Diplomarbeit
zum Thema:
IPSec Infrastruktur für Mikro-SINA

Jens Syckor
Technische Universität Dresden
Fakultät Informatik
Institut für Systemarchitektur
Lehrstuhl für Betriebssysteme

8. November 2004

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Aufbau der Arbeit 2
1.2 Danksagung 2
1.3 Selbständigkeitserklärung 2

2 Grundlagen und Stand der Technik **3**

2.1 IPSec 3
2.2 Security Associations und Key Management 5
2.3 Simple Key Management for Internet Protocols (SKIP) 6
2.4 Photuris 7
2.5 ISAKMP/IKE 8
2.5.1 ISAKMP 8
2.5.2 IKE 9
2.6 Review IPSec Infrastrukturen 12
2.6.1 Windows 2000 und XP 12
2.6.2 FreeS/WAN Linux 12
2.6.3 OpenBSD 13
2.6.4 SINA 14
2.6.5 Mikro-SINA 14

3 Entwurf **17**

3.1 Entwurfskriterien 17
3.2 Wahl des Grundkonzeptes und Protokolls 18
3.3 Mikro-SINA IPSec Infrastruktur 19
3.3.1 Analyse der SA Etablierung nach Acquire-Nachricht 20
3.3.2 Analyse der initialen SA Etablierung mittels KMd 22
3.4 Kommunikation des KMd 22
3.4.1 KMd und nicht vertrauenswürdiges Netz 23
3.4.2 KMd und Viaduct 24
3.5 Mikro-SINA IPv6 28

3.5.1 Neuerungen von IPv6 28
3.5.2 Auswirkungen auf andere Protokolle und APIs 30
3.5.3 Anpassungen an L4VFS und POSIX Umgebung 30
3.5.4 Anpassungen der Mikro-SINA Komponenten 30

4 Implementierung **35**
4.1 Abbildung des BSD Socket Interfaces auf L4 36
4.2 Abbildung des synchronen IO/Multiplexing von POSIX *select* 38
4.3 Socket Umgebung für die Portierung von Linux Kernel Netzwerk Treiber 41
4.4 PF_KEY V2 als generische Schnittstelle für SAs und Policies 41
4.5 Hilfsapplikationen 43
4.5.1 L4LX_FLIPS 43
4.5.2 IPSec Admin 44

5 Evaluation **45**
5.1 Vergleich POSIX Umgebung gegenüber anderen Konzepten 45
5.2 Komplexität des Quellcodes 46

6 Zusammenfassung und Ausblick **47**

7 Glossar **49**

Kapitel 1

Einleitung

Im Rahmen des Projektes Sichere Inter-Netzwerk Architektur (SINA) des Bundesamtes für Sicherheit in der Informationstechnik (BSI) wurden 2002 die ersten Teile einer IT-Architektur veröffentlicht, die zur Verarbeitung hoch schützenswerter Informationen in unsicheren Netzen eingesetzt wird. Basis der Komponenten SINA Client und des IPSec-Gateway SINA Box ist ein gehärtetes Linux (SINA-Linux), das sehr stark minimalisiert und sicherheitstechnisch umfangreich analysiert wurde.

Am Lehrstuhl für Betriebssysteme an der TU Dresden wurde das Forschungsvorhaben Mikro-SINA initiiert, das sich zum Ziel gesetzt hat, eine mikrokernbasierte Variante der SINA Box prototypisch umzusetzen und zu evaluieren. Im Gegensatz zur Verwendung des monolithischen Linux-Betriebssystems soll die Verwendung der Mikrokern-Umgebung eine deutlich bessere Evaluierung bzgl. Datensicherheit und eine schlankere Architektur ermöglichen. Weiterhin soll die Mikrokerntechnologie eine einfachere Integration neuer Komponenten ermöglichen, die durch die Systemarchitektur des Kernes sicherheitstechnisch isoliert sind [20].

Der aktuelle Prototyp von Mikro-SINA ermöglicht den Austausch von verschlüsselten Daten via IPSec in einem Virtual Private Network (VPN) mit manuellem Schlüsselaustausch.

Das eingesetzte manuelle Verfahren ohne zusätzliches Key Management ist bei komplexeren Netzwerkstrukturen nicht flexibel genug, da der Administrationsaufwand sehr hoch ist. Außerdem ermöglichen automatische Verfahren einen unkomplizierteren Einsatz von kurzlebigen Schlüsseln, damit wird eine erhöhte Sicherheit gewährleistet. Das Ziel dieser Arbeit ist die Evaluierung der Mikro-SINA Architektur bezüglich der IP Security Infrastruktur. Es soll eine Komponente entwickelt werden, die das Management der Security Associations (SA) und der Schlüssel automatisiert.

Weiterhin soll der Einfluss von IPv6 auf Mikro-SINA untersucht werden. Ein Entwurf der Integration von IPv6-Funktionalität ist weiterer Bestandteil der Arbeit.

1.1 Aufbau der Arbeit

Das folgende Kapitel informiert über die Grundlagen dieser Arbeit und verwandte Projekte. Es werden IPSec, das Konzept der Security Associations und der Security Policy Database erläutert. Deren Kenntnis ist fundamental wichtig, um die Verfahren für automatisches Schlüsselmanagement zu verstehen, die im Anschluss erklärt werden. Danach werden kurz einige aktuelle IPSec Infrastrukturen vorgestellt, wobei auf den aktuellen Prototyp von Mikro-SINA eingegangen wird.

Im dritten Kapitel wird die IPSec Infrastruktur für Mikro-SINA entworfen und die Komponenten bzgl. Integrität, Vertraulichkeit und Verfügbarkeit untersucht. IPv6 für Mikro-SINA wird ebenfalls in diesem Kapitel analysiert und ein Entwurf diskutiert. Die auf der Basis des Infrastruktur-Entwurfes entstandene Implementierung wird in Kapitel 4 anhand ausgewählter Details näher beleuchtet. Kapitel 5 beschäftigt sich mit der Evaluation der Architektur unter besonderer Berücksichtigung der Datensicherheit. Darauf folgt eine Zusammenfassung und ein kurzer Ausblick auf resultierende neue Fragestellungen. Den Abschluss bildet ein Glossar der verwendeten Begriffe und Abkürzungen, sowie eine Verzeichnis der Quellen.

1.2 Danksagung

Ich möchte mich bedanken bei meinem Betreuer Christian Helmuth, bei Norman Feske und Alexander Warg für ihre tolle Unterstützung. Ein ganz besonderer Dank gilt meiner Frau Silke und meiner Tocher Nele.

1.3 Selbständigkeitserklärung

Hiermit erkläre ich, dass ich meine Diplomarbeit selbständig und nur mit den aufgeführten Hilfsmitteln erstellt habe.

Jens Syckor, Dresden, den 8.11.2004

Kapitel 2

Grundlagen und Stand der Technik

Wenn man das OSI-Referenzmodell (ISO 7498-1) betrachtet, ist schnell zu erkennen, dass in allen Schichten Funktionen zum Schutze der Sicherheit notwendig sind. Einige Schutzziele aber, wie z.B. Verkehrsdatenschutz, lassen sich nur in einer bestimmen Ebene realisieren. Dennoch muss immer ein Zusammenhang der Sicherheitsaspekte im gesamten Modell betrachtet werden, wobei die maximale Sicherheit vom schwächsten Teil des Systems abhängt.

Im Folgenden wird die Sicherheit im Rahmen der Netzwerkschicht (IP-Schicht) dargestellt werden. Schutzmechanismen innerhalb der IP-Schicht ermöglichen eine sehr starke Transparenz und einfache Handhabung für den Anwender, da alle Programme, die auf das Netzwerk zugreifen, keine expliziten Sicherheitseinstellungen benötigen.

Um ein allgemeines Verständnis der Mechanismen zur Sicherheit in der IP-Schicht zu vermitteln, werde ich die Schlüsselkonzepte erläutern, die Vertraulichkeit, Authentizität und Integrität ermöglichen. Weiterhin sollen aktuelle Protokolle zur automatischen Verwaltung der Sicherheitsattribute und einige Implementierungen von IPSec Infrastrukturen näher betrachtet werden.

2.1 IPSec

Der Paketdienst IPv4 (Internet Protocol) innerhalb der Netzwerkschicht ist ohne zusätzliche Schutzfunktionen grundsätzlich nicht vertrauenswürdig, da keinerlei Authentisierung und Prüfung der Integrität durchgeführt wird. Einem Angreifer ist es so relativ problemlos möglich, die Adresse des Empfängers zu ändern oder den Inhalt eines Paketes zu manipulieren. Weiterhin gibt es keinen Schutz gegen die Wiedereinspielung

von Paketen, ein solcher Angriff wird als Replay Attacke bezeichnet. Da die Daten komplett unverschlüsselt übertragen werden, kann außerdem nicht verhindert werden, dass ein Angreifer das Paket auf dem Weg zum Empfänger ausgespäht hat.

Um Vertraulichkeit und Authentizität/Integrität in der Netzwerkschicht zu gewährleisten, benötigt IPv4 die Erweiterung IPSec. Sie bietet zwei Mechanismen (Sicherheitsprotokolle): Authentication Header (AH) und Encapsulated Payload (ESP).

AH [24] dient zur Sicherung von Authentizität und Integrität sowie zum Schutz vor Replay Attacken, es sind die Nutzdaten und Teile des Headers vor Verfälschung geschützt. Jedes Paket ist durch eine kryptographische Prüfsumme gesichert, die mittels einer Hashfunktion gebildet und zusätzlich mit einem symmetrischen Schlüssel parametrisiert wurde. Eine Replay Attacke wird durch eine Sequenznummer verhindert, die in die Berechnung der Prüfsumme mit eingeht.

ESP [25] bietet Vertraulichkeit durch Verschlüsselung, sowie die Integritätssicherung der Nutzdaten. Es werden symmetrische Verschlüsselungsverfahren wie 3DES oder IDEA eingesetzt.

Für jedes Sicherheitsprotokoll wird dem IP-Paket ein zusätzlicher Header hinzugefügt, der jeweils eine unterschiedliche Struktur hat.

Mit IPSec wird nicht nur der Aufbau einer gesicherten Punkt-zu-Punkt Verbindung (Transport Mode) ermöglicht, sondern auch der Schutz ganzer Netze mit Hilfe des Tunnel-Modus. Durch VPNs können Netze hinter einem Security-Gateway verborgen werden und so über ein nicht vertrauenswürdiges Zwischennetz (z.B. das Internet) mit anderen Partnern hinter einem VPN-Endpunkt gesichert kommunizieren. IP-Pakete werden dazu in IPSec-Paketen verpackt, d.h. getunnelt, indem sie via ESP bzw. AH geschützt und mit einem neuen IP-Header versehen werden. Diese Verschachtelung ist beliebig oft kombinierbar, es muss lediglich auf eine korrekte Anordnung geachtet werden.

Ein zentraler Punkt von IPSec ist die freie Entscheidung des Senders und Empfängers, sowie der Zwischenstationen, welche Sicherheitsanforderungen sie an bestimmte Verkehrsströme stellen. Jeder Knoten implementiert seine eigenen Sicherheitsstrategien (Security Policy), anhand derer er diese Entscheidungen trifft. IPSec setzt dahingehend keinerlei Restriktionen, es gibt keine Vorgaben, welche Verfahren jeder Knoten implementieren muss. Die Konditionen und Parameter eines Sicherheitsvertrages, auch Sicherheitsassoziation (SA) [23], müssen also jeweils ausgehandelt werden. Das sind z.B. die einzusetzenden kryptographischen Verfahren, die Schlüssel und das Protokoll, weitere können in [23] nachgelesen werden.

Eine SA ist eine uni-direktionale “Verbindung”, die einen Sicherheitsvertrag zwischen zwei Partnern für einen bestimmten Verkehrsstrom festlegt. Sie ist immer auf den Empfänger gerichtet und wird durch folgendes Tupel definiert:

IP-Zieladresse, Security Parameter Index (SPI), Protokoll

Unter Protokoll ist AH oder ESP zu verstehen. Eine Verbindung, die sowohl mit AH und ESP zu schützen ist, benötigt somit zwei SAs pro Richtung. Für eine Kombination von komplexeren Sicherheitsverträgen können mehrere SAs auf eine Verbindung angewendet werden, bezeichnet wird dies als Security Association Bundle (SA bundle). Der SPI ist ein eindeutiger Identifikator für eine SA, konkret ein 32bit Wert, der in jeden IPSec-Header eingebettet wird und dem Teilnehmer als Index auf seine lokalen Datenstrukturen der entsprechenden SA dient.

Für die Verwaltung der SAs sind in jedem System zwei konzeptionelle Datenbanken vorgesehen, die Security Association Database (SADB) und die Security Policy Database (SPD).

Die SADB verwaltet die zum aktuellen Zeitpunkt aktiven SAs eines Systems, d.h. sie bezieht sich auf die kryptographischen Endpunkte [15] der Beziehung. Jeder Eintrag wird durch ein SA-Tupel eindeutig referenziert. Er enthält zusätzlich die zu verwendenden Verschlüsselungsalgorithmen, die notwendigen Schlüssel und die Gültigkeitsdauer der SA.

Die SPD definiert, ob und wenn ja welche Sicherheitsmechanismen auf ein IP-Paket anzuwenden sind. Es werden grundsätzlich drei Nachrichtenklassen unterschieden:

1. discard, d.h. das Paket verwerfen
2. bypass, d.h. das Paket wird ohne Bearbeitung weitergeleitet
3. apply IPSec, d.h. für das Paket muss die SPD konsultiert werden.

Es müssen Regeln für eingehende und ausgehende Pakete erstellt werden. Für jedes Netzwerk-Interface, dass IPSec-Funktionen anbietet, müssen eigene Anforderungen in die SPD eingetragen werden. Die SPD wird über Selektoren indiziert, die jeweils eine unterschiedliche Granularität bieten. Selektoren sind i.A. IP-Adressen, Ports oder Namen. Bei Adressen können Netzwerkmasken, eine Liste spezieller Adressen oder der Typ (Uni-, Broadcast) als Ausprägung der Selektoren genutzt werden. Die SPD bezieht sich somit auf die Kommunikationsendpunkte [15] einer Beziehung. Für den Fall eines SA bundles legt die SPD die Reihenfolge der Anwendung der einzelnen SAs fest.

2.2 Security Associations und Key Management

Das Konzept der Security Association bietet einen sehr flexiblen Mechanismus zur Bereitstellung von Sicherheitsfunktionen in der Netzwerkschicht. Für die Etablierung der notwendigen Parameter jeder SA werden zusätzliche Protokolle bzw. Verfahren benötigt.

Ohne den Einsatz solcher Verfahren ist ein manueller Austausch der Parameter notwendig, beispielsweise durch eine mit PGP verschlüsselte E-Mail oder ein Treffen der Teilnehmer in der realen Welt. Es ist für kleinere Szenarien durchaus anwendbar. Für größere Netzwerkstrukturen wird es jedoch schnell unflexibel und schwer administrierbar.

Für sicheres automatisches Key- und SA-Management über ein unsicheres Netzwerk sind deshalb zwei Ansätze entwickelt worden:

1. Die Schlüsselinformationen werden direkt in das Payload jedes IP-Paketes durch einen zusätzlichen Header eingebettet, das Key Management verbleibt bzgl. des OSI-Modells auf der gleichen Schicht (in-band) [26]

2. Ein zusätzliches, separates Protokoll erbringt die Funktionalität, d.h. bzgl. des OSI-Modells arbeitet es auf einer höheren Ebene als IP (out-of-band)

Nachfolgend werden einige Verfahren vorgestellt, die in jeweils eine der beiden Kategorien eingeordnet werden.

2.3 Simple Key Management for Internet Protocols (SKIP)

SKIP ist ein Key-Management-Protokoll [9], dass komplett *in-band* arbeitet, d.h. in die erste Kategorie einzuordnen ist. Das Schlüsselmaterial wird durch einen zusätzlichen Header zur Verfügung gestellt, der jedem IP-Paket beiliegt.

SKIP basiert wie alle anderen betrachteten Protokolle auf dem Diffie-Hellman-Verfahren (DH). Jeder Prinzipal (hier IP-Knoten) hat einen öffentlichen Schlüssel, d.h. einen DH-Exponenten g^i mod p , der von einer dritten Instanz zertifiziert werden muss und langlebig ist. Wenn der Prinzipal J den geheimen Schlüssel j hat, ergibt sich ein gemeinsames Geheimnis g^{ij} mod p mit dem Prinzipal I. SKIP bezeichnet das Geheimnis als Masterkey K_{ij}. Jedes IP-Paket wird mit einem zufällig erzeugten Paketschlüssel K_p verschlüsselt. K_p wird zusätzlich mit dem Masterkey K_{ij} verschlüsselt und in das IP-Paket eingebaut.

- Vorteile:
  Weil das Verfahren keine Zustandsinformationen benötigt, werden keine lokalen Speicherobjekte gebraucht. Die Gefahr von “Denial of Service” Attacken bzgl. einem erhöhten Verbrauch an Rechenressourcen ist in diesem Zusammenhang als gemindert einzustufen. SKIP ist wesentlich geringer komplex als *out-of-band* Protokolle, da lediglich ein authentisierter öffentlicher DH-Exponent des Partners für die Kommunikation benötigt wird.

- Nachteile:
  Das Protokoll beschreibt keine Vorgehensweise, wie der für die Verschlüsselung jedes IP-Paketes anzuwendende Algorithmus auszuhandeln ist. Der Sender muss wissen, welche Algorithmen der Empfänger implementiert hat und den korrekten auswählen. Weiterhin ist ein Overhead an Daten vorhanden, da bei jedem Paket der SKIP-Header und der Schlüssel Kp verschickt wird. Das bedeutet, das ca. 20 - 30 Byte zusätzliche Daten pro IP-Paket hinzukommen. Ein Problem für die Vertraulichkeit ist der langlebige öffentliche Schlüssel g^i mod p. Es ist deshalb mit einem erheblichen Mehraufwand verbunden, einen neuen öffentlichen Schlüssel zu etablieren. Ein schnelles Aushandeln neuer Parameter ist somit kaum möglich.

Für SKIP gibt es mehrere Implementierungen, eine in der Praxis eingesetzte ist *SunScreen SKIP* [6] und stammt von SUN.

2.4 Photuris

Das Protokoll Photuris [21] wird zur Bereitstellung kurzlebiger symmetrischer Schlüssel benutzt und ist in die zweite Kategorie einzuordnen.

Photuris unterscheidet zwischen einer Host-Authentisierungsphase und dem wirklichen Schlüsselaustausch. Es liegen somit die notwendigen SA-Parameter und deren Gültigkeitsdauer nach Etablierung der Verbindung vor. Das Protokoll kann deshalb sehr gut für IPSec verwendet werden.

Photuris besteht aus mehreren Protokollphasen:

1. Cookie Exchange
   Die erste Phase dient zur Abwehr von DoS-Angriffen. Einem Angreifer soll es nicht gelingen, das System durch Fluten mit Authentisierungsanfragen bzw. durch die Erzwingung eines erhöhten Verbrauches an Rechenzeit lahmzulegen. Dazu werden Cookies als Pre-Authentisierungstoken benutzt. Ein Cookie ist ein Hashwert. Beide Kommunikationspartner erzeugen ihre Cookies aus verschiedenen Parametern, d.h. einem Geheimnis, ihren IP-Adressen und weiteren Daten. Die entstehenden zwei Cookies definieren genau eine Austauschbeziehung. Der Vorteil der Cookies besteht darin, das kaum Rechenzeit verbraucht wird und keine weiteren Statusinformationen gehalten werden müssen.

2. Value Exchange
   Aus den in der ersten Phase angebotenen Schlüsselaustauschverfahren werden nun die für das ausgewählte Verfahren notwendigen Werte ausgetauscht. Im Falle von DH werden somit die öffentlichen DH-Exponenten übertragen.

3. Identification Exchange
   Es werden die Identitäten der Teilnehmer übermittelt, geprüft und die DH-Exponenten authentisiert.

Nach Abschluss des Verfahrens besteht ein sicherer Kanal, über den Schlüssel z.B. für IPSec ausgetauscht werden können.

- Vorteile:
  Photuris ist ein deutlich geringer komplexes Schlüsselaustauschprotokoll im Vergleich zu anderen *out-of-band* Protokollen. Mit Hilfe des Cookie-Konzeptes werden einfache DoS-Angriffe abgewehrt.

- Nachteile:
  Obwohl das Protokoll einige Vorteile gegenüber IKE bietet, gibt es leider kaum Implementierungen. Es gibt lediglich einen Daemon für OpenBSD, die Entwicklung wurde aber eingestellt.

Auf Basis der OpenBSD-Implementierung wurde als Proof-of-Concept eine Portierung für Linux von H. Höxer vorgenommen [17].

2.5 ISAKMP/IKE

2.5.1 ISAKMP

Das “Internet Security Association and Key Management Protocol” [27] ist ein Protokollrahmen, das mehrere Nachrichtentypen definiert, auf denen das eigentliche Key-Management-Protokoll aufgesetzt wird. ISAKMP-Nachrichten werden wie bei Photuris mit dem verbindungslosen UDP versendet. UDP wird deshalb benutzt, um zu haltende Zustandsinformationen soweit wie möglich zu minimieren.

Um ISAKMP als universelles Key-Management-Protokoll zu benutzen, bietet es einen 2-Phasen-Ansatz. In der ersten Phase wird ein sicherer Kanal für das Schlüsselmaterial aufgebaut. In Phase 2 können dann für beliebige Instanzen, z.B. IPSec, über den etablierten Kanal die Schlüssel schnell und sicher ausgehandelt werden.

Phase 1 benutzt kryptographisch starke Verfahren, um Initiator und Responder zu authentisieren. Zum Schutz des ISAKMP-Kanals werden die kryptographischen Verfahren ausgehandelt und die Schlüssel etabliert.

Dieser Ansatz bietet trotz höherem Aufwand folgenden Vorteil:

- Fehler oder Angriffe in Phase 2, die zum Aufsetzen neuer SAs führen, zerstören den sicheren Kanal aus Phase 1 nicht. Er kann deshalb mehrmals für den Schlüsselaustausch benutzt werden.

Die Reihenfolge und den Inhalt der Nachrichten während der Kommunikation zwischen beiden Partnern definiert ISAKMP in fünf Exchange Types, z.B. Identity Protection und Aggressive Exchange. Sie erbringen jeweils unterschiedliche Funktionalität, z.B. nur reine Authentisierung, und benötigen eine definierte Anzahl an Nachrichten zur Etablierung des sicheren Kanals.

2.5.2 IKE

Das Protokoll "Internet Key Exchange" [18] verwendet den Protokollrahmen ISAKMP und bietet ein flexibles Verfahren, um Schlüsselaustausch und damit das Aushandeln von SAs zu ermöglichen und an verschiedene Sicherheitsanforderungen anzupassen. IKE verwendet den 2-Phasen-Ansatz von ISAKMP und nutzt in Phase 1 die ISAKMP Exchange Typen. Sie legen die Reihenfolge der Nachrichten fest.

Weil jede Folgenachricht von der empfangenen Nachricht abhängig ist, sind jeweils drei bzw. sechs Nachrichten zur Etablierung des sicheren Kanals in Phase 1 notwendig. Für IKE ergeben sich daher zwei definierte Modi, der Aggressive Mode und der Main Mode.

- Aggressive Mode:
  Der Modus ist eine Betriebsart des Aggressive Exchange von ISAKMP und benötigt drei Nachrichten. Danach sind Initiator und Responder gegenseitig authentisiert. Der Modus ist eingeschränkt bzgl. der Aushandlung von SA-Parametern, sie müssen teilweise vorher bekannt sein. Da keine Cookies verwendet werden, hat er eine weitaus höhere DoS-Angriffswahrscheinlichkeit. Die angedachte Verwendung des Modus ist die Einbindung eines Roadwarriors in ein Firmennetz. Er kennt die entsprechend verwendeten kryptographischen Algorithmen und muss sie damit nicht erst explizit aushandeln. Deshalb benötigt der Modus weniger Nachrichten zur Etablierung der SAs.

- Main Mode:
  Der Modus ist eine Betriebsart des "Identity Exchange Type" von ISAKMP. Es sind sechs Nachrichten notwendig, um den sicheren Kanal aufzubauen. Die Verwendung eines Cookie-Austausches in den beiden ersten Nachrichten bietet einen wesentlich besseren Schutz gegen DoS Angriffe. Initiator und Responder haben die höchstmögliche Freiheit bei der Aushandlung der SA-Parameter.

Nach dem Aufbau des sicheren Kanals wird dieser in Phase 2 benutzt, um die eigentlichen IKE-SAs auszuhandeln. Dafür wird ein deutlich geringerer Overhead benötigt. Es wird der Quick Mode eingesetzt, ein wesentlich schnelleres Protokoll, das lediglich drei Nachrichten benötigt. Der Quick Mode ist ein von IKE neu eingeführter Exchange Type, der nicht in ISAKMP vorkommt.

Während die IKE-SAs der Phase 2 uni-direktional sind, ist zu beachten, das die ISAKMP-SA der Phase 1 für den sicheren Kanal bi-direktional ist. Für die Authentisierung der beiden Partner und damit der ISAKMP-SA bietet IKE Verfahren, die unterschiedlichen Schutzzielen genügen.

- Pre-Shared-Keys (PSK)
  Initiator und Responder haben vorher außerhalb des Protokolls ein gemeinsames Geheimnis ausgetauscht, auf dessen Basis die Authentisierung stattfindet. Es wird keine Zertifizierungsstruktur benötigt, das Verfahren ist nur in kleineren Netzstrukturen sinnvoll einsetzbar.

- Public-Key-Encryption (PKE)
  Initiator und Responder authentisieren sich, indem sie ihre Identität mit dem öffentlichen des Schlüssel des Partners verschlüsseln. In Phase 1 ist der erste Teil der Kommunikation nicht nachweisbar bzgl. des Senders, da potenziell jede Person die Nachrichten mittels eines öffentlichen Schlüssels verschlüsselt haben kann.

- Signatur (SIG)
  Der Initiator unterschreibt die ausgetauschten Nachrichten mit seinem privaten Schlüssel. Der Responder kann anhand des öffentlichen Schlüssels des Initiators die Signatur überprüfen. Damit wird die Nichtabstreitbarkeit der Kommunikationsbeziehung erreicht. Es wird eine Zertifizierungsstruktur benötigt, damit der Responder dem öffentlichen Schlüssel vertraut und eine Man-in-the-Middle Attacke vermieden wird.

Die in die Generierung eingehenden Werte des IKE-SA Schlüsselmaterials in Phase 2 hängen von dem jeweils gewählten Authentisierungsverfahren ab, d.h. bei Wahl von SIG entstehen andere Schlüssel als bei der Wahl von PSK. Beide Modi, d.h. Aggressive bzw. Main Mode, können beliebig mit SIG, PSK oder PKE kombiniert werden. Daraus ergeben sich sechs verschiedene Möglichkeiten zum Aufbau des sicheren Kanals. Hier ist bereits die große Komplexität von IKE zu erkennen.

Für den Schlüsselaustausch in Phase 1 wird wie bei Photuris und allen anderen Verfahren DH benutzt. Außerdem wird ein Zufallswert (Nonce) N eingebunden. Durch die Zufallsgröße soll die Wahrscheinlichkeit für die Kompromittierung der Schlüssel weiter gesenkt werden.

Um den Phase 1 Exchange zu authentisieren, wird ein Masterkey Mkey erzeugt. Dieser wird je nach gewählten Authentisierungsverfahren unterschiedlich berechnet, da bei den Verfahren Parameter teils verschlüsselt, teils unverschlüsselt übertragen werden bzw. schon vorher bekannt sind. Es wird weitgehend eine strikte Trennung von Authentisierung und Schlüsselaustausch eingehalten, um so wenig wie möglich Informationen über die Schlüssel preiszugeben. Aus dem Masterkey Mkey werden weitere Schlüssel

generiert, die für die Erzeugung von Schlüsseln der Phase 2, zur Authentisierung der Phase 2 Nachrichten und zur Verschlüsselung weiterer IKE-Nachrichten benutzt werden. Sie werden jeweils mit einer Hashfunktion berechnet und sind somit statistisch voneinander unabhängig.

IKE bietet die Option Perfect Forward Secrecy (PFS). PFS bedeutet, dass ein Angreifer bei Kompromittierung eines Schlüssels ausschließlich nur die mit diesem Schlüssel verschlüsselten Daten dekodieren kann. Das wird erreicht, indem der aktuelle Schlüssel nicht in die Berechnung weiterer Schlüssel eingeht und Schlüsselausgangsmaterial nicht erneut zur Generierung neuer Schlüssel benutzt wird. Für PFS in IKE heißt das allerdings, dass ein Masterkey Mkey nur einmal verwendet werden darf, um PFS zu erreichen [26]. Das bedeutet, dass nach jedem Phase 2 Exchange die erste Phase neu aufzusetzen ist, was natürlich einen extrem hohen Overhead erzeugt.

- Vorteile:
  IKE bietet eine große Bandbreite an Möglichkeiten für unterschiedliche Vertrauens- und Authentisierungsszenarien. Der 2-Phasen-Ansatz erzeugt einen sicheren Kanal zur Etablierung von SAs, der auch durch Zerstörung/Fehler in der zweiten Phase nicht reinitiiert werden muss. Dennoch sollte die Phase 1 ISAKMP SA eine bestimmte Lebensdauer haben, damit die Möglichkeit eines Brechens der Schlüssel minimiert wird.

- Nachteile:
  Der große Umfang von IKE ist auch gleichzeitig ein Problem, d.h. IKE ist schwer auf verschiedene Angreifermodelle zu evaluieren. Es ist einfach zu komplex und daher eher fehleranfällig. Versuche, IKE zu vereinfachen, sind IKEv2 [22] und “Just Fast Keying” (JFK) [8], die momentan aber in der Praxis noch wenig Bedeutung haben.

Angriffe auf IKE Der Aggressive Mode von IKE bietet den Vorteil, durch lediglich drei Nachrichten einen sicheren Kanal zu errichten. Da aber kein Cookie-Exchange stattfindet, ist der Modus besonders für DoS-Angriffe anfällig. Außerdem gibt es einen erfolgreichen Angriff auf den Aggressive Mode in Verbindung mit PSK [2].

Ein allgemeines Problem sind DoS-Angriffe, die in Zusammenhang mit der Umsetzung des Cookie-Konzeptes entstanden sind. Ein Beispiel ist die *Cookie Jar* Attacke. Da es keine Ressourcenbeschränkung gibt, d.h. jeder Teilnehmer kann laut Spezifikation beliebig viele Verbindungen zu einem Partner öffnen, kann ein Angreifer eine große Anzahl an IKE Nachrichten erzeugen und die Antworten auf diese in einem *Cookie Jar* sammeln. Das kostet ihn kaum Ressourcen, da die Cookie-Nachrichten sehr klein sind. Danach generiert der Angreifer für die Antworten eine große Menge an Schlüsselaustausch-Nachrichten und verschickt sie gleichzeitig. Der Responder

wird komplett überlastet und in die Knie gezwungen, da die Schlüsseloperationen sehr kostenintensiv sind [30].

2.6 Review IPSec Infrastrukturen

Um eine Evaluierung und Erweiterung der Mikro-SINA IPSec Architektur besser vornehmen zu können, möchte ich im Folgenden einige in der Praxis erfolgreich eingesetzte IPSec Infrastrukturen näher vorstellen.

2.6.1 Windows 2000 und XP

Die Betriebssystemversionen Windows 2000 und XP enthalten eine IPSec Infrastruktur, um IPSec Verbindungen mit Hilfe von ISAKMP/IKE herzustellen. Die Infrastruktur besteht aus drei Hauptkomponenten:

- Policy-Agent
  Er fordert die Policies von den Policy Stores an. Stores können Active Directories, lokale Caches oder die Registry sein. Danach werden die Policies entweder an das IKE Modul zur Aushandlung der SAs bzw. an den IPSec Treiber weitergeleitet. Der aktive Teil des Agenten ist ein Windows Dienst mit dem Namen “IPSec Policy Agent” (2000) bzw. “IP Services” (XP). Er wird beim Systemstart automatisch aktiviert.

- IKE Modul
  Das IKE Modul wird bei Bedarf vom Policy Agent gestartet und handelt die Security Associations für ISAKMP (Phase 1), sowie für IPSec (Phase 2) aus. Die Parameter der SAs werden an den IPSec Treiber übermittelt.

- IPSec Treiber
  Der Treiber filtert die Pakete danach, ob sie verworfen, durchgelassen oder mit IPSec zu schützen sind. Zu schützende Pakete werden mit den notwendigen IPSec Parametern bearbeitet und an den Netzwerktreiber weitergeleitet. Der IPSec Treiber verwaltet alle aktiven SAs in seiner SADB.

Die beiden Windows-Versionen implementieren eine umfangreiche IPSec Suite, um geschützte Verbindungen mittels VPNs aufzubauen.

2.6.2 FreeS/WAN Linux

FreeS/Wan [1] ist eine frei verfügbare Implementierung von IPSec für Linux 2.2 und 2.4, sowie experimentell für 2.6. Sie besteht aus einer Menge von Kernel-Patches und User-Space Applikationen.

Die Infrastruktur von FreeS/Wan besteht aus Kernkomponenten, die sowohl SADB und als auch SPD verwalten. Im Kern wird auf die Routing-Informationen zugegriffen, um mit Hilfe der SPD zu entscheiden, ob Pakete mit IPSec geschützt werden sollen. Eingehende Pakete überprüft FreeS/WAN mit Hilfe der SADB, ob sie gültig sind und wendet danach die entsprechenden Parameter der SA an.

Im User-Space existieren Key Management Dämonen (KMd), standardmäßig ist das eine Instanz des KMd Pluto. KMd sind für die Aushandlung der ISAKMP und IPSec SAs verantwortlich und kommunizieren mit den Kernkomponenten via BSD Socket Interface. Sie

müssen als root-priviligierte Applikationen gestartet werden, ansonsten ist keine Vertraulichkeit gegeben.

FreeS/WAN wurde im März 2004 nach 5 Jahren Entwicklungszeit eingestellt, ein Hauptgrund war die fehlende Akzeptanz der 2.x Version. Aktive Weiterentwicklungen sind die Projekte Openswan [3] und StrongSwan [5]. OpenSwan bietet *Network Address Translation* (NAT), d.h. Funktionalität für Gateways, die beim Weiterleiten von IP-Paketen deren Adressen ändern. Außerdem implementiert Openswan eine größere Anzahl an kryptographischen Algorithmen, z.B. AES. StrongSwan bietet einen ähnlichen Umfang an Funktionalität wie Openswan.

2.6.3 OpenBSD

IPSec wird im Kern von OpenBSD als ein Paar von Sicherheitsprotokollen (AH und ESP) implementiert [16]. Eintreffende IPSec-Pakete werden einfach an das entsprechende Protokoll zu weiteren Verarbeitung übermittelt. Die im Kern befindliche SADB wird zur Ermittlung der zugehörigen SA benutzt, indem Informationen, konkret z.B. die SPI, aus dem Paket verwendet werden. Sobald das IPSec Paket bearbeitet wurde, wird es wieder dem IP Modul zugeführt.

Für ausgehende Pakete, d.h. die über das Routing aus dem vertraulichen Netz oder von höheren Schichten des OSI-Modells eintreffen, wird in der ebenfalls im Kern befindlichen SPD nachgeschaut, ob es mit IPSec zu schützen ist und die eventuell vorhandene SA aus der SADB geholt.

Falls keine SA existiert, wird analog wie bei FreeS/WAN ein KMd im User-Space zur Aushandlung der SA via BSD Socket Interface beauftragt. Der KMd ist *isakmpd*, eine User-Applikation, die den Protokollrahmen ISAKMP und deren momentan einziges Protokoll IKE verarbeiten kann. Damit soll *isakmpd* für weitere Protokolle offen sein, die auf das Framework ISAKMP aufsetzen. Die Ungenauigkeiten in der Definition von ISAKMP hat aber schon viel Kritik hervorgerufen [13], sodass die Wahrscheinlichkeit eines weiteren Protokolls für die aktuelle Version von ISAMKP eher gering ist.

Da es keine standardisierte Form gibt, Policies zu spezifizieren und auszuführen, hat das OpenBSD Team ein Trust Management System eingesetzt. Trust Management Systeme bieten eine abstraktere Schnittstelle zur Spezifizierung von Security Policies und deren Beziehungen zu Entitäten der Systeme [16]. OpenBSD verwendet konkret KeyNote [10] als Instanzierung eines Trust Management Systems.

OpenBSD liefert somit standardmäßig IPSec im Kern. Dadurch ergeben sich klare Designvorteile im Vergleich zu FreeS/WAN, da IPSec komplett integriert ist.

2.6.4 SINA

Die Sichere Inter-Netz Architektur (SINA) ist eine vom BSI und der secunet AG entwickelte Architektur zum Schutz hochwertiger Daten [4]. Sie besteht z.Zt. aus folgenden Komponenten, die für unterschiedliche Anwendungsszenerien eingesetzt werden können:

- SINA-Box
  Sie stellt das auf dem IPSec Standard basierende VPN-Gateway dar und enthält neben mehreren Kryptomodulen das IPSec/IKE-Modul sowie Management- und Intrusion-Detection-Systeme. Sie verbindet das vertrauliche Netzwerk mit dem nicht vertrauenswürdigen Inter-Netzwerk. Dafür hat die Box zwei Netzwerkkarten, eine für klare Kommunikation innerhalb des gesicherten Bereichs und eine für kryptierte Kommunikation nach außen.

- SINA-Client
  Der Client ist ein Thin-Client, der mit der SINA-BOX via IPSec-Tunnel kommuniziert. Er besitzt keinen persistenten Speicher, die sicherheitsrelevanten Daten verbleiben im geschützten Netzwerk. Der SINA-Client ermöglicht so hoch sicherheitsintensiven Applikationen Zugang zum vertraulichen Netzwerk.

- SINA-Management
  Zur Konfiguration der einzelnen Komponenten, wie z.B. der SINA-Box, wird das skalierbare SINA-Management verwendet. Im Gegensatz zu den beiden vorigen Teilen besitzt es einen persistenten Speicher, indem u.a. die Konfigurationsdateien gehalten werden.

Allen Komponenten dient als Basis ein stark minimalisiertes und sicherheitstechnisch gehärtetes Linux, das SINA-Linux.

2.6.5 Mikro-SINA

Während alle vorigen IPSec-Infrastrukturen auf einem monolithischen Betriebssystem beruhen und nur einige Komponenten, wie z.B. das Key Management, in root-priviligierte Nutzer-Anwendungen auslagern, geht Mikro-SINA einen komplett neuen

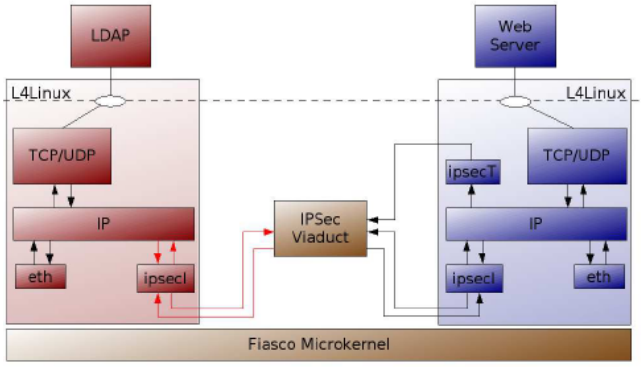

Abbildung 2.1: Mikro-SINA Architektur. Die sicherheitskritische Komponente, d.h. das Viaduct, ist braun dargestellt. Die rote L^{4}Linux Instanz hält die Verbindung zum sicheren, die blaue L^{4}Linux Instanz zum äußeren Netz. Entnommen aus [19].

Weg in Bezug auf IPSec. Es hat sich zum Ziel gesetzt, die SINA-Box prototypisch als VPN-Gateway in einer Mikrokern-Umgebung umzusetzen.

Durch die Verwendung eines Mikrokerns, konkret L4/Fiasco, wird eine Kapselung der einzelnen Komponenten der IPSec Architektur ermöglicht. Somit lassen sie sich bzgl. Datensicherheit deutlich besser evaluieren und es wird eine klare Trennung zwischen nicht vertrauenswürdigen und vertrauenswürdigen Teilen erreicht.

Abbildung 2.1 zeigt den aktuellen Stand der Mikro-SINA Architektur zu Beginn dieser Arbeit. Sie besteht aus zwei L^{4}Linux Instanzen, wobei eine die Kommunikation in den sicheren Teil des Netzwerkes ermöglicht und die andere zur Kommunikation in das nicht vertrauenswürdige Inter-Netzwerk dient.

Beide Instanzen sind über eine Brückenkomponente, dem Viaduct, miteinander verbunden. Das Viaduct übernimmt damit alle sicherheitskritischen Aufgaben, d.h. es ermöglicht den Datenfluss zwischen vertrauenswürdigem und nicht vertrauenswürdigem Netz. Es implementiert IPSec, die Sicherheitsprotokolle AH und ESP, basiert auf IPv4 und hat exklusiven Zugang zur SADB und SPD. Damit bestimmt es, wie und welche IP-Pakete zu sichern sind.

Der Nutzer muss somit sowohl dem Mikrokern als Basis als auch dem Viaduct vertrauen. Die äußere L^{4}Linux Instanz, im Weiteren *Internet L^{4}Linux*, muss nicht auf Vertraulichkeit untersucht werden, da sie nur geschützte Daten sieht. Für die innere L^{4}Linux Instanz, nachfolgend als *privates L^{4}Linux* bezeichnet, ist die Vertraulichkeit ebenfalls nicht nachzuweisen, da sie gekapselt in einer Sandbox läuft, d.h. keine Möglichkeit hat, vertrauliche Daten an nicht autorisierte Komponenten weiterzugeben. Sie kann die Daten nur in das vertrauliche Netz und an das Viaduct weiterleiten.

Kapitel 3

Entwurf

Ausgehend von der Einführung beider Grundkonzepte für SA- und Key-Management (*in-band* bzw. *out-of-band*) im letzten Kapitel möchte ich im ersten Teil meine Auswahl des Ansatzes und jeweiligen Protokolls erläutern. Aus der Entscheidung resultierend werde ich mein Modell der IPSec Infrastruktur für Mikro-SINA entwickeln. Es werden die einzelnen Komponenten näher beleuchtet und die Interaktionen zwischen ihnen analysiert bzgl. der Schutzziele Integrität, Vertraulichkeit und Verfügbarkeit. Für die Kommunikation der Komponenten werden die entsprechenden Schnittstellen vorgestellt und evaluiert. Im Anschluss wird auf das Thema IPv6 in Verbindung mit Mikro-SINA eingegangen. Dabei werden die Neuerungen von IPv6 erläutert und danach auf die IPSec Infrastruktur bezogen.

Für ein Verständnis der Schutzziele im Rahmen der IPSec Infrastruktur soll zuerst eine allgemeine Definition der drei Ziele nach [29] vorgenommen werden. Unter Vertraulichkeit ist zu verstehen, dass Informationen nur Berechtigten bekannt werden. Integrität bedeutet, dass Informationen richtig und vollständig sind, oder aber eine Veränderung erkennbar ist. Verfügbarkeit heißt, Informationen sind zugänglich zum Zeitpunkt, an dem sie benötigt werden.

3.1 Entwurfskriterien

Für die Entwicklung der IPSec Infrastruktur sollen folgende Designkriterien gelten:

1. hohe Sicherheitsanforderungen, d.h. nach [29] die Schutzziele
    - Vertraulichkeit. Durch die Erweiterung der Architektur soll der vorhandene Grad an Vertraulichkeit mindestens gleich bleiben, im optimalen Fall sich stark erhöhen.
    - Integrität. Die vertraulichen Daten dürfen beim Einsatz der KMds und anderer IPSec Komponenten nicht verfälscht werden.

- Verfügbarkeit. Bei der Nutzung der zusätzlichen Key-Management Komponenten darf die gesamte Architektur nicht in ihrer Zugänglichkeit beeinträchtigt werden. Die Aushandlung der SAs darf die Mikro-SINA Architektur nicht blockieren.

Für die Einhaltung dieser Anforderungen sollen die Komponenten eine möglichst gute Evaluierbarkeit aufweisen.

2. hohe Interoperabilität
   Mikro-SINA soll mit anderen IPSec Implementierungen (z.B. FreeS/WAN) SAs aushandeln können, dadurch wird ein hoher Migrationsgrad für Mikro-SINA in bestehenden VPNs erreicht.

3. hohe Wiederverwendbarkeit
   Wenn möglich, sollen vorhandene, bewährte Lösungen auf L4 portiert werden, da das IPSec Management sehr komplex ist. Wiederverwendung bezieht sich dabei nicht nur auf Software, sondern auch auf Schnittstellen, die auf L4 abgebildet werden.

3.2 Wahl des Grundkonzeptes und Protokolls

Mit Hilfe der Entwurfskriterien ist intuitiv klar, dass nur *out-of-band* Protokolle in Frage kommen. Sie bieten gegenüber *in-band* Ansätzen eine viel höhere Flexibilität trotz erhöhten Aufwandes. Um nur ein Beispiel zu nennen: erhöhte Sicherheitsanforderungen implizieren den Einsatz möglichst kurzlebiger Schlüssel. Dieses Kriterium ist nur mit *out-of-band* Konzepten zu erfüllen.

Im nächsten Schritt habe ich mich für das *out-of-band* Protokoll ISAKMP/IKE entschieden. Wie bereits angedeutet, gibt es viele Kritikpunkte am IPSec Framework, speziell aber auch an IKE. Sie beruhen besonders auf der hohen Komplexität der Protokolle, so schreiben B. Schneier und N. Ferguson in Bezug auf IPSec: "*Security's worst enemy is complexity*" [13]. Auch die RFCs, wie ISAKMP, sind oft nicht sehr eindeutig in ihren Formulierungen. Beide Autoren schlagen deshalb einige signifikante Vereinfachungen vor, z.B. Abschaffung des Transport Modus und bi-direktionale anstelle von uni-direktionalen IPSec SAs. Dennoch, "*IPSec is the current best practice*" [13], es gibt momentan keinen anderen Ansatz, der eine so hohe Flexibilität und Sicherheit bietet. Ein weiterer Aspekt, sich für IKE zu entscheiden, ist die Interoperabilität. Alle vorgestellen Infrastrukturen für IPSec arbeiten mit IKE. Wenn Mikro-SINA also keine Insellösung darstellen soll, ist unbedingt ISAKMP/IKE als Protokoll zu favorisieren.

3.3 Mikro-SINA IPSec Infrastruktur

Die vorhandene Mikro-SINA Architektur ermöglicht den Aufbau von VPNs ausschließlich mit manuellem Schlüsselaustausch. Die Entscheidung IKE als SA- und Key-Management Protokoll einzusetzen bedingt, dass die Funktionalität der Brückenkomponente, im Weiteren Viaduct genannt, untersucht werden muss.

Das Viaduct bietet folgende Dienste:

- Verwaltung der SADB und SPD, Persistenz der notwendigen Parameter von SAs und Policies
- Sicherstellung der Kommunikation zwischen dem vertrauenswürdigen und nicht vertrauenswürdigen Netz
- Entscheidung, welche IP-Pakete mit IPSec behandelt werden sollen auf Basis der SPD bzw. SADB
- Verschlüsselung der IP-Pakete, Entschlüsselung der IPSec Pakete

Das Viaduct implementiert damit die absolut notwendigen Kernfunktionalitäten zum Aufbau von IPSec Verbindungen. Die gestellten Designziele erfordern deshalb, das für die automatische Aushandlung der SAs eine neue Komponente, der Key Management Dämon (KMd), eingeführt werden muss. Die Beibehaltung des hohen Sicherheitsniveaus bei Hinzufügung des KMd wird durch die klare Aufgabentrennung zwischen Viaduct und KMd erreicht.

Der KMd ist verantwortlich für:

- Aufbau der ISAKMP SA in Phase 1,
- Aufbau der IPSec SAs in Phase 2,
- damit Aushandlung der notwendigen SA-Parameter,
- Übergabe der SAs an das Viaduct, das die IPSec Verbindungen etabliert,
- Übergabe von SPD Einträgen, um neue IPSec Verbindungen zu ermöglichen.

KMd und Viaduct arbeiten somit zusammen, wobei der KMd zusätzliche Funktionalitäten erbringt. Bei deaktivierten KMd ist das IPSec VPN Gateway weiterhin funktionsfähig, aber nur mit manuellem Keying. Abbildung 3.1 zeigt die entstandene Architektur.

Für die Untersuchung der Interaktionen der IPSec Infrastruktur bzgl. Vertraulichkeit, Integrität und Verfügbarkeit sollen zwei mögliche Szenarien näher analysiert werden:

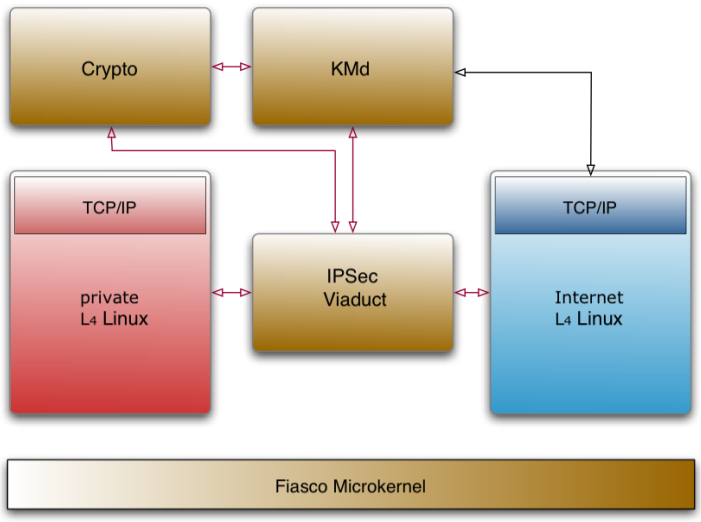

Abbildung 3.1: Mikro-SINA IPSec Infrastruktur. Zu sehen sind die private L^{4}Linux Instanz, die zum sicheren, vertraulichen Bereich gehört, das Internet L^{4}Linux, es stellt die Verbindung zum nicht vertrauenswürdigen Inter-Netzwerk her, und die sicherheitskritischen Komponenten, die braun gefärbt sind.

- für ein aus dem sicheren Netz abgehendes IP-Paket soll automatisch eine IPSec Verbindung aufgebaut werden
- der KMd soll initial eine beliebige Menge an SAs aushandeln und die entsprechenden Daten an das Viaduct weiterreichen

3.3.1 Analyse der SA Etablierung nach Acquire-Nachricht

Im ersten Szenario finden folgende Interaktionen zwischen den Komponenten der IPSec Infrastruktur statt, die mit Abbildung 3.2 illustriert werden, wobei die Reihenfolge den Nummern in der Abbildung entspricht:

1. Nach dem Eintreffen eines IP-Paketes aus dem vertraulichen Netz über das private L^{4}Linux beim Viaduct stellt es fest, dass in der SPD ein Selektor existiert, der die Etablierung einer IPSec Verbindung fordert (apply Regel).
2. Im Falle, dass der Verweis des SPD Eintrages in die SADB leer ist, d.h. es existiert noch keine entsprechende SA, hat das Viaduct eine Anfrage (acquire) an den KMd zu schicken.
3. Der KMd als Initiator baut aufgrund der Acquire-Nachricht via IPC eine Verbindung mit dem Transport/Netzwerk Stack in der Internet L^{4}Linux Instanz auf, um

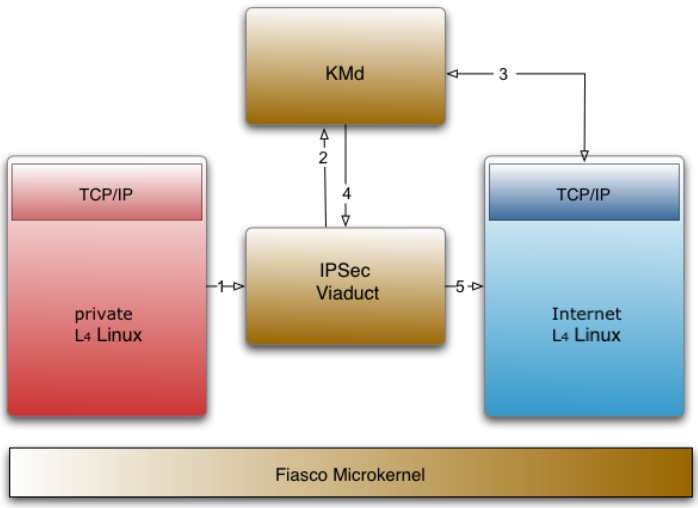

Abbildung 3.2: Mikro-SINA IPSec Infrastruktur Szenario. Automatische Etablierung der SA nach Acquire-Nachricht des Viaducts.

mit den Partnern über das nicht vertrauenswürdige Inter-Netzwerk kommunizieren zu können. ISAKMP/IKE verwendet zwecks Reduzierung des Overheads, d.h. Vermeidung der Persistenzhaltung von Zustandsinformationen, als Transportprotokoll UDP. Als Stack kann sowohl der *FLexible IP-Stack* (FLIPS) als auch ein Wrapper eingesetzt werden, der die Daten direkt in das Internet L^4Linux weiterreicht. Ihm muss nicht vertraut werden, da der KMd nur gesicherte Daten an ihn weitergibt. Weiterhin werden keine Anforderungen bzgl. der Integrität an ihn gestellt, falls der Stack sie verletzen sollte, kann es der KMd mit Hilfe der in IKE referenzierten Verfahren feststellen.

4. Bei erfolgreicher Etablierung der SA wird der KMd die entsprechenden SA-Parameter an das Viaduct weiterleiten.

5. Danach kann das Viaduct die Regeln der entstandenen SA auf das IP-Paket anwenden, das mit IPSec geschützte Paket über die Internet L^4Linux Instanz verschicken und die SA für die zukünftige Verarbeitung weiterer IP-Pakete in seiner SADB ablegen.

Für die Ver- und Entschlüsselung bei der Etablierung der SA bietet sich für den KMd der Zugriff auf einen externen Crypto-Server an, der ihm die notwendige Arbeit abnimmt. Es ist offensichtlich, dass dieser Server vertrauenswürdig und integer sein muss.

Dem KMd selbst muss bzgl. der Aushandlung der SAs und ihrer Übergabe an das Viaduct vertraut werden, da er die notwendigen SA-Parameter mit dem anderen VPN

Knoten austauscht und im Klartext sehen kann. Ansonsten könnte der KMd bei einer SA und Zusammenarbeit mit dem anderen VPN Knoten ein sehr schwaches Verfahren oder NULL Encryption aushandeln.

Um die Verfügbarkeit des KMd zu gewährleisten, ist ein Zeitlimit bei der Etablierung der SAs zu setzen. Wie in 2.5.2 diskutiert, sind durch die in [18] nicht vorgeschriebene Ressourcenlimitierung DoS-Angriffe auf den KMd und damit die gesamte IPSec Infrastruktur durchführbar. Deshalb sollte der KMd zum Schutz der Verfügbarkeit ein Verbindungslimit pro Kommunikationspartner setzen.

3.3.2 Analyse der initialen SA Etablierung mittels KMd

Im zweiten Szenario wird der KMd beim initialen Start eine Verbindung zum TCP/IP Stack aufbauen, um mit der Außenwelt zu kommunizieren. Danach werden eine Menge von SAs zwischen dem KMd und VPN Knoten ausgehandelt. Bevor die erfolgreich ausgehandelten SAs an das Viaduct übergeben werden, hat der KMd entsprechende Policies in der SPD des Viaducts zu erzeugen. Das ist notwendig, weil die IP-Adressen der Policies die Kommunikations-Endpunkte der zu sichernden Verbindung darstellen im Gegensatz zu den kryptographischen Endpunkten, die in der SA stehen. Nur im Transport-Modus fallen kryptographische und Kommunikations-Endpunkte zusammen.

Nach erfolgreicher Einrichtung der SPD Einträge werden die ausgehandelten SAs an das Viaduct übergeben. Die Brückenkomponente trägt sie in seine SADB ein. Sobald nun ein zu schützendes IP-Paket das vertrauenswürdige Netz verlassen will, schaut das Viaduct in seine SPD und wird den neuen Eintrag finden, der auf die vorher ausgehandelte SA in der SADB verweist.

Während im ersten Szenario dem KMd bezüglich der Aushandlung der SAs vertraut werden muss, kommt nun ein weiterer Vertrauensaspekt hinzu, die Manipulation der SPD. Wenn der KMd das Recht hat, Policies zu verändern, kann er nun sogar selbst bestimmen, welche zu schützenden IP-Pakete ungeschützt das Netz verlassen.

Beide Szenarien zeigen also, dass die Software des KMd bzgl. aller drei Schutzziele gründlich zu evaluieren ist, ansonsten kann die gesamte IPSec Infrastruktur kompromittiert werden. Wir müssen also voraussetzen können, dass der KMd eine sicherheitskritische Komponente ist, der man vertrauen kann.

3.4 Kommunikation des KMd

Nachdem der Daten- und Informationsfluss auf einer abstrakten Ebene untersucht wurde, soll im Folgenden eine genauere Betrachtung der Kommunikation zwischen KMd, Netzwerk und dem Viaduct durchgeführt werden. Im Hinblick auf eine mögliche Por-

tierung nach L4 habe ich deshalb vorhandene Schnittstellen evaluiert, die zwischen KMd und Kernkomponenten zur Verwaltung der SAs und der SPD existieren.

Aus den analysierten Szenarien ergeben sich drei Kommunikationswege:

1. Kommunikation mit dem Netz der Außenwelt, d.h. wie ist der KMd über das Netzwerk ansprechbar und wie kann er sich selbst an andere VPN Knoten wenden?
2. Kommunikation mit den Kernkomponenten (Key Engine) für das Key Management (SADB)
3. Kommunikation mit der Key Engine zur Verwaltung der SPD

Für die anschließenden Untersuchungen sind folgende KMd recherchiert worden:

- **Pluto.** aus dem FreeS/WAN Projekt [1]
- **Racoon.** KMd für Darwin, Mac OS X, Linux 2.6, FreeBSD und NetBSD
- **isakmpd.** KMd für OpenBSD und Linux 2.6

Die Analyse der gewählten KMds ergab, dass alle via dem BSD Socket API mit dem Netzwerk und auch mit den Kernkomponenten zum automatischen Key Management kommunizieren. Für die Manipulation der SPD gibt es keine einheitlichen Lösungen, dafür musste ein spezieller Ansatz entwickelt werden.

3.4.1 KMd und nicht vertrauenswürdiges Netz

BSD Socket Interface In der Unix-Welt, aber auch in anderen Betriebssystemen, wird zur Kommunikation zwischen Prozessen die BSD Socket Schnittstelle verwendet. Sie ist eine Standard API und ermöglicht neben verbindungsorientierter (TCP) und verbindungsloser (UDP) auch Kommunikation zwischen Prozessen im selben System. Ihr Entwurf basiert auf einer möglichst einfachen Integration in die Unix Systeme. Bis auf einige spezifische Funktionen zum Auf- und Abbau der Verbindungen (Sockets) können deshalb Funktionen wie read und write transparent verwendet werden.

Weil alle evaluierten KMd grundsätzlich das BSD Socket Interface benutzen, habe ich mich für die Abbildung der Socket-Schnittstelle als Kommunikationsabstraktion für L4 entschieden. Für die Integration der Socket API wurde das L4 Virtual File System (L4VFS) gewählt, da es grundlegende Funktionen wie read, write und close implementiert, die transparent für Sockets benutzt werden können. Innerhalb des L4VFS gab es bereits eine erste Version für Netzwerk-Kommunikation, die auf BSD Sockets ausgelegt war. Der TCP/IP Stack FLIPS hatte ebenfalls auf Sockets basierte Funktionsaufrufe. Mit der Verwendung und Erweiterung der L4VFS net_io Funktionen wurden die IDL-Interfaces von FLIPS und L4VFS verschmolzen. Damit können für BSD Socket

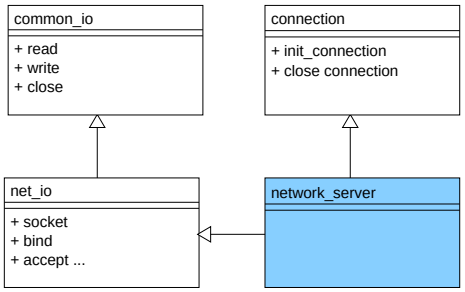

Abbildung 3.3: L4VFS Network Server. Die Vererbungshierarchie der IDL-Interfaces innerhalb von L4VFS. Der Network Server bietet die Basis der des BSD Socket APIs auf Seiten der Server. Sie implementieren *common io*, *net io* und *connection*.

Kommunikation die Interfaces aus L4VFS transparent verwendet werden. Abbildung 3.3 zeigt die entstandenen IDL Schnittstellen.

Die Socket Aufrufe der Clients, in meinem Fall der KMd, wurden auf IPC-Calls an Socket Server abgebildet, die jeweils das Interface *network server* implementieren und somit beliebig ausgetauscht werden können. Als Network Server können z.B. FLIPS oder auch der von mir implementierte Wrapper L4LX_FLIPS benutzt werden. L4LX_FLIPS transferiert die Socket Aufrufe in eine beliebige L^{4}Linux Instanz, siehe 4.5.1. In der Mikro-SINA IPSec Infrastruktur übergibt der KMd so die Daten an das Internet L^{4}Linux.

3.4.2 KMd und Viaduct

SA und Key-Management Wie bereits mehrfach deutlich wurde, ist das Management der SAs und deren Etablierung eine sehr komplexe Aufgabe. Da es mehrere Protokolle dafür gibt und man sie u.U. dynamisch austauschbar benutzen will, wurde eine Schnittstelle gefordert, mit Hilfe derer diese Aufgabe aus den Kernkomponenten herausgelöst werden kann. Somit wird der Kern auch nicht unnötig belastet und schlanker gehalten. Ein solches Interface zur Key Engine im Kern ermöglicht dadurch konsequenterweise den Einsatz von root-priviligierten User-Space KMd, die die Etablierung der SAs übernehmen. Der Kern ist somit nur noch für die Verwaltung, Speicherung und Anwendung der SAs verantwortlich und arbeitet protokollunabhängig.

Eine solche generische Schnittstelle für IPSec ist PF_KEY V2 [28]. Bei allen evaluierten KMd wird sie eingesetzt. Als Schnittstelle zur Key Engine im Kern benutzt sie das BSD Socket API. PF_KEY V2 wird demzufolge im Kern als neue Protokollfamilie (PF_KEY) angemeldet. Im Kern werden die Daten der User-Space KMd danach nicht über das Netz verschickt, sondern lokal verarbeitet.

PF_KEY bietet Nachrichten zum Anlegen, Abfragen, Verändern und Löschen von SAs. Außerdem gibt es Botschaften, die von der Key Engine im Kern zur Benachrich-

tigung der KMd dienen. Das tritt u.a. genau dann ein, wenn eine neue SA benötigt wird und entspricht dem in 3.3.1 besprochenen Szenarios.

Grundsätzlich erfolgt der Aufbau eines Sockets für PF_KEY folgendermaßen:

```
s = socket(PF_KEY, SOCK_RAW, PF_KEY_V2);
```

Danach kann der KMd mit Hilfe des zurückgegebenen File-Handles s Nachrichten, sogenannte Datenpakete, mit der Key Engine im Kern austauschen, z.B. mit den Funktionen read oder write.

Die Datenpakete werden vom Kern syntaktisch überprüft und semantisch interpretiert. Das Ergebnis wird an alle User-Prozesse als neue PF_KEY V2 Nachricht verschickt, die einen Socket der PF_KEY Familie geöffnet haben. Dadurch ist es möglich, eine beliebige Menge von KMd und anderen Programmen, z.B. Administrationstools, gleichzeitig für das SA Management zu benutzen.

Die Nachrichten bestehen aus einem geforderten Header, der wie bei IPv6 auf eine 64-Bit-Grenze ausgerichtet ist. Im Header wird mit *sadb msg type* der Typ der Nachricht festgelegt, z.B. SADB_ADD zum Anlegen einer neuen SA im Kern. Die notwendigen Daten jedes Nachrichtentyps werden als Erweiterungsheader an den PF_KEY V2 Header gehangen. Sie haben jeweils eine variable Länge.

Der Vorteil von PF_KEY V2 liegt in der beliebigen Erweiterbarkeit der Schnittstelle. Für zukünftige Protokolle oder neue Nachrichtentypen müssen einfach neue Erweiterungsheader eingeführt werden, die der Kern transparent verarbeiten kann.

Für den Einsatz von PF_KEY V2 für Mikro-SINA sprechen deshalb mehrere Gründe:

1. die Entscheidung, das BSD Socket API auf L4 abzubilden, bietet die Grundlage, PF_KEY V2 einzusetzen
2. beliebige Austauschbarkeit der KMd, Mikro-SINA ist damit nicht auf einen bestimmten Dämonen festgelegt, da alle evaluierten KMd PF_KEY V2 als Schnittstelle nutzen
3. stark generischer Ansatz, neue bzw. Erweiterung vorhandener Protokolle für das SA Management bedingen nur eine Erweiterung oder Veränderung der PF_KEY Nachrichtentypen, die Schnittstelle bleibt aber die gleiche und muss nicht verändert werden

Die Kommunikation zwischen KMd und Viaduct wird somit über BSD Sockets, speziell über PF_KEY V2, realisiert. Da nur das Viaduct die Key Engine verwaltet und exklusiven Zugriff hat, stellt sich die Frage, ob die vorhandene Schnittstelle des Viaducts auf PF_KEY V2 angepasst oder ein Adapter zwischen KMd und Viaduct entworfen werden soll, der PF_KEY V2 auf das vorhandene Interface des Viaducts abbildet.

Der Adapter bietet eindeutige Vorteile:

1. Die vorhandene IPSec Infrastruktur kann ohne die zusätzlichen Komponenten für automatische SA Etablierung komplett weiterverwendet werden, d.h. es besteht weiterhin die Möglichkeit eines minimalen Vertrauensszenarios.

2. Das stark komplexe Parsing der PF_KEY V2 Nachrichten wird nicht in das Viaduct überführt. Es bleibt weiterhin sehr gut evaluierbar und als sicherheitskritische Basiskomponente für Mikro-SINA sehr klein.

3. Das Viaduct kann von Problemen entlastet werden, die durch den Zugriff mehrerer Clients entstehen. Für das Viaduct existiert nur ein SA Management Client, der PF_KEY V2 Adapter. Er übernimmt die Aufgabe der Verwaltung aller Clients, d.h. der KMd.

4. Die mögliche Nutzung durch unterschiedliche IKE Implementierungen wird unterstützt, z.B. im Hinblick auf den in SINA stark modifizierten KMd Pluto. Somit bietet der Adapter die vereinfachte Migration anderer KMd.

Für den Adapter besteht die Wahl, ob er in einem eigenen Adressraum laufen soll bzw. als Thread in der Task des KMd. Die Platzierung in einem eigenen Adressraum ist vorzuziehen, insbesondere im Hinblick darauf, wenn mehrere KMd gleichzeitig eingesetzt werden sollen. So ist es z.B. möglich, dass ein KMd nur ESP bearbeitet, ein zweiter für AH verantwortlich ist und beide sicherheitskritisch voneinander getrennt arbeiten. Der Adapter in seiner eigenen Task gewährleistet, dass vertrauliche Daten nur dem jeweils zugehörigen KMd zugestellt werden.

Die Abbildung 3.4 stellt den Entwurf mit mehreren PF_KEY Clients und dem PF_KEY V2 Adapter dar, der ein selbständiger L4 Server ist. Da der Adapter das BSD Socket Interface implementiert, ist er wie FLIPS ein weiterer L4VFS Network Server.

Hinsichtlich der gestellten Entwurfskriterien, insbesondere der Sicherheitsanforderungen, ist klar, dass der Adapter eine weitere sicherheitskritische Komponente darstellt. Sie muss vertrauenswürdig und integer sein. Es ergibt sich eine erweiterte Vertrauenskette: Das Viaduct vertraut dem Adapter und dieser wiederum dem KMd bzw. der Menge der KMd.

Security Policy Management Für das Management der Policies in der SPD gibt es keinen einheitlichen Ansatz, d.h. keine generalisierte Lösung wie PF_KEY V2 für das SA Management. Die Analyse der Szenarien in 3.3.1 und 3.3.2, sowie die Evaluation der ausgewählten KMd zeigten die Notwendigkeit, dass die KMd Zugriff auf die SPD benötigen. Für ein ausgehendes IP-Paket wird z.B. ein Eintrag in der SPD benötigt, der u.U. erst nach der Aushandlung einer SA durch den KMd entstanden ist. Das bedeutet, der KMd muss den Eintrag erzeugen können.

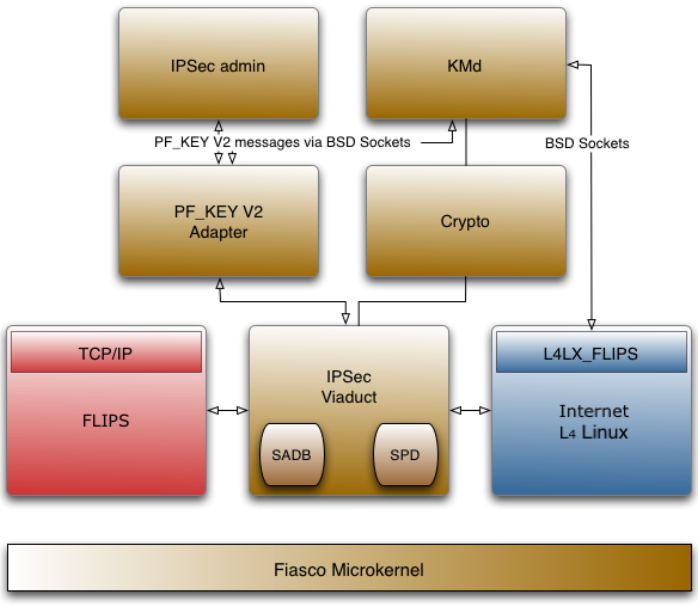

Abbildung 3.4: Erweiterter Entwurf der Mikro-SINA IPSec Infrastruktur. Der KMd und ein administrativer PF_KEY Client, der *IPSec admin*, kommunizieren mit dem Viaduct über den PF_KEY V2 Adapter.

Das Viaduct bietet in seinem Interface die Möglichkeit der Erstellung, Veränderung bzw. Löschung von Einträgen in der SPD. Um einen einheitlichen Ansatz für SA und SPD anzubieten, ist es von Vorteil, wenn die vorhandenen Schnittstellen des Viaducts übernommen und nicht auf PF_KEY V2 umgestellt werden. Somit kann das im vorigen Abschnitt dargestellte minimale Vertrauensszenario erhalten bleiben. Deshalb kommt eine Erweiterung von PF_KEY V2 für das Management der SPD in Frage. Alle evaluierten KMd gehen diesen Weg, um eine gemeinsame Schnittstelle zum Kern zu benutzen. Das erhöht auch die Sicherheit durch Minimierung der Komplexität.

Das Problem liegt im Detail, da die KMd keine komplett einheitlichen neuen PF_KEY V2 Nachrichtentypen für die SPD benutzen. Eine genauere Untersuchung zeigte aber, dass es eine minimale Schnittmenge gibt, die alle KMd unterstützen. Diese Menge an Nachrichtentypen lässt sich auch einfach auf das Interface des Viaducts abbilden.

Die folgenden zwei Nachrichtentypen sind deshalb als neue PF_KEY V2 Nachrichtentypen in den PF_KEY V2 Adapter integriert:

- **ADD_FLOW.** Hinzufügung eines SPD Eintrages
- **DEL_FLOW.** Löschung eines SPD Eintrages

Die FLOW Typen sind eine Entwicklung von OpenBSD und finden u.a. auch in FreeS/WAN Verwendung. Die Funktionen benötigen minimal die IP-Adressen und die SA-Parameter für die Bearbeitung der SPD.

3.5 Mikro-SINA IPv6

Die der Version 4 von IP zu Grunde liegenden Standards reichen in die erste Hälfte der 70er Jahre zurück. Die Größe einer Adresse mit 32bit entsprach dabei den Vorstellungen des Jahres 1975 [12]. Der Siegeszug von IPv4 begann Anfang der 80er Jahre, nachdem das US-Verteidigungsministerium TCP/IP als Standard festgelegt hatte. Besonders nach Einführung des WWWs gab es einen riesigen Schub für das Internet, sodass es bereits um 1990 Befürchtungen gab, die Adressmenge würde knapp werden. Deshalb wurden erste Ideen entwickelt, IPv4 durch eine neue Protokollversion abzulösen. IPv6 wurde ab 1993 von der IETF entwickelt und ist seit 1997 “Draft-Standard”.

Im Folgenden sollen knapp die wichtigsten Änderungen gegenüber IPv4 vorgestellt werden. Weiterhin soll untersucht werden, ob Protokolle oberhalb von IP anzupassen sind, das betrifft insbesondere die in der Mikro-SINA IPSec Infrastruktur benötigten TCP, UDP und IKE. Kurz wird auf die Neuerungen in der BSD Socket Schnittstelle eingegangen.

Danach sollen die Komponenten von Mikro-SINA bzgl. IPv6 analysiert werden. Es stehen folgende Fragen im Mittelpunkt:

1. Können die Komponenten mit Änderungen übernommen werden? Wenn ja, welche Teile müssen überarbeitet werden?

2. Sind Komponenten komplett auszutauschen? Wenn ja, dann wird ein Entwurf der neuen Komponente vorgestellt.

3.5.1 Neuerungen von IPv6

Adressen IPv6 bietet 128bit lange Host-Adressen, wobei drei Arten unterschieden werden:

- **Unicast.** Adressierung einzelner Interfaces eines Rechners.

- **Anycast.** Adressierung einer Gruppe von Interfaces, das IP-Paket wird aber nur an eine Schnittstelle der Gruppe ausgeliefert, z.B. an die nächste, erreichbare Schnittstelle.

- **Multicast.** Adressierung einer Gruppe von Interfaces, die sich i.A. an verschiedenen Rechnern befinden. Das Paket wird an alle Schnittstellen der Gruppe ausgeliefert.

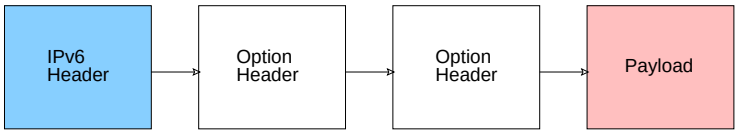

Abbildung 3.5: IPv6 Header mit verketteten Erweiterungs-Headern (Optionen) und dem Payload.

Während bei IPv4 nur eine sehr grobe Einteilung der Adressen in Klassen vorgenommen wurde, ermöglicht IPv6 ganz neue Prinzipien zur Adressaufteilung [12]. Außerdem ist es möglich, IPv4 Netze in IPv6 zu tunneln bzw. umgekehrt, diese Migration fördert eine schrittweise Umstellung auf das neue Protokoll.

Header Für einen beschleunigten Transport der Pakete wurde der Header von IPv6 radikal verkleinert auf ein absolut notwendiges Minimum, denn Router brauchen nur einen kleinen Informationsanteil. Weitere Funktionen und Erweiterungen werden in zusätzliche, optionale Header ausgelagert, die vom Basisheader aus verkettet angehangen werden. Jeder Erweiterungs-Header hat nur eine bestimmte Funktion und muss lediglich dann interpretiert werden, wenn der Inhalt für das jeweilige Protokoll von Bedeutung ist. Abbildung 3.5 verdeutlicht das neue Header-Konzept. Ein minimales IPv6 Paket besteht damit aus dem IPv6-Header und dem Payload. Mit dem Header-Konzept kann IPv6 leicht erweitert werden, dafür sind lediglich neue Erweiterungs-Header zu definieren.

IPSec Wie in 2.1 dargelegt, bietet IPv4 weder Vertraulichkeit noch Integrität der IP-Pakete. Dafür dient die Erweiterung IPSec. Sie entstand zuerst im Rahmen der Entwicklung von IPv6, wurde aber als Ableger schnell unabhängig von der IP Version weiter konstruiert. Für IPv6 war IPSec deshalb von Beginn an integraler Bestandteil. Das neue Header-Konzept von IPv6 bedingt für die Benutzung von IPSec die Einhängung von Erweiterungs-Headern für ESP bzw. AH. Der jeweilige interne Headeraufbau der Sicherheitsprotokolle ist für IPSec und IPv6 gleich.

Weitere Neuerungen Der neue IP-Header bietet keine explizite Checksumme mehr, die Größe des Paketes wird als ausreichende Prüfsumme betrachtet, weiterhin wird der Test übergeordneten Protokollen wie TCP oder SCTP überlassen. IPv6-Routern ist die Fragmentierung der Pakete entzogen wurden, nur noch der Absender darf fragmentieren. Ist ein Paket zu groß, bekommt er eine Fehlermeldung und muss die maximale Paketlänge (Maximum Transmission Unit MTU) anpassen. Es wird empfohlen, die Paketlänge per Discovery zu ermitteln, um die MTU für jeden Pfad anzupassen. IPv6 spezifiziert eine minimale Paketlänge von 1.280 Bytes, die jeder Knoten zu garantieren hat [11].

3.5.2 Auswirkungen auf andere Protokolle und APIs

TCP und UDP Bei beiden Protokollen musste die Berechnung der Prüfsumme angepasst werden, da sie sich zusätzlich neben dem Header und den Daten auch aus den Adressen des IP-Headers berechnet.

IPSec, insbesondere IKE Mit der Integration von IPSec ist es nun möglich, auch 128bit lange Host-Adressen zu nutzen. Z.Zt. gibt es für IPSec, als auch für IKE, keine Unterstützung der unterschiedlichen Adresstypen, wie z.B. für Anycast. Das verdeutlicht die genannte Strategie bei der Entwicklung von IPSec, möglichst unabhängig von der IP Version einsetzbar zu sein. Es wird deshalb nur Unicast unterstützt. IKE selbst ist deshalb bis auf die Länge der Host-Adressen nicht von Veränderungen betroffen. Für den Einsatz des IPSec Sicherheitsprotokolls AH in IPv6 ist zu beachten, dass sich die Berechnung der AH-Prüfsumme aufgrund des neuen IPv6-Headers verändert hat, da einige IPv4 Headerfelder nicht mehr existieren.

BSD Socket API Für IPv6 existiert eine neue Protokollfamilie: PF_INET6. Die neue Datenstruktur, die die IPv6 Adresse aufnimmt ist *struct in6 addr*. Weiterhin existiert eine entsprechende protokollabhängige Datenstruktur: *struct sockaddr in6*. Zusätzlich wurden eine Reihe neuer Funktionen eingeführt, die i.A. zur Umsetzung von Namen und Adressen dienen.

3.5.3 Anpassungen an L4VFS und POSIX Umgebung

Das für die Abbildung des BSD Socket API notwendige Interface *net io* aus L4VFS ist grundlegend nicht auf eine bestimmte IP Version ausgelegt, da die Socket Struktur auf eine enttypisierte Bytesequenz abgebildet wird. Allerdings ist die maximale Feldgröße an IPv4 orientiert, d.h. für IPv6 ist sie dementsprechend zu vergrößern. Das impliziert notwendigerweise auch die Anpassung der L4VFS Network Server Stubs.

Die Portierung in eine POSIX Umgebung ist nahezu problemlos möglich, da die verwendete dietlibc bereits fast vollständig für IPv6 erweitert wurde und die entsprechenden neuen Datenstrukturen enthält. Lediglich der Header für die Definition von IPv6 Paketen fehlt.

3.5.4 Anpassungen der Mikro-SINA Komponenten

KMd Für die Anwendung des in dieser Arbeit betrachteten IKEv1 sind, wie im vorigen Abschnitt dargelegt, keine großen Änderungen der KMd notwendig. Sie betreffen nur die verwendete POSIX Umgebung, die IPv6 Funktionalität anbieten und die längeren Host-Adressen, für die der KMd erweitert werden muss. Bei PF_KEY V2 folgt

den Adress-Erweiterungsheadern der SA jeweils eine Struktur der Form *sockaddr*, damit ist die Schnittstelle unabhängig von der IP Version und kann problemlos für IPv6 eingesetzt werden. Für die Nutzung von PF_KEY V2 mit IPv6 im KMd ist deshalb eine Anpassung der Quellen notwendig, die die Kommunikation mit der Key Engine über diese Schnittstelle realisieren. Die evaluierten KMd *isakmpd* und *racoon* implementieren bereits IPv6 und verarbeiten beide IP Versionen parallel.

PF_KEY V2 Adapter Die Änderungen betreffen wie bei den KMd die Erweiterung der PF_KEY V2 Quellen für IPv6, also hier in Bezug auf die Adress-Erweiterungsheader für die 128bit Host-Adressen. Weiterhin müssen die Schnittstellen zum Viaduct mit der neuen Adressbreite versehen werden. Da außerdem keine Änderungen vorzunehmen sind, ist es möglich, den Adapter für beide IP Versionen einzusetzen.

L4TUN Treiber Für Weiterleitung der geschützten und ungeschützten IP-Pakete an das Viaduct aus dem vertrauenswürdigen bzw. nicht vertrauenswürdigen Netz werden innerhalb der L^{4}Linux Instanzen Komponenten benötigt, die solche Aufgaben übernehmen und dafür mit dem Viaduct kommunizieren. Sie werden im aktuellen Prototyp als L4TUN Treiber bezeichnet und sind in den Kern von L^{4}Linux integriert. Die Weiterleitung der mit IPSec geschützten IP-Pakete von der Außenwelt an das Viaduct wird mittels L4TUN, das in L^{4}Linux je ein Sicherheitsprotokoll für ESP bzw. AH installiert, durchgeführt. Sobald also in L^{4}Linux ein IP-Paket eintrifft und als Protokoll des Payloads entweder ESP oder AH eingetragen ist, wird es an den Protokollstub von L4TUN übergeben, der das Paket an das Viaduct überträgt. Der Stub für das Sicherheitsprotokoll ESP bzw. AH ist aber in L^{4}Linux abhängig von der Version des Internet Protokolls, d.h. für IPv6 müssen diese Segmente von L4TUN neu implementiert werden. Die anderen Teile von L4TUN, die die Kommunikation mit dem Viaduct bearbeiten, sind jedoch unabhängig von der IP Version. Somit kann der größte Teil von L4TUN für IPv6 wiederverwendet werden.

Viaduct Das Viaduct ist diejenige vertrauenswürdige Komponente, die entscheidet, ob ein IP-Paket verworfen, verschlüsselt bzw. unbehandelt weitergeleitet werden soll. Sie ermöglicht damit im Falle des Tunnelmodus den Aufbau von VPNs. Um diese Entscheidungen treffen zu können, muss das Viaduct besonders innerhalb der Netzwerkschicht arbeiten, d.h. z.B. konkret im Falle des Tunnelmodus dem IP-Paket einen ESP Header einfügen und einen neuen IP Header voranstellen. Deshalb benötigt das Viaduct eine Implementierung des IP Protokolls. Der aktuelle Prototyp des Viaducts setzt speziell IPv4 um, was ein Hauptgrund für seine minimale Komplexität ist [19]. Für eine Beibehaltung einer guten sicherheitskritischen Evaluierbarkeit, besonders unter dem Gesichtspunkt geringe Komplexität, ist es deshalb sinnvoll, für IPv6 eine neue Version

zu konzipieren. Das IPv4 Viaduct wurde deshalb auf seinen Entwurf untersucht, um Grundkonzepte übernehmen zu können und einen hohen Grad an Wiederverwendbarkeit zu ermöglichen. Im ersten Schritt wurden die vorhandenen Schnittstellen evaluiert, sie sollten weitestgehend erhalten bleiben, um eine gute Migration des IPv6 Viaducts in die bestehende Mikro-SINA Architektur zu unterstützen.

Das Viaduct bietet den anderen Komponenten Funktionen zum Management von SPD und SADB, für beide Funktionsklassen werden jeweils IP-Adressen benötigt. Die Evaluation ergab, das die bestehenden Interfaces unabhängig von der IP Version und deshalb für IPv6 wiederverwendbar sind, lediglich die Abfrage einer SPI mit *sadb get spi* muss für die neue Adressbreite angepasst werden. Die zu übergebenden IP-Adressen werden in Strukturen gekapselt. Da sie intern im aktuellen Prototyp nur 32bit IPv4 Host-Adressen bieten, ist eine Erweiterung für IPv6 notwendig.

Nach der Untersuchung der Schnittstellen zur Mikro-SINA Umgebung wurde das innere Modell des Viaducts näher analysiert. Die Fokussierung des aktuellen Viaducts auf IPv4 ist folgerichtig in dem Teil des Entwurfs wiederzufinden, der das IP Protokoll abbildet. Aufgrund des anderen Aufbaus von IPv6, speziell des neuen Header-Konzeptes, ist dieser neu zu entwickeln. Das neue, objektorientierte Modell, dargestellt in Abbildung 3.6, basiert auf der Klasse *IPv6 Paket*, die eine Menge von verketteten Erweiterungs-Headern referenziert, das können z.B. der Fragment-Header oder der Hop-by-Hop-Header sein. ESP und AH sind ebenfalls Erweiterungs-Header, sie erben von *IPSec Header*, das die Gemeinsamkeiten beider IPSec Protokolle aggregiert. Ein minimales Paket besteht lt. Spezifikation nur aus dem Basis-Header, d.h. im Modell aus einem Objekt der Klasse *IPv6 Paket*.

Neben dem kompletten Austausch des Protokollentwurfes für IPv6 sind alle Komponenten zu verändern, die die eigentliche Verarbeitung der Pakete vornehmen. Sie benötigen notwendigerweise Wissen über den internen Aufbau von IP und sind im aktuellen Prototyp auf IPv4 ausgelegt. Konkret sind das i.A. die Klassen, die IP-Pakete mit IPSec schützen bzw. aufgrund der Policy unbehandelt weiterleiten. Der Entwurf dieser Komponenten kann übernommen werden, da die Schnittstellen nicht auf eine bestimmte IP Version festgelegt ist. Damit ist nur die Implementierung auf IPv6 umzustellen.

In diesem Zusammenhang werden weitere Veränderungen an SA und Policies notwendig, da sie momentan auf IPv4 ausgelegt sind. Jede SA wird im Viaduct über SPI, Ziel-IP-Host-Adresse und Typ eindeutig referenziert. Die Ziel-Adresse muss deshalb für IPv6 verlängert werden. Das gleiche Problem trifft auf die Policies zu, hier sind neben der Quell- und Ziel-Adresse die Netzwerkmasken anzupassen. Die Erweiterungen der SA und Policies ziehen konsequenterweise auch Veränderungen an der SADB und SPD nach sich, das betrifft wiederum alle Schnittstellen, die IP-Host-Adressen verarbeiten.

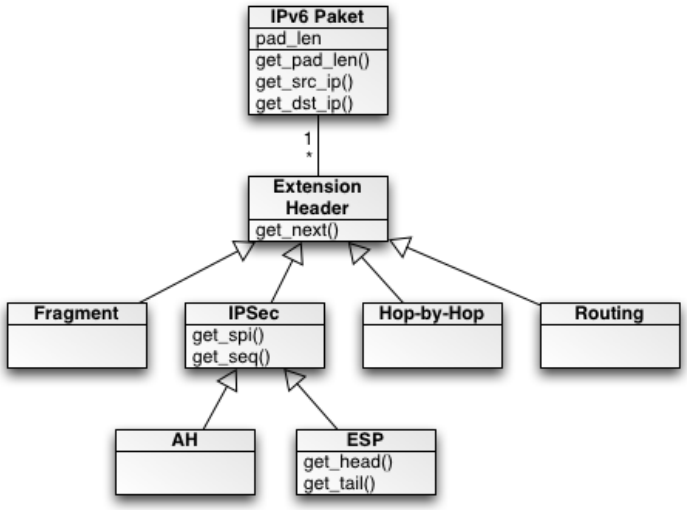

Abbildung 3.6: IPv6 Protokoll-Entwurf für das Viaduct. Die Klasse IPv6 Paket kapselt den Basisheader und bietet Zugriff auf die Erweiterungsheader. Ein minimales IPv6 Paket besteht nur aus dem Basisheader. Die IPSec-Header sind speziellere Erweiterungsheader, ESP z.B. benötigt eine Funktion *get tail()* zum Zugriff auf den ESP_Trailer nach dem Payload.

Zusammenfassend zeigte die Evaluation des IPv4 Viaducts, dass der vorhandene Entwurf bis auf das spezifische IP Modell für die IPv6 Version übernommen werden kann. Es sind zahlreiche Komponenten aufgrund der neuen Host-Adressbreite anzupassen. Neben dem neuen IPv6 Protokollmodell sind die Klassen neu zu implementieren, die die IP-Pakete bearbeiten müssen, z.B. mit IPSec.

Schlussfolgerungen Die notwendigen Veränderungen der Mikro-SINA IPSec Infrastruktur für IPv6 zeigen, dass die Gesamtarchitektur grundlegend unabhängig von der IP Version ist, d.h. es müssen keine neuen Elemente eingeführt werden. Anpassungen sind nur innerhalb bzw. an den Schnittstellen der Komponenten notwendig. KMd, PF_KEY V2 Adapter und L4TUN können mit den entsprechenden Änderungen parallel für beide IP Versionen eingesetzt werden. Für das Viaduct sollte unter Berücksichtigung der Komplexität eine separate Version für IPv6 umgesetzt werden.

Kapitel 4

Implementierung

Aufgrund der Entwurfskriterien habe ich mich entschieden, einen vorhandenen KMd nach L4 zu portieren. Es gibt mehrere gut evaluierte IKE Applikationen, die auch auf Interoperabilität vom VPNC [7] untersucht worden sind. Das VPNC Konsortium agiert als internationale Handelsorganisation auf dem VPN Markt. Zu ihm gehören namhafte Unternehmen wie Cisco Systems, Microsoft und Nokia. Das VPNC hat einige Referenzimplementierungen ausgewählt, gegen die neue IKE Produkte getestet werden müssen.

Eine der KMd Referenzimplementierungen ist der *isakmpd* aus dem OpenBSD Projekt, für den ich mich entschieden habe, weil er neben IKE offen für neue Implementierungen von ISAKMP ist, sowie sehr gut evaluiert und dokumentiert wurde. Den FreeS/WAN KMd Pluto habe ich nicht in Betracht gezogen, da das FreeS/WAN Projekt eingestellt wurde.

Da die KMd, im Weiteren der *isakmpd*, Userland Applikationen sind, benötigen sie eine entsprechende Umgebung an Bibliotheken, wie z.B. eine libc. Die notwendigen Funktionen können auf verschiedene Weise zur Verfügung gestellt werden, in meiner Arbeit habe ich mich entschieden, eine POSIX Umgebung zu schaffen. Das hat in Bezug auf die in Kapitel 3 gestellten Designkriterien mehrere Vorteile:

- gute Austauschbarkeit der Komponenten, besonders der KMd
- hoher Grad an Wiederverwendbarkeit, die implementierten POSIX Funktionen werden von einer Vielzahl anderer Software benutzt, damit wird deren Portierung nach L4 stark vereinfacht
- gute Evaluierbarkeit, die POSIX Funktionen sind gut dokumentiert und standardisiert

Für L4Env wird die in Entwicklung befindliche dietlibc [31] als POSIX Umgebung genutzt, die gegenüber der GNUlibc deutlich schlanker ist, sowie sich gut eignet für

statisch gelinkte Software. Mit der Nutzung der dietlibc für *isakmpd* soll damit auch eine deutlich schlankere und so besser evaluierbare Software für die Mikro-SINA Architektur erzeugt werden. Da bestimmte POSIX Funktionen aus der dietlibc nicht einfach für L4 übernommen werden konnten, z.B. das BSD Socket API, waren sie neu zu entwerfen. Die L4Env Version der dietlibc bietet dafür das Konzept der *Backends*. Sie bieten die Möglichkeit, vorhandene POSIX Funktionen neu zu implementieren und bei Bedarf zur Applikation hinzuzulinken. Während der Arbeit sind einige neue Backends entstanden, die für *isakmpd* und auch für die Portierung anderer Software mit Hilfe einer POSIX Umgebung notwendig sind. Einige der Backends werden in den folgenden Abschnitten vorgestellt.

4.1 Abbildung des BSD Socket Interfaces auf L4

Nachdem in 3.4.1 die grundlegende Struktur der IDL-Schnittstellen im L4VFS für die Socket-Kommunikation vorgestellt worden ist, möchte ich nun einige Implementierungsdetails erläutern. Beispielhaft sollen zwei Socket-Funktionen herausgegriffen werden:

```
int accept(int s, struct sockaddr *addr, socklen_t *addrlen);
int sendmsg(int s, const struct msghdr *msg, int flags);
```

Der Aufruf von *accept* blockiert solange, bis eine Anfrage auf dem Socket s eintrifft. Er gibt dann das Handle auf einen neuen Socket zurück, der die Verbindung repräsentiert.

Das Problem bei der Abbildung von *accept* auf IPC ist der Parameter *addrlen*, er ist lt. Definition eine “value-result” Variable, d.h. initial gibt sie *sizeof(addr)* an und nach Beendigung der Funktion die Größe, die für *sizeof(addr)* gebraucht worden wäre, d.h. *addr* kann u.U. zu klein gewesen sein.

Der IDL-Compiler benutzt *addrlen* als Größe des zu übertragenden IPC Puffers für *addr*. Nach Beendigung der IPC wird der Puffer mit Hilfe der Größe von *addrlen* nach *addr* kopiert. Für den Fall aber, dass *addrlen* initial zu klein war und im Server größer als *sizeof(addr)* wird, tritt im Client ein Buffer Overflow ein. Abbildung 4.1 zeigt den Fehlerfall anhand des übergebenen Wertes von *addrlen*. Deshalb wird für die eigentlich benötigte Größe von *addr*, die am Ende von *accept* zurückgeben wird, ein zusätzlicher Parameter für die IPC benötigt. In der IDL hat die Funktion folgende Signatur:

```
int accept([in] object_handle_t fd,
           [in, out, size_is(addrlen)] char addr[],
           [in, out] int *addrlen,
           [out] int *actuallen);
```

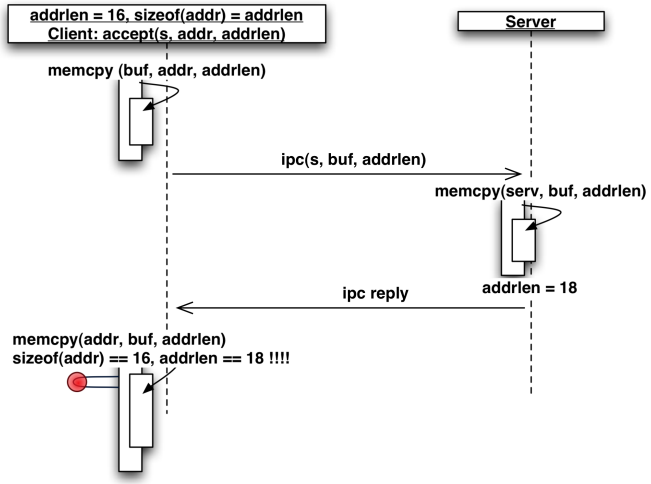

Abbildung 4.1: Abbildung von accept auf IPC. Da *addrlen* > *sizeof(addr)* nach Rückkehr aus der IPC ist, gibt es einen Buffer Overflow beim Kopieren von buf nach addr.

Die Funktion *sendmsg* dient zur Übertragung einer Nachricht an den Socket s. Das Problem bei der Abbildung der Funktion liegt an der komplexen Struktur *msghdr*. Sie enthält eine Menge von Zeigern auf IO-Bereiche, die an s übertragen werden sollen. Sie können an beliebigen Stellen im Speicher liegen. Z.Zt. gibt es noch keine standardisierten Konzepte seitens der IDL-Compiler, solche komplexen Strukturen via IPC zu übertragen. Deshalb muss die Struktur *msghdr* in einzelne Elemente zerlegt versendet und auf der Server-Seite wieder zusammengesetzt werden. Die Menge der IO-Bereiche werden dabei in einen linearen Puffer kopiert, der komplett übertragen wird. Der Server hat die Zeiger auf den Beginn der einzelnen IO-Bereiche innerhalb des linearen Puffers neu zu berechnen. Nachteile der Zerlegung und des Kopierens in den Puffer sind ein erhöhter Speicherbedarf und zusätzlich benötigte Rechenzeit.

Die einzelnen IDL-Funktionen sind nicht an eine bestimmte Umgebung gebunden und können losgelöst davon eingesetzt werden. Für den Einsatz in einer POSIX Umgebung, d.h. in meinem Fall für eine Einbindung in die dietlibc, wurde ein entsprechendes Backend für Socket-Funktionen implementiert. Es bildet das BSD Socket API auf die IDL-Funktionen ab, so müssen z.B. die IO-Bereiche von sendmsg in den linearen Puffer gelegt werden. Weiterhin werden die L4 Fehlermeldungen auf POSIX Fehler abgebildet. Für die zu portierende Software ist es deshalb transparent, welche Implementierung für die BSD Socket Funktionen sich im jeweiligen Backend verbirgt, solange die Semantik nicht verändert wird.

4.2 Abbildung des synchronen IO/Multiplexing von POSIX *select*

Der KMd *isakmpd* arbeitet wie die meisten UNIX UDP-Server event-basiert, d.h. mit einer select-Routine, gefolgt von einem Aufruf der verantwortlichen Handler für die einzelnen File Deskriptoren. Für die zu schaffende POSIX Umgebung wurde deshalb die Entscheidung getroffen, die Funktionalität von *select* auf L4 abzubilden.

Unter POSIX *select* wird ein synchrones I/O Multiplexing verstanden, d.h. ein Prozess kann auf mehrere Ressourcen blockierend gleichzeitig warten, ohne ständig pollen zu müssen. Er verlässt den Aufruf von *select*, wenn mindestens eine Ressource aktiv wird, d.h. der Prozess kann Daten von ihr empfangen oder ihr senden, ohne dabei selbst zu blockieren. Die Funktion hat folgende Schnittstelle:

```
int select(int n, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

Es werden drei Mengen von File Deskriptoren übergeben:

- **readfds** Beobachtung, ob ein Lesen blockiert.
- **writefds** Beobachtung, ob ein Schreiben blockiert.
- **exeptfds** Beobachtung, ob eine Ausnahme (Exception) eintritt. Sie wird z.B. bei Sockets ausgelöst, wenn “out-of-band” Daten zum Empfang bereit sind. Das sind solche Daten, die außerhalb des normalen Datenstromes liegen.

Die Variable **n** enthält den größten File Deskriptor aller drei Mengen plus eins. Nach Rückkehr aus *select* wird die Anzahl der Deskriptoren zurückgegeben, die nicht blockierend sind. Mit **timeout** kann die maximale Wartezeit festgelegt werden, nach der *select* zurückkehren soll.

Bei einem monolithischen Betriebssystem ist *select* ein synchroner Aufruf in den Kern, der den User-Prozess blockiert. Im Gegensatz dazu kann bei dem Mikrokernsystem L4/Fiasco via IPC nur auf einem Server blockiert werden und nicht auf eine Menge m mit $|m| > 1$. Es muss deshalb das synchrone Verhalten von *select* auf eine asynchrone Kommunikation mit einer Menge von Servern abgebildet werden.

Ein erster, intuitiver Ansatz ist die Delegierung der *select* Funktionalität in die jeweiligen Server. Daraus entstehen mehrere Probleme:

1. Der Client übergibt den m Servern jeweils eine Menge von n File Deskriptoren, wobei gilt $|m| > 1$ und $|n| >= 1$ und wartet, bis mindestens ein Deskriptor s aus n bereit wird während des gesetzten Timeout. Dann kehrt der *select* Aufruf lt. Definition zurück. Allerdings sind nun die restlichen m - 1 Server immer

noch blockiert und könnten nur mittels “harten” Rücksetzens der Registerwerte (exregs) wieder aus diesem Zustand herausgelöst werden. Die Erzeugung eines Threads pro s in der Task des Servers ist nur eine Verschiebung des Problems, da die blockierten Threads, auf die nicht mehr gewartet wird, auch mittels exregs zurückzusetzen sind.

2. Wenn in der Latenzzeit bis zum Absenden der letzten *select* Nachricht des Clients das s eines Servers bereit wird, muss dieser Server warten, da der Client noch keine Antworten entgegennimmt.

Deshalb blockiert nur der Client-Thread selbst beim Aufruf von *select*, die asynchrone Kommunikation mit einer Menge von Servern wird auf ein neues Konzept abgebildet. Der Entwurf wird mit Abbildung 4.2 illustriert, die nachfolgenden Interaktionsschritte entsprechen den Nummern in der Abbildung.

Der select-Thread des Clients meldet sich bei seinem *Notification-Listener* Thread an (1), dieser wird für die Lösung des zweiten Problems benötigt. Der *Notification-Listener* Thread hat die Aufgabe, auf entsprechende *Notification* Nachrichten der Server zu warten und danach den select-Thread aufzuwecken. In (2) verschickt der select-Thread des Clients nun für jeden File Deskriptor s an den jeweiligen Server eine *Notification-Request* Nachricht. Sie zeigt dem Server, dass der Client an einer Änderung des Zustandes von s auf “nicht blockierend” interessiert ist. Danach legt sich der Client mittels eines Semaphors schlafen. Sobald eine ihn betreffende *Notification* beim *Notification-Listener* eingetroffen (4) ist, wird der Client vom *Notification Listener* geweckt (5) bzw. kehrt nach Ablauf des Timeout automatisch aus dem Semaphore zurück. Danach meldet der select-Thread des Clients i.A. alle File Deskriptoren bei den Servern mittels *Notification-Clear* wieder ab (6).

Damit wird das synchrone *select* auf eine Menge von Nachrichten abgebildet, die die einzelnen Server nicht blockieren. Die Notification-Implementierung des Servers kann damit auch an seine entsprechende Architektur angepasst werden und ist für den Client völlig transparent. In Abbildung 4.2 wird z.B. in (3) ein eigener *Notification Sender* für die gesamte Task initialisiert, der das Versenden der *Notifications* übernimmt.

Die POSIX Umgebung des *select* Aufrufes im Client wurde durch die Implementierung eines Backends in der dietlibc realisiert. Es kapselt die Erzeugung des *Notification Listeners* der Client Task und versendet die *Notification-Request* bzw. *Notification-Clear* Nachrichten an die Server. Zur Veranschaulichung der Interaktionen soll der nachfolgende Pseudo-Code dienen, den das Backend implementiert:

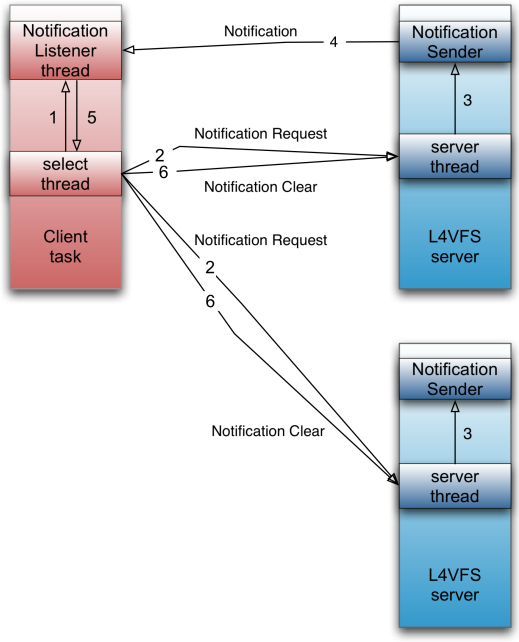

Abbildung 4.2: Abbildung von POSIX *select* auf eine asynchrone Kommunikation in der Mikrokernumgebung L4/Fiasco.

```
begin select
register at notification listener

for each file descriptor s
  send notification_request(s) to the responsible server

semaphore_down_timed

determine non blocking file descriptors

for each file descriptor s
 send notification_clear(s) to the responsible server
end select
```

Für FLIPS wurde der *select* Mechanismus durch die Wiederverwendung von Linux Kernel Code realisiert. Für jeden Socket existiert im Linux-Kern eine Warteschlange (WaitQueue), in die sich Prozesse eintragen, die auf eine Zustandsänderung des Sockets warten. Für FLIPS existiert nun pro Task ein Thread, in Abbildung 4.2 der *Notification*

Sender, dessen ID in die Warteschlange des entsprechenden Sockets eingetragen wird, sobald die *Notification-Request* eines Clients eintrifft. Der *Notification Sender* Thread von FLIPS wird geweckt, sobald ein Socket bereit geworden ist, d.h. er sendet danach dem zugehörigen *Notification Listener* des Clients eine *Notification*.

4.3 Socket Umgebung für die Portierung von Linux Kernel Netzwerk Treiber

Der TCP/IP Server FLIPS basiert im wesentlichen auf der Portierung von IP Linux Kern Code nach L4, unterstützt durch die für Linux Geräte Treiber vorhandene L4 Umgebung *dde linux*. Nach Einführung des BSD Socket API in L4VFS wurde FLIPS auf deren Schnittstellen umgestellt und ist somit ein L4VFS Network Server. FLIPS implementiert L4VFS mit Hilfe der portierten BSD Socket Schicht aus dem Linux Kern. Sie bietet ein transparentes Interface für Linux Netzwerk Protokolle, so im Falle von FLIPS für die Suite TCP/IP. Jedes Netzwerkprotokoll, wie z.B. IP, muss sich bei der BSD Socket Schicht im Kern anmelden, dafür gibt es die Funktion *sock register*. Übergeben wird dabei der Zeiger auf eine Struktur vom Typ *net proto family*, die den Namen des Protokolls enthält, z.B. bei IPv4 PF_INET. Weiterhin hat die Struktur eine Funktion zur Erzeugung eines Sockets der Protokollfamilie. Diese bindet die entsprechenden Funktionen wie accept oder sendmsg an den Socket, da sie protokollabhängig implementiert sein können.

Im Hinblick auf Wiederverwendung wurde die Extrahierung der portierten BSD Socket Schicht des Linux Kerns aus FLIPS evaluiert, als Socket Umgebung für die Portierung von weiteren Linux Netzwerkprotokollen nach L4. Da die Socket Schicht nahezu keine Querverbindungen zu den tieferen Ebenen hat, konnte diese Extrahierung unterstützt durch das *dde linux* vorgenommen werden. Die implementierte Bibliothek *socket linux* ermöglicht damit die vereinfachte Portierung von Linux Netzwerkprotokollen nach L4.

4.4 PF_KEY V2 als generische Schnittstelle für SAs und Policies

Die Implementierung von PF_KEY V2 in einem Adapter zwischen KMd und Viaduct als Schnittstelle für die Etablierung und das Management der SAs, sowie der Policies, soll im folgenden Kapitel näher beleuchtet werden. Für die Mikrokernumgebung ist der Adapter wie FLIPS ein L4VFS Network Server, da PF_KEY V2 via BSD Socket API interagiert.

Das Parsen und Interpretieren der PF_KEY V2 Nachrichten ist eine sehr komplexe Aufgabe, da die Nachrichten längenvariabel sind und der Umfang von PF_KEY V2 sehr groß ist. Darum war es effizienter, eine vorhandene Software für den Adapter zu portieren. Weil die PF_KEY V2 Verarbeitung meist im Kern stattfindet, gibt es keine generalisierte Userland Bibliothek, die hätte verwendet werden können. Die vorhandene Linux Netzwerk Umgebung und die neue Bibliothek *socket linux* für L4 legten es deshalb nahe, eine Linux Lösung zu wählen. Ein weiterer Grund war die Wiederverwendung des BSD Socket API für PF_KEY, es wird wie in 3.4.2 dargelegt, einfach als neues, lokales Netzwerkprotokoll im Kern installiert.

Für den Adapter wurde die PF_KEY V2 Schicht aus dem FreeS/WAN Projekt [1] portiert, denn sie bietet eine sehr kompakte und übersichtliche Lösung für den Linux Kern, die sich außerdem sehr gut aus den Quellen herauslösen lies. Ein weiterer positiver Punkt sind die in FreeS/WAN implementieren PF_KEY V2 Extensions für die Verwaltung der SPD, die zu der in 3.4.2 evaluierten minimalen Schnittmenge gehören. Damit kann der Adapter mit einer großen Anzahl unterschiedlicher KMd kommunizieren, so auch mit dem portierten *isakmpd* aus dem OpenBSD Projekt.

Die für Verwaltung der SADB und SPD vorhandenen Funktionen der FreeS/WAN PF_KEY V2 Schicht waren zu substituieren, da sie ursprünglich auf Linux Kern Strukturen abgebildet wurden. Wie in 3.3 untersucht, ist die Verwaltung von SADB und SPD die Aufgabe des Viaducts. Um diese Aufgaben möglichst transparent, d.h. unabhängig von den eingesetzten Implementierungen, vom Adapter zu lösen, wurde das Konzept des *Backends* eingeführt. Es kapselt die Ersetzung der vorhandenen Funktionen für SADB und SPD. In der ersten Version existiert ein statisches Backend, das die Verwaltungsfunktionen auf IPCs an das vorhandene Viaduct, den *nethub*, abbildet. Statisch bedeutet, das es zu Übersetzungszeit dem Adapter hinzugefügt wird. Abbildung 4.3 zeigt die entwickelte Schichtenarchitektur des PF_KEY V2 Adapters.

Mit Hilfe von *dde linux* und *socket linux* konnten die FreeS/WAN Quellen in eine lauffähige, stabile Linux Umgebung unter L4 überführt werden. Die PF_KEY V2 Schicht meldet sich im Adapter während der Initialisierung, wie IP bei FLIPS, bei *socket linux* mit *sock register* an und kann dann über L4VFS von anderen Servern, z.B. den KMd, angesprochen werden. Ein weiterer Vorteil der Verwendung von *socket linux* ist, dass sich die Bibliothek um die Verwaltung der Sockets, d.h. der Verbindungen zu den Clients kümmert, was als Aufgabe des Adapters in 3.4.2 formuliert worden war.

Ein weiteres Implementierungsproblem waren die im ersten Szenario des Abschnittes 3.4.2 und im PF_KEY V2 Standard eingeführten Acquire-Nachrichten. Sie gehören in das jeweilige Backend, das die Kommunikation zur SADB bzw. SPD implementiert, weil sie nur vom Viaduct getriggert werden können. Es hat als einzige Instanz den exklusiven, direkten Zugriff auf die SADB, sowie SPD und besitzt daher das Wissen, wann eine SA abgelaufen ist bzw. wenn eine neue SA zu erstellen ist. Dann muss der Adapter vom Viaduct automatisch eine Acquire- Nachricht bekommen, damit er

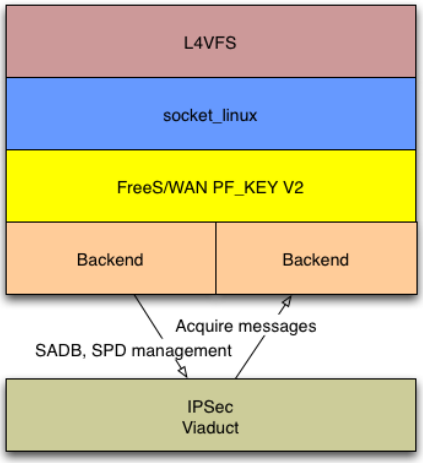

Abbildung 4.3: Schichtenarchitektur des PF_KEY V2 Adapters

die SA aushandeln kann. Deshalb benötigt der Adapter einen zusätzlichen Thread, der sich beim Viaduct für den Empfang solcher Benachrichtigungen anmeldet, bei Erhalt eine entsprechende PF_KEY V2 Nachricht generiert und an die KMd weiterschickt. Die grundsätzliche Implementierung, d.h. die Erzeugung der PF_KEY V2 Nachricht, war bereits in FreeS/WAN enthalten, es mussten lediglich die erhaltenen Daten vom Viaduct im Backend überprüft und an die entsprechenden Funktionen weitergereicht werden.

Da die PF_KEY V2 Schicht in FreeS/WAN die Benachrichtigung der Linux Prozesse über die Warteschlangen der Sockets löst, war eine Implementierung von *select* im Adapter notwendig. Dafür konnte die bereits in FLIPS eingeführte Lösung wiederverwendet werden, denn beide Server stützen sich auf die gleichen Mechanismen ab. FreeS/WAN legt dafür die Datenpakete, d.h. z.B. bei einem Acquire, auf die Receive Queue des zugehörigen Sockets. Da die Clients, im meinem Szenario z.B. *isakmpd*, sich mit einem *select* Aufruf schlafen legen, werden sie geweckt, sobald die Daten in diese Queue übertragen worden sind.

4.5 Hilfsapplikationen

4.5.1 L4LX_FLIPS

Für die Unterstützung eines Rapid Prototyping bei der Implementierung und Portierung neuer BSD Socket API Funktionalität in L4 wurde für L^{4}Linux ein Programm entwickelt, welches das gleiche Interface wie FLIPS und PF_KEY V2 Adapter anbietet, d.h. ein L4VFS Network Server ist. Als Userland Applikation mit L4 Funktionalität

leitet es die BSD Socket Aufrufe von Clients in den L^{4}Linux Kern weiter. Damit kann man relativ einfach die Kommunikation zwischen POSIX Clients und den L4VFS Network Servern testen. Weiterhin kann es sich als sehr nützlich erweisen bei Szenarien, die sicherheitskritische Funktionalität in einen L4 Server auslagern wollen und nicht unbedingt ein minimalistisches FLIPS benötigen. In diesen Modellen kann eine vorhandene L^{4}Linux Instanz für die Socketfunktionen als Legacy System mit Hilfe von L4LX_FLIPS wiederverwendet werden.

4.5.2 IPSec Admin

Neben der automatischen Etablierung der SAs ist es oft notwendig, eine manuelle Administration der SADB bzw. SPD zur Laufzeit vornehmen zu können. Alle evaluierten IPSec Infrastrukturen bieten deshalb ein Administrationstool an, das wie die KMd via PF_KEY V2 Nachrichten mit der IPSec Implementierung im Kern kommuniziert. Ein solches Tool bietet als weiteren Vorteil eine einfache und gute Testmöglichkeit der IPSec Infrastruktur, speziell der PF_KEY V2 Komponenten. Im Rahmen des Diploms wurde deshalb das Administrationstool *IPSecadm* aus dem OpenBSD Projekt portiert. Für das Tool müssen die gleichen Sicherheitskriterien wie für KMd gelten, da es die SADB bzw. SPD manipuliert, d.h. z.B. die Schlüssel übergibt. Folglich kann *IPSecadm* nicht wie ursprünglich als root-priviligierte Userland Applikation in einer Legacy Linux Instanz laufen, weil dieser nicht vertraut werden kann. Deshalb wird *IPSecadm* in einer eigenen L4 Task ausgeführt. Als Eingabeschnittstelle wurde das Fenstersystem DOpE [14] gewählt, weil es durch seine geringe Code-Komplexität von weniger als 10.000 Zeilen als vertrauenswürdig angesehen wird. *IPSecadm* nutzt das Terminal-Widget von DOpE, das von einem L4VFS Terminalserver verwaltet wird. Der Einsatz eines Terminalservers impliziert, das dieser Server eine weitere vertrauliche Komponente ist, weil er die einzelnen Zeichenketten von *IPSecadm* verarbeitet. Für eine sicherheitskritisch minimalisierte Version muss deshalb in einer weiteren Arbeit evaluiert werden, ob eine eigene Implementierung innerhalb von *IPSecadm* Vorteile hat. In der Praxis sind Testszenarien das Haupteinsatzgebiet von *IPSecadm*, da in Endsystemen wie SINA oft keine grafische Oberfläche wie DOpE zur Verfügung steht, die SAs werden meist mit Smartcards oder anderer Hardware in das System eingefügt.

Kapitel 5

Evaluation

5.1 Vergleich POSIX Umgebung gegenüber anderen Konzepten

Mittels der in 3.1 gestellten Kriterien wurde für die Entwicklung des KMd und weiterer Komponenten die Entscheidung getroffen, eine POSIX Umgebung für die Portierung der Software zu schaffen. Daraus resultieren einige wichtige Vorteile, besonders in Hinblick auf die Schutzziele Vertraulichkeit und Integrität, aber auch in Bezug auf die Fehlerwahrscheinlichkeit:

1. Minimale Anpassungen. Die zu portierende Software muss bestenfalls nur sehr geringfügig verändert werden. Wenn stattdessen POSIX durch komplett neue, andere Softwareteile ersetzt wird, dann impliziert das eine erhöhte Fehlerwahrscheinlichkeit, die sich u.U. auch auf andere, nicht veränderte Teile auswirken kann.

2. Verbesserte Evaluation. Die Verwendung der POSIX Interfaces unterstützt, dass die Software nicht neu bzgl. dieser Funktionen evaluiert werden muss, da die Schnittstellen unverändert übernommen werden. Es müssen lediglich die Backends evaluiert werden, die POSIX auf L4 abbilden. Für sie muss garantiert sein, das sie die Semantik der Funktionen korrekt nachbilden und die definierten Rückgabewerte einhalten.

Ein Nachteil ist der erhöhte Aufwand an Zeit für Entwurf und Implementierung gegenüber einer proprietären Lösung, der aber dadurch aufgewogen wird, dass eine Vielzahl an Software mittels der entwickelten POSIX Umgebung einfacher portiert werden kann.

5.2 Komplexität des Quellcodes

Bei sicherheitskritischen Architekturen wie Mikro-SINA ist es notwendig, die Komplexität der Komponenten zu untersuchen, d.h. im Rahmen einer Evaluation auch den Umfang an Quellcode. Je größer diese Zahl ist, umso schwieriger wird es sein, die Nachweisbarkeit der Ziele Vertraulichkeit, Integrität und Verfügbarkeit zu gewährleisten.

Mehrfach wurde bereits angedeutet, dass IPSec und ISAKMP/IKE sehr umfangreich sind. Daraus resultiert zwangsläufig eine erhöhte Komplexität des Quellcodes. Die Einbettung des KMd *isakmpd* in eine POSIX Umgebung bietet die nahezu unveränderte Portierung des Quellcodes nach L4. Hierdurch kann auch die gegebene Zahl von etwa 30.000 Zeilen des *isakmpd* aus [16] angenommen werden (ohne plattformabhängige Teile und Tests). Diese sehr große Zahl kommt besonders dadurch zustande, weil *isakmpd* nicht nur eine reine IKE-Implementierung ist, sondern auch Umsetzung des Frameworks ISAKMP, wie in Abschnitt 2.6.3 erläutert. Der sicherheitskritische und unbedingt notwendige funktionale Anteil für IKE beschränkt sich nach eigenen Messungen auf ca. 14.000 Zeilen.

Weiterhin besteht auch ein hoher Bedarf an Rechenzeit und Speicherressourcen. So wird z.B. bei jeder Etablierung einer SA in beiden Modi von IKE mindestens eine Diffie Hellmann (DH) Exponentiation benötigt. DH gilt als sehr rechenintensiv, denn der Schlüssel k eines Partners berechnet sich aus $k = z^a \ mod \ p$, wobei a ein geheimer Wert und z vorher ausgetauscht worden ist.

Neben dem KMd wird auch der PF_KEY V2 Adapter benötigt. Er wird für die Interpretation der PF_KEY V2 Nachrichten und deren Abbildung auf das Viaduct eingesetzt. Die eigene Implementierung besteht aus 8.300 Zeilen Quellcode, wobei 5.500 Zeilen aus dem FreeS/WAN Projekt portiert worden sind. Der Codeanzahl kann nicht verkleinert werden, da der Adapter bereits auf das minimal notwendige Funktionsniveau gesenkt wurde.

Insgesamt kostet die Erweiterung der IPSec Infrastruktur für automatische SA Etablierung somit mindestens 22.300 Zeilen Quellcode. Diese Zahl setzt sich aus dem minimal benötigten Anteil des *isakmpd* und dem Quellcode des PF_KEY V2 Adapters zusammen. Als Vorteil ist die Verteilung der beiden Anteile auf zwei verschiedene Adressräume zu werten, die funktional getrennt sind. Damit können sie unabhängig voneinander sicherheitskritisch evaluiert werden.

Kapitel 6

Zusammenfassung und Ausblick

Fazit Im Rahmen dieser Arbeit wurde die vorhandene Mikro-SINA IPSec Infrastruktur von mir evaluiert und mit Komponenten zur automatischen Etablierung von Security Associations für die geschützte Kommunikation via IPSec erweitert. Ich habe Entwurfskriterien eingeführt, die zur Entwicklung der erweiterten Architektur als Leitfaden dienten. Sie wurden bei der Herausarbeitung und Implementierung der notwendigen Schnittstellen für die Interaktion der Komponenten in Betracht gezogen. Als Schnittstellen wurden das BSD Socket API für die Kommunikation via Sockets und PF_KEY V2 für das Management der Security Associations auf L4 abgebildet und implementiert. Die neue Architektur wurde bzgl. Vertraulichkeit, Integrität und Verfügbarkeit evaluiert. Mit Hilfe der zwei behandelten Szenarien konnte ich zeigen, dass die erweiterte Infrastruktur die gestellten Entwurfsziele, besonders in Bezug auf die Schutzziele, erfüllt. Außerdem wurde mittels der Kriterien nachgewiesen, dass die neue IPSec Infrastruktur die bestehende erweitert, d.h. für ein minimales Vertrauensszenario kann der zu Beginn der Arbeit vorhandene Prototyp unverändert eingesetzt werden. Weiterhin wurde Mikro-SINA bzgl. IPv6 in Hinblick auf notwendige Ergänzungen bzw. Änderungen der Komponenten untersucht. Für IPv6 habe ich einen erweiterten Entwurf der Mikro-SINA Architektur vorgestellt.

Im praktischen Teil der Implementierung sind die entworfenen Komponenten von mir umgesetzt worden, so u.a. ein PF_KEY V2 Adapter und eine Portierung des KMd *isakmpd* aus dem OpenBSD Projekt. Unter den Gesichtspunkten Wiederverwendbarkeit und Austauschbarkeit ist eine POSIX Umgebung zur Lauffähigkeit der KMd geschaffen worden. Sie erleichtert außerdem die Portierung von weiterer, auf dem BSD Socket API aufbauender, Software nach L4. So entstand schon während dieser Arbeit unter Verwendung der erzeugten POSIX Umgebung ein Server für die Kommunikation via Unix Domain Sockets. Weiterhin habe ich die BSD Socket Schicht aus dem Linux Kern herausgelöst, sie kann als Umgebung für die Portierung von Linux Netzwerkpro-

tokollen benutzt werden. Sie wurde bisher konkret im PF_KEY V2 Adapter und FLIPS eingesetzt.

Ausblick Für die Realisierung der kryptographischen Algorithmen setzt *isakmpd* im aktuellen Prototyp die libdes-Bibliothek ein, die bereits im Viaduct Verwendung findet. Sie bietet jedoch nur einen begrenzen Umfang an Algorithmen. In 3.3.1 ist der Crypto-Server in die IPSec Infrastruktur einbezogen worden, der die kryptographischen Algorithmen implementiert. Als weitere Aufgabe ist deshalb eine Anpassung von *isakmpd* notwendig, um den Crypto-Server anstatt der *libdes* einzusetzen. Damit wird der KMd entlastet und auf seine Kernaufgaben konzentriert. Die portierte Version des *isakmpd* unterstützt nur PSK, d.h. es ist noch keine SA-Etablierung mit Zertifikaten möglich. Ein solche Erweiterung des KMd ist besonders sinnvoll für den Einsatz in der Praxis und sollte deshalb in Betracht gezogen werden.

Für die Entscheidungen, welche Policies auf IP-Pakete anzuwenden sind, setzen die Mehrzahl der evaluierten IPSec Infrastrukturen lediglich die SPD ein. Sie bietet meist nur eine Art Filter auf Netzwerk-Ebene. In vielen Szenarien ist es aber von Vorteil, wenn abstraktere Konzepte zur Entscheidungsfindung herangezogen werden können. Das OpenBSD Projekt verwendet dafür das Konzept der Trust Management Systeme, speziell KeyNote [10]. Trust Management Systeme bieten ein Modell zur Spezifizierung von Security Policies, sie binden dabei Schlüssel direkt an die Autorisierung zur Ausführung bestimmter Aufgaben. Für die Mikro-SINA IPSec Infrastruktur sollte deshalb evaluiert werden, ob und wie solche Systeme innerhalb einer Mikrokern-Umgebung zum Einsatz kommen können. Ein Trust Management System für Mikro-SINA ist zudem auch allgemein für L4Env interessant, da viele Komponenten Sicherheitsmechanismen benötigen. Ein weiterer, notwendiger Punkt ist die Implementierung und Erweiterung von Mikro-SINA für IPv6. In der aktuellen Forschung wird momentan, z.B. im Racoon-Projekt, über eine Umsetzung von IKEv2 nachgedacht. Das neue Protokoll sollte deshalb auch für Mikro-SINA untersucht werden.

Kapitel 7

Glossar

AH Authentication Header, Sicherheitsprotokoll zum Schutz von Integrität und Authentizität der IP-Pakete

ESP Encapsulated Payload, Sicherheitsprotokoll zum Schutz von Vertraulichkeit und Integrität der IP-Pakete

FreeS/WAN Projekt für die IPSec Erweiterung für Linux ab Version 2.2

IDL Interface Definition Language, standardisierte Sprache für netzwerktransparente Schnittstellen zwischen Anwendungen

IKE Internet Key Exchange, Protokoll zur Etablierung und Verwaltung von SAs

IPC Inter Process Communication, Übertragung von Daten zwischen Prozessen, die in verschiedenen Adressräumen liegen

IPSec Internet Protocol Security, Protokoll zur Schaffung von Vertraulichkeit und Integrität für die IP-Schicht

ISAKMP Internet Security Association and Key Management Protocol, Framework für das Key- und SA-Management

OSI Open Systems Interconnection Reference Model, Schichtenmodell zur Beschreibung offener Kommunikationsarchitekturen

KMd Key Management Dämon, User-Space Applikation zur Etablierung der ISAKMP bzw. der IPSec SAs

PF_KEY V2 Generisches Key Management Protokoll, Schnittstelle zur Key Engine im Kern des Betriebssystems

SA Security Association, Sicherheitsvertrag zwischen zwei Kommunikationspartnern, darin Festlegung der Authentifizierungs- und Verschlüsselungsverfahren

SADB Security Association Database, konzeptionelle Datenbank, die die zum aktuellen Zeitpunkt gültigen SAs enthält

SKIP Simple Key Management for Internet Protocols, *in-band* Protokoll für das Key-Management, Schlüsselmaterial wird in das Payload jedes IP-Paketes eingebettet

SPD Security Policy Database, konzeptionelle Datenbank, enthält Regeln ob und wenn ja welche IPSec-Mechanismen auf ein IP-Paket angewendet werden sollen

SPI Security Parameter Index, Identifier, der in jedem IPSec-Paket enthalten ist und auf die lokale Datenstruktur einer SA verweist, in der Praxis ein 32bit Wert

Literaturverzeichnis

[1] FreeS/WAN Linux FreeS/WAN website. URL: `http://www.freeswan.org`.

[2] IKECrack an Open Source IKE/IPSec authentication crack tool. URL: `http://ikecrack.sourceforge.net`.

[3] Openswan - an Open Source IPSec Implementation for Linux. URL: `http://www.openswan.com`.

[4] Sichere Inter-Netzwerk Architektur (SINA). URL: `http://www.bsi.bund.de/fachthem/sina/index.htm`.

[5] Strongswan - an Open Source IPSec Implementation for Linux. URL: `http://www.strongswan.org`.

[6] SunScreen SKIP. URL: `http://wwws.sun.com/software/skip`.

[7] VPNC Virtual Private Network Consortium website. URL: `http://www.vpnc.org`.

[8] W. Aiello, S. M. Bellovin, M. Blaze, and et al. Just Fast Keying: Key Agreement in a Hostile Internet, 2003.

[9] A. Aziz and M. Patterson. SKIP: Simple Key-Management for the Internet Protocol. Conference Proceedings of INET, June 1995.

[10] M. Blaze, J. Feigenbaum, J. Ioannidis, and et. al. RFC 2704: The KeyNote Trust Management System Version 2, September 1999.

[11] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. Internet Engineering Task Force: RFC 2460, December 1998.

[12] H. P. Dittler. *IPv6, das neue Internetprotokoll*. dpunkt Verlag, 2002.

[13] N. Ferguson and B. Schneier. A cryptographic evaluation of IPsec. Technical report, 3031 Tisch Way, Suite 100PE, San Jose, CA 95128, USA, 2000.

[14] Norman Feske and Hermann Härtig. DOpE — a Window Server for Real-Time and Embedded Systems. Technical Report TUD-FI03-10-September-2003, TU Dresden, 2003.

[15] Günter Schäfer. *Netzsicherheit*. dpunkt Verlag, 2003.

[16] S. Hallquist and A. D. Keromytis. Implementing Internet Key Exchange (IKE). USENIX, 2000.

[17] Hans-Jörg Höxer. Portierung des Photuris Key Agreement Dämons von OpenBSD nach Linux. Studienarbeit, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2001.

[18] D. Harkins and D. Carrel. RFC 2409: The Internet Key Exchange (IKE). URL: `http://www.ietf.org/rfc/rfc2409.txt`, November 1998.

[19] C. Helmuth, N. Feske, and A. Warg. Mikro-SINA, Zusammenfassung zum 5. Zwischenbericht. 2004.

[20] C. Helmuth, A. Westfeld, and M. Sobirey. μSINA - Eine mikrokernbasierte Systemarchitektur für sichere Systemkomponenten. In *Deutscher IT-Sicherheitskongress des BSI*, volume 8 of *IT-Sicherheit im verteilten Chaos*, pages 439–453. Secumedia-Verlag Ingelsheim, May 2003.

[21] P. Karn and W. Simpson. RFC 2522: Photuris: Session-Key Management Protocol. URL: `http://asg.web.cmu.edu/rfc/rfc2522.html`, March 1999.

[22] C. Kaufman. Internet Key Exchange (IKEv2) Protocol. URL: `http://www.ietf.org/internet-drafts/draft-ietf-ipsec-ikev2-17.txt`, September 2004.

[23] S. Kent and R. Atkinson. RFC 2401: Security Architecture for the Internet Protocol. URL: `http://www.faqs.org/rfcs/rfc2401.html`, November 1998.

[24] S. Kent and R. Atkinson. RFC 2402: IP Authentication Header (AH). URL: `http://www.faqs.org/rfcs/rfc2402.html`, November 1998.

[25] S. Kent and R. Atkinson. RFC 2406: IP Encapsulating Security Payload (ESP). URL: `http://www.faqs.org/rfcs/rfc2406.html`, November 1998.

[26] Kai Martius. *Sicherheitsmanagement in TCP/IP-Netzen*. Vieweg Verlag, 2000.

[27] D. Maughan, M. Schertler, M. Schneider, and J. Turner. RFC 2408: Internet Security Association and Key Management Protocol (ISAKMP). URL: `http://www.ietf.org/rfc/rfc2408.txt`, November 1998.

[28] D. McDonald, C. Metz, and B. Phan. RFC 2367: PF KEY Key Management API, Version 2. URL: `http://www.faqs.org/rfcs/rfc2367.html`, July 1998.

[29] Andreas Pfitzmann. Sicherheit in Rechnernetzen: Mehrseitige Sicherheit in verteilten und durch verteilte Systeme, 2000.

[30] W. Simpson. IKE/ISAKMP considered harmful. *USENIX ;login*, 24(6), 1999.

[31] F. von Leitner. dietlibc - a libc optimized for small size. URL: `http://www.fefe.de/dietlibc`.