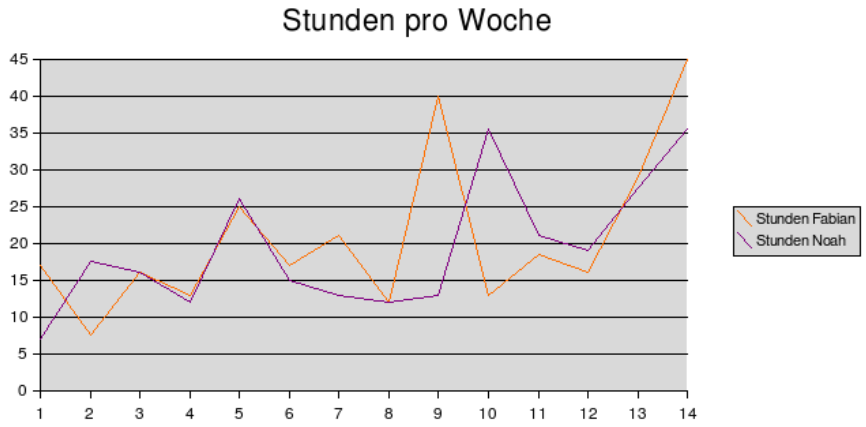

Abbildung A.2: Stunden pro Woche

A.3 Qualitätssicherung

A.3.1 Tests

Nach jeder noch so kleinen Änderung haben wir das Programm übersetzt und manuell getestet. Der Test garantierte die grundlegende Funktonsfähigkeit des Charon Daemon auf Client und Serverseite.

A.3.2 Subversion

Um sicherzustellen dass wir jederzeit zu jedem belibigen Projektstatus zurückkehren können, haben wir mit Subversion gearbeitet. Es erlaubt uns auch bei gefunenen Fehlern nachzuvollziehen wie er entstanden ist. Wenn man das einige Male gesehen hat, so ist man sich der Fehlerquellen besser bewusst.

A.3.3 Pair-Programming

Heikle Methoden habe wir oft im Pair-Programming gelöst. D.h. einer schreibt und der andere überlegt sich genau was abläuft und wo es Fehler geben könnte. Das auftreten von Fehlern die nicht von Compailer gefunden werden (wie z.B. Speicherlekts) kann so merklich verringert werden.

A.3.4 Code Reviews

Wir absolvierten regelmässige Code Reviews indem wir den entwickelten Programmcode gegen gelesen und analysiert haben. Nebst funktionalen Problemen haben wir uns dabei vor allem aber auch auf Refactoring und gutes Design fokussiert. Somit war sichergestellt, dass Flüchtikeitsfehler nur schwer im Code bestehen blieben und eine gute Lesbarkeit gewährleistet werden konnte.

Anhang B

Erfahrungsberichte

B.1 Fabian Hartmann

Die Ausschreibung dieser Arbeit fiel mir sogleich auf, als ich mir einen Überblick über sämtliche Studienarbeiten verschaffte. Der Inhalt sprach viele meine Interessen an. Vor allem aber wollte ich der Open Source Gemeinde von Linux einen Dienst erweisen. Dass wir in C zu programmieren hatten, empfand ich anfangs als kein allzu grosses Hindernis.

Nun, nachdem diese Studienarbeit dem Ende entgegen geht, muss ich ganz klar gestehen, dass ich mich geirrt habe. Das, in den Betriebssystem-Modulen, erlernte C-Programmierwissen reichte oft nicht aus, um sich im Code vernünftig bewegen zu können. Natürlich machten wir während der gesamten Studienarbeit enorme Fortschritte und dürfen jetzt wohl ohne zu übertreiben von uns behaupten, dass wir auch in C professionelles Software-Engineering betreiben können.

B.1.1 Angenehme Betreuung

Martin Willi konnte uns durch sein Fachwissen und seiner Erfahrung einige sehr gute Tips geben und in die Architektur von charon einführen. Herr Steffen stand uns zusätzlich zur Seite, als es um strongSwan-generelle Dinge und unsere Ziele ging. Ich lernte ihn dabei als sehr gemütlichen, angenehmen Menschen kennen, der sehr viel von dem versteht, was er unterrichtet und entwickelt. Ich schätzte diese Art und es verhalf sicher zu einem gesunden Arbeitsklima. Allerdings war es für uns oft fast unmöglich herauszufinden, wo wir mit unserem Projekt standen. Etwas mehr klares Feedback hätte uns sicher nicht geschadet, sondern eher noch zusätzlich angespornt.

B.1.2 Erfolgreiche Zusammenarbeit

Die Zusammenarbeit mit Noah Heusser verlief während den gesamten 14 Wochen problemlos. Wir ergänzten uns vielleicht nicht optimal, da wir uns gegenseitig nicht immer mitreissen vermochten, um so das Projekt noch stärker vorantreiben zu können. Trotzdem empfand ich es als angenehm mit Noah zusammenarbeiten zu können. Er überraschte mich, der mich selbst auch als

überzeugter Linuxler bezeichnet, immer wieder durch sein enormes Fachwissen, insbesondere im Linux-Bereich.

B.1.3 Auch Linux-Entwickler kochen nur mit Wasser

Durch das Studium von RTNetlink bedingte Durchforsten der Linux-Kernel-Internas entpuppte sich für mich als eine weitere Erfahrung, die ich nicht missen möchte. Ich erkannte, dass auch sämtliche Linux-Hacker, welche an Kernel und Netlink rumbasteln, auch nur Menschen sind und keine übernatürlichen Gurus. Auch diese Leute nutzen (leider) die so verführerische `goto`-Funktion oder schreiben Code, welchen jeglichen, in den SE-Modulen erlernten, Designprinzipien widersprechen.

B.1.4 LaTeX ist eine Wucht

Noah konnte mich zu Beginn der SA davon überzeugen die Dokumentation in LaTeX zu verfassen. Ich hatte auch genug von Word und Openoffice und wollte vor allem auch mal etwas Neues kennenlernen. Auch wenn es zusätzlichen Aufwand bedeutete, da wir beide noch Neulinge auf diesem Gebiet waren, stellte sich dies als die richtige Entscheidung heraus. Ich bin während der SA zu einem riesigen Fan von LaTeX geworden und werde sicher auch weitere Arbeiten ebenfalls damit verfassen. Klar gibt es auch hier gewisse Tücken und Macken gegenüber Word oder Openoffice, aber ist das grobe Konzept einmal begriffen, kann mit diesem mächtigen Werkzeug prächtige Textdokumente erzeugen.

B.1.5 Erwartungen nicht ganz erfüllt

Obschon wir extrem viel Zeit in diese Studienarbeit investiert haben, kann ich nicht ganz zufrieden mit dem Resultat sein. Wir haben uns anfangs ganz klar höhere Ziele gesteckt wie z. Bsp. die Implementation eines IP-Pools. Leider erfüllten wir lediglich das Minimalziel. Mögliche Gründe dafür sind möglicherweise das ungenügende C-Fachwissen oder die Tatsache, dass wir uns ganz einfach überschätzt haben. Vermutlich war es letztendlich eine Mischung aus Beidem und die Tatsache, dass Software-Engineering in der Programmiersprache C einfach ziemlich mühsam ist.

B.2 Noah Heusser

Ich kann sagen, dass ich enorm viel gelernt habe. Man könnte sogar sagen, dass es etwas zuviel war.

B.2.1 C-Programmierung $\neq$ C-Software Engineering

Bevor ich mit der Aufgabe anfing, bin ich von Herr Steffen darauf hingewiesen worden, dass ich noch nie mit C gearbeitet habe. Er fragte, ob ich mir das denn zutrauen würde. Für mich war die Antwort klar: „Ja!!!“. Heute weiss ich, es geht nicht nur darum die Sprache zu kennen. Dies ist einfach. Es geht auch darum zu verstehen wie Software Engineering mit ihr betrieben wird. D.h. welche Debugger gibt es und wie sind diese zu benutzen. Welche speziellen Probleme tauchen bei dieser Sprache auf usw.. Damit habe ich zeitintensiv gekämpft. Besondere Probleme hatte ich in den folgenden Punkten:

- Debugging
- Auffinden von Speicherlecks und Doublefrees
- Unabhängiges Testen einzelner Komponenten
- Zusammenhänge im Make-Baum verstehen

B.2.2 Fremden Code lesen ist hart

Leseschwierikeiten

Zum ersten mal in meiner Informatiklaufbahn, musste ich in diesem Projekt viel fremden Code verstehen. Dies zum einen, weil ich mich in Charon einarbeiten musste und zum Andern, weil die beste Anleitung für RTNetlink der iproute2 Quellcode ist. Für mich musste ich feststellen, dass ich mehr Mühe hatte fremden Code zu analysieren als Fabian.

Die Lücke

Da ich das Lesen von Code als eine der wichtigsten Tätikeiten eines Programmierers empfinde, bin ich da auf eine Lücke gestossen, die ich unbedingt füllen will. Deswegen habe ich mir auch ein Buch zu dem Thema gekauft. Auch wenn ich noch nicht sehr weit gekommen bin, so habe ich doch schon einige Teile daraus für mich nutzen können. Das äussert sich dahingehend, dass ich heute nicht mehr vor einem grossen Knäuel Code stehe, sondern gleich mit einer Selektion beginnen kann.

Das Modul

Die HSR wird vermutlich mit Herr Sommerlad für die Master-Studenten ein Code-Reading Modul anbieten.

B.2.3 Subversion, das Unverzichtbare

Subversion ist ein Genuss!!! Auch wenn ich die Vorzüge von SVN gegenüber Concurrent Versioning System (CVS) noch nicht gespürt habe, würde ich in Zukunft auf SVN setzen. Dies, weil ich keine Nachteile sehe und ich mir die Vorteile als Projektmanager (z.B. move) angenehm vorstelle. Nur hätte ich den SVN Server gerne von zu Hause aus abgerufen ohne VPN.

Jedoch egal ob SVN oder CVS, Software Engineering ohne Versionierungstools scheint mir unmöglich.

B.2.4 LaTeX ist cool

Für unsere Dokumentation haben wir mit LaTeX gearbeitet. Dies weil wir ein American Standard Code for Information Interchange (ASCII) Format haben wollten, indem wir ohne Probleme via SVN zusammenarbeiten können. Zusätzlich hat LaTeX den Vorteil, dass man genau sieht, was man macht. Es gibt keine unsichtbare Formatierung aus vergangenen Formatierungsaktionen. Man kann sehr einfach für alles einen eigenen Tag definieren und hat für das ganze Dokument eine perfekte Corporate Identity.

Die Entscheidung

Leider hat LaTeX aber auch ein paar Tücken, die man erst kennen lernen und umschiffen muss. Wir, die unsere Dokumentation in den letzten 2 Wochen schreiben *peinlich* haben viel geschwitzt, um das zu erreichen, was wir wollten.

Nicht perfekt

Als Fazit kann ich sagen die nächste Arbeit wird wieder LaTeX oder vielleicht DocBook sein, aber sicher nicht OpenOffice oder Word. Es ist aber auch nicht perfekt.

B.2.5 Die Arbeit hat mich stark gefordert

Meine Erwartungen an mich waren ziemlich hoch. Ich sehe auf unser Resultat und hätte mehr erwartet. Nur war der Einsatz eigentlich da. In einer kommenden Arbeit werde ich meine Einschätzung vorsichtiger vornehmen. Weiterhin finde ich das Tool strongSwan cool.

Abkürzungsverzeichnis

IPSec Internet Protocol Security

IKE Internet Key Exchange Protocol

NAT Network Address Translation

VPN Virtual Private Network

DNS Domain Name System

IP Internet Protocol

WINS Windows Internet Name Service

CP Configuration Payload

UML User Mode Linux

SA Security Association

ITA Institute für Internet Technologien und Anwendungen

RFC Request for Comment

HSR Hochschule für Technik Rapperswil

RADIUS Remote Authentication Dial-In User Service

DHCP Dynamic Host Configuration Protocol

SVN Subversion

CVS Concurrent Versioning System

SSH Secure Shell

API Application Programming Interface

ASCII American Standard Code for Information Interchange

IFA Interface Address

ICMP Internet Control Message Protocol

RTA Netlink Routing Attribute

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literaturverzeichnis

[CK05] Ed. Microsoft C. Kaufmann. Ikev2. Rfc, IETF, 12 2005.

[EM04] Patrick Rayo Eric Marchionni. User-mode-linux testsuite für linux strongswan. Diplomarbeit, Zürcher Hochschule Winterthur, 2004.

[Hor04] Neil Horman. *Understanding and Programming with Netlink Sockets*, version 0.3 edition, 12 2004.

[JH05] Martin Willi Jan Hutter. strongswan 2 - eine ikev2 implementierung für linux. Diplomarbeit, Hochschule für Technik Rapperswil, Institut für Internet-Technologien und Anwendungen, 12 2005.

[rtn99] *RTNETLINK(7) Manpage*, 4 1999.

[SK98] R. Atkinson S. Kent. Security architecture for the internet protocol. Rfc, IETF, 11 1998.

[TB06] Daniel Röthlisberger Tobias Brunner. Nat traversal for strongswan 2. Studienarbeit, Hochschule für Technik Rapperswil, Institut für Internet-Technologien und Anwendungen, 7 2006.