Hochschule Offenburg
University of Applied Sciences
Unternehmens- und IT-Sicherheit

UNITS (WS15/16)

Fakultät für Medien und Informationswesen

Userspace IPsec-Processing unter Windows

Author:
Noel Kuntze

Supervisor:
Prof. Dr. D. Westhoff

31. August 2016

Inhaltsverzeichnis

Abbildungsverzeichnis

Codeverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ABI Application Binary Engine

ACL Access Control List

AEAD Authenticated Encryption with Associated Data

AH Authentication Header

API Application Programming Interface

ARP Address Resolution Protocol

BA Bachelorarbeit

BEET Bound-End-to-End-Tunnel

BGP Border Gateway Protocol

CA Certificate Authority

CBC Cipher Block Chaining

CP Configuration Payload

CRL Certificate Revocation List

DH Diffie Hellman

EAP Extensible Authentication Protocol

ESP Encapsulated Security Payload

FD File Descriptor

GUI Graphical User Interface

HMAC Hashed Message Authentication Code

HSO Hochschule Offenburg

ICMP Internet Control Message Protocol

IETF Internet Engineering Task Force

IKE Internet Key Exchange

ISAKMP Internet Security Association and Key Management Protocol

ISIS Intermediate System To Intermediate System

IP Internet Protocol

IPv4 Internet Protocol version 4

IPv6 Internet Protocol version 6

IPH IP Helper

IPsec Internet Protocol Security

ISP Internet Service Provider

L2TP Layer 2 Tunneling Protocol

MAC Message Authentication Code

MFA MehrFaktorAuthentifizierung

MTU Maximum Transmission Unit

MSS Maximum Segment Size

NAT Network Address Translation

OCSP Online Certificate Status Protocol

OSPF Open Shortest Path First

PPTP Point to Point Tunneling Protocol

PBR Policy Based Routing

PFS Perfect Forward Secrecy

PSK PreShared Key

RFC Request For Comment

RFC Request For Comment

RW Roadwarrior

SA Security Association

SAD Security Association Database

SP Security Policy

SPD Security Policy Database

SPI Security Parameter Index

SSTP Secure Socket Tunneling Protocol

TCP Transmission Control Protocol

TOS Type Of Service

TS Traffic Selector

TTL Time To Live

UDP User Datagram Protocol

WFP Windows Filtering Platform

VICI Versatile IKE Configuration Interface

VM Virtual Machine

VPN Virtual Private Network

VTI Virtual Tunnel Interface

1 Einleitung

1.1 Abstract

Der Mangel an Unterstützung für TUN/TAP-Geräte unter Windows von strongSwan, sowie Fehler und fehlende Unterstützung für Mehrfaktorauthentifizierung im nativen IPsec-Stack von modernen Windows-Versionen zusammen mit dem Mangel an frei verfügbaren Alternativen zu den mangelhaften existierenden Lösungen lässt den Bedarf an einem Client für IPsec-Roadwarrior-VPNs ungedeckt.

Diese Arbeit zielt darauf ab, Unterstützung für den OpenVPN TAP-Treiber für Windows in strongSwan zu implementieren, sodass die Grundlage für die Implementierung eines offenen Clients auf Basis von strongSwan gelegt ist.

1.2 Motivation

Die Motivation für die Arbeit ist der Mangel eines modernen, flexiblen, offenen und gepflegten IPsec-VPN-Clients für die Windows-Platform. Aufgrund der Erfahrungen aus der Unterstützung des strongSwan-Projekts wurde der Bedarf für so eine Software entdeckt, der durch die Ergänzung von fehlender Funktionalität abgedeckt wird. Der Mangel an starker Kryptografie und existierende Implementierungsfehler im nativen Client von neueren Windowsversionen wird oft bemängelt und ist im Jahr 2016 nicht mehr adequat zur Bedrohungslage durch staatliche Angreifer.

2 Grundlage

Um IPsec zu verstehen, müssen zuerst die Netzwerkgrundlagen verstanden werden.

2.1 Netzwerke

Moderne Computernetzwerke basieren auf Ethernet und Internet Protocol version 4 (IPv4) oder Internet Protocol version 6 (IPv6), wie standardisiert von der Internet Engineering Task Force (IETF) in diversen Request For Comments (RFCs). Der Empfang und das Senden von Daten geschieht mit Hilfe von Netzwerkschnittstellen, die die Computer mit dem Netzwerk verbinden. Die Adressierung der Computer im Netzwerk geschieht mithilfe der Internet Protocol (IP)-Adresse des Empfängers. Um bidirektionale Kommunikation zu ermöglichen, enthält ein IP-Paket als Quelle die IP-Adresse des Senders. Die Übertragung von IP-Paketen zwischen verschiedenen Computern findet durch Nutzung der vorhandenen Übertragungswege statt, sei es Ethernet über entsprechende Kabel oder über andere Träger, wie Glasfaser oder Funk. Die Adressierung der Netzwerkschnittstellen erfolgt über deren MAC-Adressen, was für Internet Protocol Security (IPsec) jedoch nicht relevant ist, da nur IP-Pakete oder die Payload davon, also das Protokoll auf der Transportschicht, geschützt werden.

2.2 Routing

Beim Routen eines Pakets wird nach dem Empfang oder Senden eines IP-Pakets durch Nutzung des sogenannten "Routing Table" nachgeschlagen, wohin ein Paket weitergeleitet werden muss, um den Empfänger zu erreichen. Hierbei wird nach dem längsten passenden Präfix im Routing Table gesucht, um die korrekte Route zu finden. Der Time To Live (TTL)-Wert des Pakets wird überprüft und dekrementiert. Wenn er 0 erreicht, wird das Paket verworfen und eine Internet Control Message Protocol (ICMP)-Fehlermeldung an den Absender verschickt.

2.2.1 Policy Based Routing

Policy Based Routing ist eine Sonderform des Routings. Hierbei wird nicht die Zieladresse zum Finden der Route genutzt, sondern eine Regel.

Linux implementiert Policy Based Routing (PBR) durch die die Nutzung von Regeln, die Pakete auf Basis von entweder Firewall-Markierungen, der Ziel- oder Quelladresse, dem Wert des Type Of Service (TOS)-Felds oder der benutzten eingehenden oder ausgehenden Netzwerkschnittstelle in bestimmte Routing-Tabellen umleitet. Die Firewallmarkierung kann ist 32 Bit lang und kann in der Firewall (iptables, nftables, ebtables) auf eigens gewählte Werte und mit eigens gewählten Regeln gesetzt werden. Linux erlaubt Anwendungen die Firewallmarkierung direkt im Netzwerksockel zu setzen.

Windows implementiert kein PBR.

2.3 IPsec

IPsec ist ein Konzept zum Absichern von beliebigen IP-Paketen oder deren Nutzlasten. Es stellt Vertraulichkeit, Authentizität und Schutz vor Replay-Attacken bereit. Die Implementierung der Schutzmechanismen beruht auf sogenannten Security Associations (SAs) und Security Policy

(SP), die genutzt werden um die Daten zu schützen. Die Kombination aus zugehörigen SAs und SPs wird allgemein als CHILD_SA bezeichnet.

IPsec an sich lebt nur im Kernel, jedoch wird für die Aushandlung von Sitzungsschlüsseln, Algorithmen und dem Erneuern derselben eine Komponente im Userland benötigt.

Im Userland existiert in der Regel ein Systemdienst (Daemon), der eine oder mehrere Versionen von Internet Key Exchange (IKE) spricht und so ermöglicht IPsec SAs mit anderen Computern auszuhandeln um IP-Pakete zu schützen. Es steht jedem Systemadministrator jedoch frei die IPsec SAs und SPs manuell zu konfigurieren.[1]

Der Daemon kommuniziert mit dem Kernel und übergibt ihm die ausgehandelten IPsec SAs und SPs, die in der Security Association Database (SAD) und Security Policy Database (SPD) verwaltet werden.

Die Kommunikation zwischen den Diensten wird über Authentifizierungsverfahren wie PreShared Key (PSK), Zertifikatsauthentifizierung, RSA- oder ECDSA-Schlüssel oder Extensible Authentication Protocol (EAP) unter Benutzung des Oakley-Protokols[2] abgesichert, welches mithilfe des Diffie Hellman (DH)-Schlüsselaustauschprotokolls geheime Schlüssel zwischen den Teilnehmern aushandelt.[3][4]

2.3.1 IKE

IKEv1/ISAKMP IKEv1 wird oft als äquivalent zu Internet Security Association and Key Management Protocol (ISAKMP) verwendet, wobei es auch keinen für diese Arbeit bedeutsamen Unterschied gibt. Daher werden die Begriffe hier auch äquivalent genutzt. IKEv1 ist die erste, älteste Version des IKE-Protokolls, welches zum Aushandeln von CHILD_SAs genutzt wird.

IKEv2 IKEv2 ist die logische Weiterentwicklung von IKEv1 und beinhaltet Verbesserungen im Bezug auf den Verbindungsaufbau, Robustheit, Sicherheit und Flexibilität[5]. Mit der neuen Version wurde XAUTH durch EAP ersetzt, was die Delegation der Authentifizierung zu einem oder mehreren RADIUS-Servern ermöglicht und weitere Authentifizierungsmodi ermöglicht. Des weiteren wurde der unsichere "Aggressive Mode entfernt" und der Standard klarer formuliert.

IKE_SA Eine IKE_SA bezeichnet eine Verbindung, die zwei Teilnehmer mittels IKE ausgehandelt haben. Eine IKE_SA wird durch die IP-Adressen, sowie durch Security Parameter Indexes (SPIs) identifiziert. Beiden Teilnehmern ist ein mittels IKE ausgehandeltes geteiltes Geheimnis bekannt, welches genutzt wird um Nachrichten zwischen den Teilnehmern zu verschlüsseln und zu authentisieren.

IKE_SAs haben zu CHILD_SAs eine 1:N-Beziehung. Eine einzelne IKE_SA kann keine, eine oder mehrere CHILD_SAs verwalten.

CHILD_SA Eine CHILD_SA ist die Sammlung aller zugehörigen IPsec SAs und SPs, die zu einem Tunnel gehören. Wenn der Tunnel mit Komprimierung ausgehandelt wurde, so gehören

[1] SK05, S. 18.
[2] Hil98.
[3] Cha+14, S. 50.
[4] Dou+98, S. 8.
[5] Cha+14, S. 136, 137.

die Komprimierungs-SAs, die dabei erstellt werden, auch dazu[6][7].

Das Aushandeln einer CHILD_SA wird unter IKEv1 im sogenannten Quick Mode durchgeführt, der dem Aggressive Mode oder Main Mode folgt. Eine CHILD_SA wird als ganzes verwaltet. Wenn eine einzelne SA gelöscht wird, so wird auch die dazugehörige SA in der Gegenrichtung und die SPs gelöscht. Um die kryptografischen Schlüssel von CHILD_SAs zu erneuern muss sie komplett neu ausgehandelt werden. In verschiedenen Implementierungen wird das Rekeyen von CHILD_SAs mit IKEv1 unterschiedlich gehandhabt. Für IKEv2 ist das Verhalten standardisiert[8].

In der Regel werden unterschiedliche Schlüssel für jede einzelne SA einer CHILD_SA und jeden einzelnen eingesetzten kryptografischen Algorithmus gesetzt.

Das Zuordnen von IPsec-Paketen zu IPsec-SAs wird bewerkstelligt, indem nach dem SPI des Pakets in der SAD gesucht wird.

Unterschiede in Schreibweisen in der Bachelorarbeit und den RFCs Diese spezielle Begrifflichkeit stammt aus dem "strongSwan"-Quellcode, in dem die Datenstrukturen für eine IKE SA und eine CHILD SA Unterstriche enthalten. Um die Begriffe einheitlich zu verwenden wird der Unterstrich immer mitgeschrieben. Dies verdeutlicht auch, dass hier speziell von "strongSwan" gesprochen wird. In den RFCs über IPsec wird ebenfalls von IKE_SAs und CHILD_SAs gesprochen, dort jedoch ohne Unterstrich. Was "strongSwan" ist wird in Unterabschnitt 3.2 erläutert.

PFS Perfect Forward Secrecy (PFS) disjunkt die Schlüssel für die CHILD_SAs von den Schlüsseln für die IPsec-SAs, indem beim Aushandeln einer CHILD_SA ein weiterer DH-Schlüsselaustausch stattfindet[9]. Ohne PFS werden die Schlüssel für die CHILD_SAs vom Schlüssel der IKE_SA abgeleitet. Wenn ein kryptografischer Schlüssel für den Verschlüsselungsalgorithmus einer IPsec SA kompromittiert wird, so kann ein Angreifer nur den Verkehr entschlüsseln, der mit dieser SA verschlüsselt wurde.

Rekeying und Reauthentication Beide IKE-Versionen unterstützen das Reauthentifizieren von IKE SAs und das Rekeyen von CHILD_SAs[10]. Der Zweck der Reauthentifizierung einer IKE_SA ist zu überprüfen, ob die Authentifizierungsinformation eines Teilnehmers noch gültig ist, zum Beispiel ob ein Zertifikat ausgelaufen ist oder gesperrt wurde über eine Certificate Revocation List (CRL) oder mittels Online Certificate Status Protocol (OCSP). Der Zweck von Rekeying ist die kryptografischen Schlüssel in gewissen Abständen zu wechseln, zum Beispiel alle 6 Stunden, oder wenn 4 GB übertragen wurden oder wenn eine gewisse Anzahl Pakete übertragen wurde. Dies wird getan, damit die Menge an Daten, die bei einer Kompromittierung eines Schlüssels offengelegt wird, begrenzt ist.

[6] Abr+01, S. 7.
[7] Cha+14, S. 61.
[8] Cha+14, S. 16.
[9] Cha+14, S. 13.
[10] Cha+14, S. 616.

2.3.2 Modi

IPsec unterstützt verschiedene Modi wie Daten über das VPN versendet werden. Die Modi unterscheiden sich hinsichtlich der Tatsache wie sie die IP-Pakete übertragen

Transport-Modus Im Transport-Modus wird der IP-Header des zu schützenden Pakets nicht übertragen. Der IP-Header wird auf Basis der eingetragenen IP-Adressen der genutzten IPsec-SAs bestimmt und nach der Überprüfung des empfangenen Pakets ergänzt. Der Sinn dahinter ist, dass der Overhead der SAs verringert wird, wenn die Quell- oder Zieladressen der getunnelten Pakete sich nicht von denen der SAs unterscheiden.

Abbildung 1: Transport-Modus

IP-Kopf	*IPsec-Paketkopf*	*Nutzlast (Schicht 4 und höher)*	*IPsec-Anhänger*

Tunnel-Modus Im Tunnel-Modus werden die kompletten IP-Pakete übertragen und mittels des ausgehandelten Protokolls geschützt.

Abbildung 2: Tunnel-Modus

Äußerer IP-Kopf	*IPsec-Paketkopf*	*Innerer IP-Paketkopf mit Nutzlast*	*IPsec-Anhänger*

BEET Bound-End-to-End-Tunnel (BEET) ist ein spezieller Modus, bei dem "virtuelle" IP-Adressen für die Endpunkte ausgehandelt werden. Die IP-Header der übertragenen Pakete werden jedoch nicht mitgesendet, sodass eine Struktur wie im Transport-Modus entsteht. Die Quell- und Zieladressen werden nach dem Empfang des IPsec-Pakets durch Suchen der SPI in der SAD ergänzt, sodass das Paket an eine lokale Anwendung ausgeliefert werden kann.[11]

2.3.3 Protokolle

ESP ESP ist das Standardprotokoll zum Absichern von IPsec-basierten Virtual Private Networks (VPNs), da es Vertraulichkeit, Integrität und Replay-Schutz für die übertragenen Pakete unterstützt. Dabei wird optional ein Verschlüsselungsalgorithmus, sowie ein Hashed Message Authentication Code (HMAC) eingesetzt oder ein Authenticated Encryption with Associated Data (AEAD)-Algorithmus.[12]

UDP Encapsulation UDP Encapsulation ist ESP in einer UDP-Hülle. Es wird standardmäßig eingesetzt, wenn erkannt wurde, dass zwischen den Teilnehmern Network Address Translation (NAT) eingesetzt wird.[13] Es wird auch eingesetzt, wenn einer der Teilnehmer keine reinen ESP-Pakete versenden kann, wie zum Beispiel wenn IPsec im Userspace implementiert ist und aus einem beliebigen Grund kein Sockel implementiert wurde, der es ermöglicht reine ESP-Pakete zu versenden.

[11] Appendix B PRJ15.
[12] Ste05b.
[13] Mar+05.

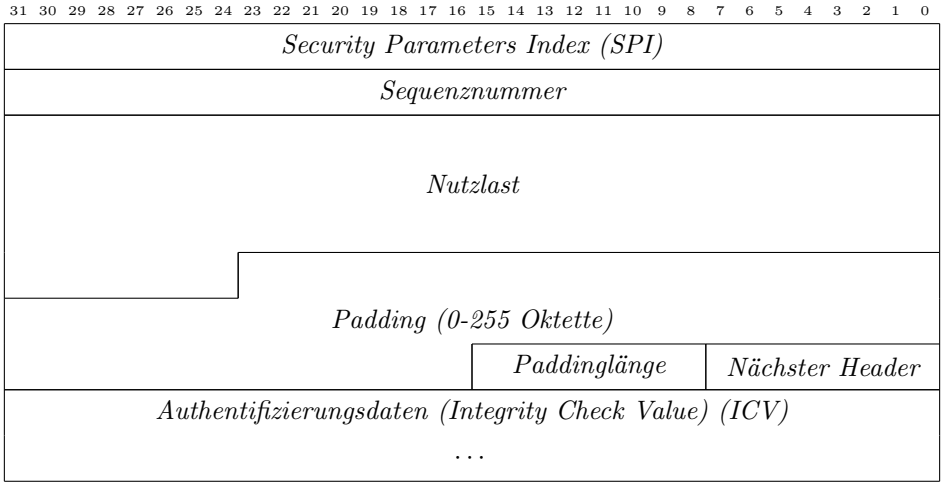

Abbildung 3: ESP

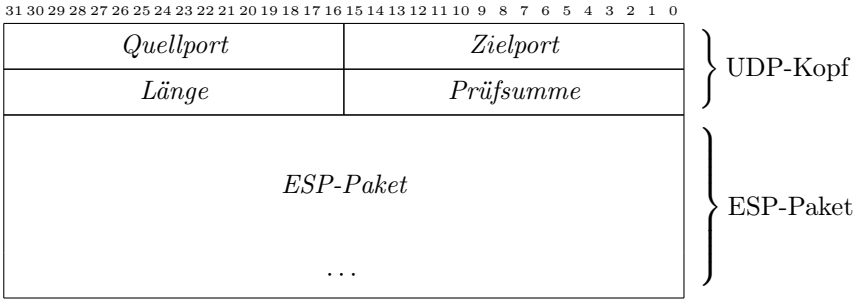

Abbildung 4: UDP Encapsulation

AH AH ist ein Protokoll, welches den Header des darunterliegenden IP-Pakets absichert, sowie die darin eingebetteten Daten. Im Header des darunterliegenden Pakets werden nur die statischen Felder abgesichert. AH bietet Integrität und Replay-Schutz. Es unterstützt keine Vertraulichkeit, da der Verkehr nur mittels eines HMAC abgesichert wird.[14]

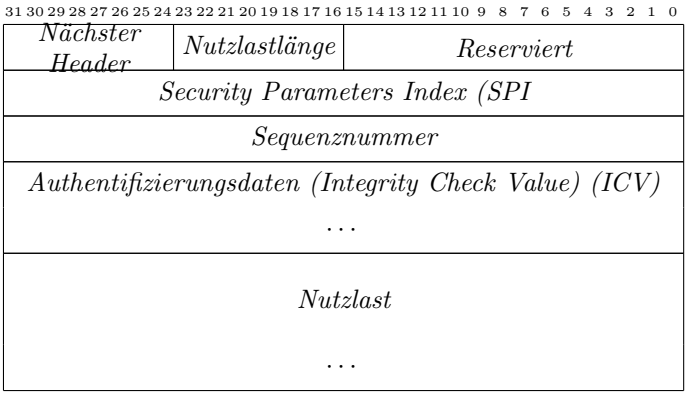

Abbildung 5: AH

[14] Ste05a.

2.3.4 MTU und MSS

Das Tunneln von Paketen verändert die Maximum Transmission Unit (MTU) des Trägers, da die Größe des darunterliegenden Datenträgers (IP, (UDP), ESP/AH) beachtet werden muss, sowie die Blockgröße des eingesetzten Verschlüsselungsalgorithmus und der eingesetzte Message Authentication Code (MAC). Gewöhnliche Computer nehmen in der Regel eine MTU von 1500 Byte an und berechnen aus dieser MTU die TCP-Maximum Segment Size (MSS) und geben diese beim Verbindungsaufbau mit Transmission Control Protocol (TCP) an. Da die daraus resultierenden Pakete jedoch dann zu groß sind, müssen diese fragmentiert werden. Dies wird mit einer ICMP-Nachricht bewerkstelligt, die an den Sender geschickt wird. Es gibt jedoch leider viele TCP-Implementierungen, die dann nicht fragmentieren und so keine Verbindung aufbauen können. Des weiteren gibt es viele Internet Service Provider (ISP), die ICMP-Nachrichten verwerfen. Dieses Problem kann umgangen werden, indem auf dem IPsec-Router der MSS-Wert von TCP-Nachrichten heruntergesetzt wird, sodass eine niedrigere MSS als die MTU des Tunnels zuzüglich des TCP-Overheads entsteht. Die Fragmentierung von IP-Paketen auf dem IPsec-Router sollte das eigentlich umgehen, aber es funktioniert in der Regel nicht. Dies kann auch aus Sicherheitsgründen so sein, da für die Überprüfung des Packets bezüglich der ausgehandelten SPs zwischen den Teilnehmern erst alle Fragmente gesammelt werden müssten, was bezüglich des Speicherbedarfs problematisch sein kann.[15]

2.3.5 Standardszenarien für IPsec

IPsec wird alltäglich für die Absicherung von Verkehr zwischen einem Router oder VPN-Server und Roadwarrior (RW) genutzt. Ein weiteres Szenario ist die Absicherung von Verkehr zwischen zwei Routern über einen Site-to-Site-Tunnel.

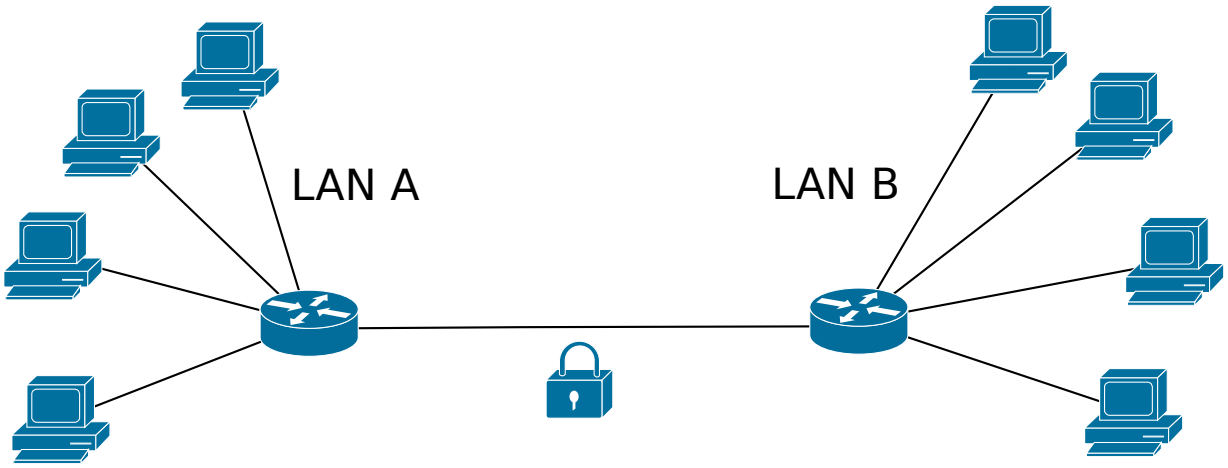

Abbildung 6: Site-to-Site-Szenario

2.3.6 IPsec unter Linux

Unter Linux gibt es zwei verschiedene IPsec-Stacks. Es existiert einerseits der KLIPS-Stack, der vom FreeS/WAN-Projekt entwickelt wurde und auf Route based IPsec basiert.

[15] Ste05b, Kapitel 3.4 Inbound Packet Processing.

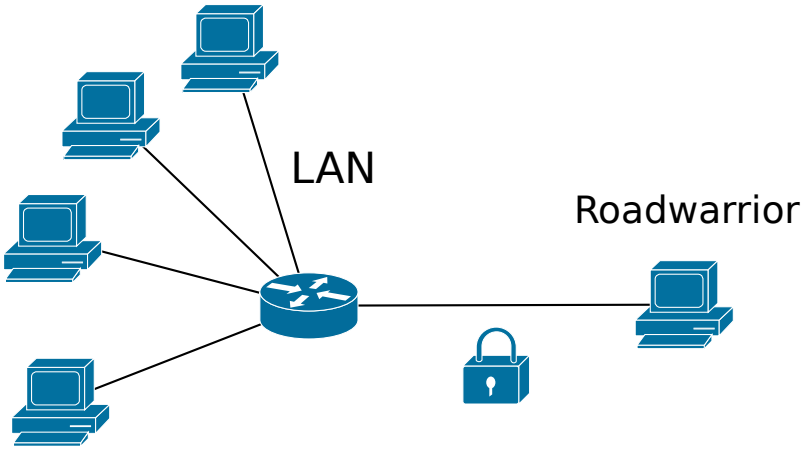

Abbildung 7: Roadwarrior-Szenario

Des weiteren gibt es den XFRM-Stack, welcher auf Policy based IPsec basiert, jedoch mithilfe eines Virtual Tunnel Interface (VTI) auch für Route based IPsec genutzt werden kann.

Beide Stacks sind konform zu RFC 2367[16].

Die Kommunikation mit dem Kernel bezüglich Netzwerkfunktionen geschieht über den Netlink-Sockel. Das ”strongSwan”-Projekt implementiert mit dem Dienst ”charon” einen ISAKMP/IKEv1 und IKEv2-Keying-Daemon auf Linux. Die notwendigen Funktionen für die Kommunikation mit dem Kernel sind in ”charon” im Plugin ”kernel-netlink” implementiert. Über Netlink werden auch die SAD und SPD verwaltet.

2.3.7 IPsec unter Windows

Microsoft hat im Laufe der Entwicklungsgeschichte von Windows drei verschiedene IPsec-Implementierungen implementiert, die teilweise parallel existierten.

In der Firewall von Windows existiert eine Implementierung, die Policy-based ist und daher flexibler ist und es erlaubt Firewall-Regeln zu schreiben, die abhängig davon sind, ob ein Paket mit IPsec geschützt war.

”strongSwan” implementiert mit ”charon-svc” einen ISAKMP/IKEv1 und IKEv2-Keying-Daemon auf Windows. Die Kommunikation mit dem Kernel ist dort in den Plugins ”kernel-wfp” und ”kernel-iph” implementiert.

Windows Agile VPN Client Seit Windows 7 existiert in Windows eine Implementierung von IPsec im Netzwerkmnanager, der den Config-Modus implementiert, sodass Windows als RW operieren kann. Diese Implementierung nutzt eine Route-based-Implementierung, wie in Unterunterabschnitt 2.3.8 erklärt. Diese Implementierung heißt offiziell ”Windows Agile VPN Client” und implementiert Point to Point Tunneling Protocol (PPTP), IPsec+Layer 2 Tunneling Protocol (L2TP), Secure Socket Tunneling Protocol (SSTP) und IKEv2. Sie existiert seit Windows 7.

Die Fähigkeiten dieser Implementierung lässt sich aus den Tabellen im Appendix unter Unterabschnitt 6.1 ablesen.

[16] DB98.

IKEEXT Windows wird mit einem Dienst namens "IKEEXT" ausgeliefert, welcher genutzt werden kann, um IKE_SAs und CHILD_SAs auszuhandeln. Standardmäßig ist dieser Dienst aktiv und lauscht auf User Datagram Protocol (UDP) Port 500 und 4500. Diese Implementierung unterstütz den Config-Modus nicht und ist daher nicht für die Benutzung als RW geeignet.[17]

WFP Windows Filtering Platform (WFP) ist die grundlegende Technologie im Netzwerkstack von Windows, mit der IPsec im Kernel, sowie die Firewall dort implementiert wurde. "charon" kann mit WFP mithilfe des "kernel-wfp"-Plugins kommunizieren und SAs und SPs in die jeweiligen Datenbanken einfügen.

WFP hat mehrere Probleme, unter anderem mit virtuellen IP-Adressen, UDP encapsulation, sowie mit mehreren Traffic Selectors im Transport-Modus. Des weiteren stellt WFP keine Statistiken über die einzelnen SAs bereit, was verhindert, dass volumenbasiertes Rekeying und das Auslesen von Nutzungsstatistiken der SAs implementiert werden kann.

Aufgrund der Problematik bezüglich virtueller IPs ist es nicht sinnvoll einen RW-Client auf Basis von WFP zu erstellen.[18][19]

IPH Windows bietet die IP Helper (IPH)-Funktionen an, die das Verwalten von IP-Adressen, Routen und Geräten erleichtert. "strongSwan" nutzt diese Funktionen im "kernel-iph"-Plugin, um ein Kernel-Interface für Windows mit diesen Funktionen zu implementieren.

2.3.8 Route Based

Route based heißt, dass Pakete, die über das VPN gesendet werden sollen, in eine virtuelle Netzwerkschnittstelle geroutet werden. Da gewöhnliches Routing auf Basis der Zieladresse geschieht, unterstützt solche eine solche Implementierung in der Regel keine SPs, die nur bestimmte Protokolle oder Ports schützen oder wenn das der Fall ist, so ist die Kommunikation mit den anderen Ports oder Protokollen nicht mehr möglich, da jegliche Pakete, die in die Schnittstelle gelangen und von den SPs nicht zugelassen sind, verworfen werden.

Implementierungen von IPsec auf Endgeräten sind in der Regel Route based, gleichzeitig werden die ausgehandelten SPs jedoch weiterhin erzwungen. Route based VPNs werden für gewöhnlich dann genutzt, wenn dynamisches Routen genutzt wird, wie zum Beispiel Border Gateway Protocol (BGP), Intermediate System To Intermediate System (ISIS) oder Open Shortest Path First (OSPF), da beim Verändern der Routen die SPs nicht verändert und damit keine neuen CHILD_SAs ausgehandelt werden müssen. Ein weiterer Vorteil ist, dass Neulinge es leichter haben den Paketfluss zu verfolgen.

Um zu verhindern, dass der IKE-Verkehr über das VPN geleitet wird, wird eine Route für die IP-Adresse des Peers in den Routing Table installiert, die den Verkehr am VPN vorbei leitet. Aus diesem Grund ist es auch nicht möglich mit Route based IPsec mit der IP des Peers unter der Nutzung von IPsec zu kommunizieren.

Eine Möglichkeit Pakete, die nicht von den Policies geschützt werden, nicht ins VPN zu leiten und damit die Kommunikation mit ihnen zu ermöglichen ist PBR anzuwenden und ebendiese Pakete speziell zu markieren und eine andere Routing-Entscheidung zu treffen.

[17] 16e.
[18] TM14.
[19] Mar14.

2.3.9 Policy Based

Policy based bedeutet, dass die IPsec-SPs, die ausgehandelt wurden, direkt genutzt werden welche Pakete geschützt werden. Eine solche Implementierung ermöglicht es, IPsec zur Absicherung von sehr genau spezifiziertem Verkehr zu nutzen, wie zum Beispiel nur ICMP-Pakete mit Typ 0, Code 0 (Echo Reply), sowie Typ 8 und Code 0 (Echo Request). Policy based IPsec ermöglicht es IPsec direkt in den Stack zu integrieren, ohne notwendigerweise die Routen zu verändern. Um sicherzustellen, dass der IKE-Verkehr des IKE-Daemons nicht mittels einer IPsec-SA geschützt wird, was unter Umständen die Kommunikation zwischen den IKE-Daemons verschiedener Systeme verhindern kann, werden bestimmte Optionen auf dem jeweiligen Netzwerksockel gesetzt, die Pakete, die von diesem Sockel kommen, von den SPs und SAs nicht bearbeitet werden.

2.3.10 TUN und TAP-Geräte

TUN- und TAP-Geräte sind virtuelle Netzwerkadapter, die von lokalen Anwendungen genutzt werden können um vom Kernel IP-Pakete oder Ethernet-Rahmen zu empfangen. Pakete und Rahmen, die der Kernel über ein TUN- oder TAP-Gerät erhält werden gleich behandelt als wären sie über eine physikalische Schnittstelle empfangen worden. Sie werden auch gleich verwaltet, so können ihnen zum Beispiel IP-Adressen zugewiesen und Routen über so eine virtuelle Schnittstelle angelegt werden.

TUN-Geräte TUN-Geräte sind virtuelle Netzwerkschnittstellen, die auf der Netzwerkschicht arbeiten. Das heißt, dass sie IP-Pakete senden und empfangen ohne einen darunterliegenden Ethernet-Rahmen. Das heißt im Umkehrschluss das über ein TUN-Gerät keine Kollisionsdomäne erreichbar ist, ARP nicht benutzbar ist und das ”next hop”-Feld einer Route in der Routing-Tabelle keine Bedeutung hat, wenn die Route über ein TUN-Gerät führt.

TAP-Geräte TAP-Geräte verhalten sich wie physikalische Ethernet-Schnittstellen. Das heißt, dass prinzipiell über sie ARP und Ethernet gesprochen werden können und eine Kollisionsdomäne erreichbar ist. TAP-Geräte können genutzt werden um eine Brücke zwischen einem lokalen Netzwerk und einem entfernten Netzwerk über einen VPN-Dienst mithilfe eines virtuellen Brückengeräts zu bauen.

3 Basis

Als Basis der Arbeit wurde der quelloffene VPN-Dienst strongSwan gewählt, welcher von der Hochschule für Technik Rapperswill entwickelt wird. Um die Pakete vom Kernelspace in Empfang nehmen zu können, wird der quelloffene TAP-Gerätetreiber von OpenVPN Technologies, Inc. genutzt. Dieser stellt funktionierende und nutzbare TUN- und TAP-Geräte bereit, mit denen gearbeitet werden kann. Für die Verarbeitung der Pakete wird die in strongSwan integrierte Bibliothek libipsec, das Plugin kernel-libipsec, sowie bestehende Teile von libstrongswan aus strongSwan genutzt.

3.1 Bestehende Implementierungen für Windows

In Abschnitt 6 ist eine Auflistung aller mir bekannten IPsec-Clients für Windows, inklusive Matrizen der jeweils unterstützten Algorithmen und Verfahren in den ISAKMP- und IKE-Standards.

3.2 strongSwan

strongSwan implementiert einen beträchtlichen Teil der Funktionalität der RFCs für IKEv2 und ISAKMP/IKEv1[20]. Die Software ist multithreaded und in objektorientiertem C geschrieben. Die Grundlage wurde von Martin Willi und Jan Hutter in ihrer Diplomarbeit im Jahr 2005 gelegt[21]. "strongSwan" wird unter anderem von der secunet AG in den SINA-Boxen für Hochsichere Anwendungen, Alcatel-Lucent, Astaro und der US-Regierung genutzt.[22]

3.2.1 charon

"charon" ist der Daemon, der in der Diplomarbeit[23] entwickelt wurde. Er arbeitet parallel und wurde in objektorientiertem C geschrieben, so wie die Software Xine. Intern werden verschiedene Bibliotheken genutzt, angefangen von "libstrongswan" bis zu "libcharon". Es existieren verschiedene Versionen von "charon" für verschiedene Zwecke, zum Beispiel "charon-svc". "charon-svc" ist eine Version von "charon", die als Dienst auf Windows genutzt werden kann.

Jegliche Funktionalität von strongSwan ist in Form einer Bibliothek untergebracht und in objektorientiertem C geschrieben. Das ermöglicht es die Namensräume in strongSwan relativ frei zu halten und Objekte gleichen Typs gleich zu behandeln, sowie die Duplikation von kurzem Code zu vermeiden.

Ein Beispiel dafür ist libstrongswan, welche unter anderem allgemeine Datenstrukturen wie Linked Lists, Arrays und Hashtables, sowie Code für TUN-Geräte und Multithreading implementiert.

Kernel-Interfaces "charon" benötigt für die Verwaltung von IP-Adressen, Routen, TUN-Schnittstellen, Firewallregeln und der SPD und der SAD Zugriff auf den Kernel. Die dafür nötigen Funktionen werden in Plugins für "libcharon" implementiert, die die dafür benötigten Funktionen in einem Standardisierten Interface implementieren. Die momentan existierenden Plugins, die Kernel-Schnittstellen implementieren sind "kernel-pfroute", "kernel-netlink", "kernel-wfp" und "kernel-iph".[24]

3.2.2 Userspace Processing

Userspace Processing wurde in "charon" von Guiliano Grassi und Ralf Sager für Android implementiert, um die Entwicklung der strongSwan-App für Android zu ermöglichen. Auf Android ist Kernelspace Processing von IPsec-Paketen nicht möglich, da die Privilegien der Anwendung das nicht erlauben.

Um Userspace Processing von IPsec-Paketen zu implementieren benötigt die Anwendung, die die Pakete verarbeiten soll, Zugriff auf die Pakete und muss, damit die darin enthaltenen

[20] 16f.
[21] JM05.
[22] And11, Folie 8.
[23] JM05.
[24] 16g.

Pakete an den Zielort weitergeleitet werden können, eine Möglichkeit haben sie an den Kernel zum Routing zu übergeben. In der Regel wird dies mit TUN- oder TAP-Geräten bewerkstelligt, die einen virtuellen Netzwerkadapter darstellen. Am einen Ende hängt der Kernel mit seiner Routing-Engine und den Firewallregeln, am anderen Ende hängt die Anwendung, die Pakete auf einen File Descriptor (FD) oder ein Handle schreibt und davon liest.

Pakete, die in den Adapter gesendet werden werden von der Anwendung empfangen und Pakete, die in das Handle geschrieben werden, tauchen auf dem Adapter auf Seiten des Betriebssystems auf.

OpenVPN TAP-Treiber Der OpenVPN-Tap-Treiber ist unter den Namen "TAP-Windows" bekannt und wurde von OpenVPN Technologies, Inc. für den Einsatz mit OpenVPN entwickelt. Der Treiber stellt unter Windows ein Ethernet-Gerät dar, inklusive Kollisionsdomäne. Dies ist für dein Einsatz mit IPsec etwas unbeholfen, da IPsec IP-Pakete überträgt und keine Ethernet-Rahmen. Der Treiber unterstützt, wie OpenVPN, das Tunneln von Ethernet-Frames jedoch (TAP-Modus). Der Treiber bewerkstelligt das, indem er den Ethernet-Rahmen eines eingehenden Pakets vom Host verwirft und nur das IP-Paket weiterleitet. Vice Versa ergänzt er IP-Pakete, die er von der Software erhält um einen Ethernet-Rahmen. Um über den Adapter kommunizieren zu können stellt der Treiber einen virtuellen Router bereit, dessen IP-Adresse, wie die lokale und die entfernte IP-Adresse(n) konfigurierbar ist. Der Treiber reagiert nicht auf jede ARP-Anfrage, sondern nur auf solche, bei denen die Quelladresse im ARP-Request der lokalen konfigurierten gleicht, sowie die erfragte Adresse der konfigurierten entfernten Adresse (oder einer der Entfernten, falls ein gesamtes Netzwerk als entfernte IP konfiguriert wurde) gleicht. Die MAC-Adresse des virtuellen Routers hängt von der GUID des Adapters ab, welche zufällig ist und bei der Erstellung des Adapters initialisiert wird. Die MAC-Adresse kann auch über die Registry verändert werden.

Um mit dem virtuellen Adapter kommunizieren zu können, wird ein Handle mit dem Gerät geöffnet, über das dann asynchron gelesen und geschrieben werden kann.

Betriebsmodi Der TAP-Treiber beherrscht drei verschiedene Betriebsmodi für verschiedene Zwecke. Laut dem Quellcode des Treibers ist der Point-To-Point-Modus veraltet.

TUN-Modus Im TUN-Modus schneidet der Treiber den Ethernet-Rahmen um das IP-Paket ab und reicht das Paket über das Handle an die Software weiter. Pakete, die an den Treiber gegeben werden, werden um einen Ethernet-Rahmen ergänzt, bei dem die Zieladresse die MAC-Adresse des virtuellen Adapters ist und die Quelladresse die MAC-Adresse des virtuellen Routers. In diesem Modus existiert eine lokale IP-Adresse, ein virtueller Router und ein entferntes Netzwerk.

TAP-Modus Im TAP-Modus kopiert der Treiber jeden Ethernet-Paket zwischen Anwendung und Gerät.

Point-To-Point-Modus Im Point-To-Point-Modus existiert eine lokale IP und eine entfernte IP. Empfangene Ethernet-Frames werden entfernt und nur das enthaltene IP-Paket weitergereicht.

libipsec libipsec ist eine Bibliothek, die IPsec im Tunnelmodus implementiert. Sie ist Teil des strongSwan-Quellcodes. Sie wird für die Verarbeitung von IP-Paketen in der Regel auf Systemen genutzt, die keine (funktionierenden) IPsec-Implementierung enthalten.

libipsec besteht aus zwei Komponenten: Einerseits die Bibliothek, die die SAD und die SPD implementiert, sowie die Verarbeitung (libipsec) und ein Plugin, welches ein Kernel-Interface für das Verwalten der SAD und SPD emuliert (kernel-libipsec).

libipsec empfängt und sendet Pakete über die gleichen Sockets, den der restliche Teil des Daemons benutzt, um mit anderen IKE-Peers zu kommunizieren. Das sind Sockets, die auf UDP mit Port 500 und 4500 gebunden sind; die IKE-Standardports. libipsec unterstützt die Verarbeitung von UDP-Encapsulation-Paketen, sowie von normalen ESP-Paketen, da sie sich strukturell nur gering unterscheiden. Jedoch wird aufgrund der Tatsache, dass mit den gegebenen Sockets kommuniziert wird, UDP-Encapsulation forciert. Um ESP-Pakete verschicken zu können, müsste ein Socket geöffnet werden der das verschicken von solchen Paketen erlauben würde. Das ist jedoch auf der Plattform, für die die Bibliothek ursprünglich geschaffen wurde (Android) nicht erlaubt, da die Privilegien der Anwendung dort sehr beschränkt sind.

Prinzipiell wäre es jedoch möglich dies zu implementieren. Dafür müsste ein raw-Socket geöffnet werden, über den rein IP gesprochen wird.

libstrongswan libstrongswan ist eine interne Bibliothek von strongSwan, die von den verschiedenen Versionen von ”charon” geladen wird, um diverse Funktionalität zu erhalten, wie Linked-Lists, Hashtables, Arrays, sowie die Funktionen um TUN-Geräte zu erstellen und zu öffnen.

Der Code für das lesen und schreiben von Paketen existiert bereits für Linux, Mac OSX und FreeBSD. Die Operationen basieren dabei auf File Descriptors, welche mit poll() multiplext nach Daten abgefragt werden.

4 Vorgehensweise

Zuerst wurden alle nötigen Änderungen ausfindig gemacht. Dies beinhaltete die Migrierung von Code von UNIX/Linux-spezifischem IO-Multiplexing mittels poll() zu anderen Verfahren.

Des weiteren mussten fehlende Features gefunden werden. Einen Hinweis darauf ließ sich bereits auf den Wiki-Seiten über die Plugins für Windows finden. Dieser ergab, dass die Verwaltung von virtuellen IPs unter Windows momentan nicht unterstützt wurde.

Weitere Punkte wurden nach dem ersten Kompilierungsversuch, sowie der Beobachtung des Verhaltens während dem Tunnelaufbau herausgefunden.

Die Unterstützung für virtuelle IP-Adressen wurden aus dem Zweig ”windows-vip” vom strongSwan Git-Repository bezogen und angepasst. Der Großteil des Quellcodes stammt also von Martin Brunner, der diesen Code geschrieben hat. Der Code musste noch um die Beachtung eines Konfigurationswertes ergänzt werden.

Die Installation der Routen in kernel-libipsec wurde angepasst, sodass es mit dem TAP-Treiber funktioniert.

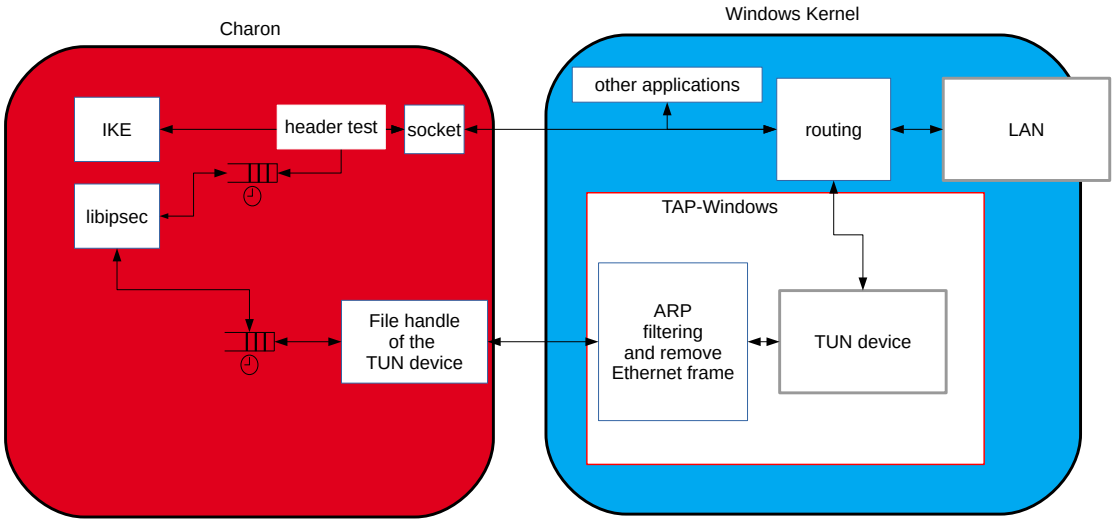

Abbildung 8: Datenflussdiagramm

4.1 Portierung von libipsec auf Windows

Die Portierung von libipsec auf Windows, sodass sie dort lauffähig ist, ist das Ziel der Bachelorarbeit. Die zu portierenden Teile des Programmcodes betreffen nur die Codesegmente, die Plattformspezifische Application Programming Interfaces (APIs) oder Application Binary Engines (ABIs) verwenden, also primär alles um Dateiein- und Ausgabe, sowie das verwalten von TUN-Geräten. Unter Unix und Linux werden hier FDs verwendet. Unter Windows werden stattdessen Handles genutzt, welche einen anderen Dateityp darstellen. Des weiteren unterscheidet sich die Methode zum Multiplexen von Lese-Aufrufen einer Liste von Dateien stark. Unter Unix und Linux wird hierfür poll() genutzt, unter Windows geschieht das jedoch Event-basiert mit WaitForMultipleObjects(). Explizite Beispiele hierfür sind im Abschnitt über die Portierung von libipsec sichtbar. Bei der Portierung wurde für das verwalten von Speicherabschnitten primär alloca() genutzt, statt malloc() und seine Unterarten. Der Unterschied hierbei ist die Speicherdauer. Mit alloca() allokierter Speicher ist nur bis zum Verlassen der Funktion gültig und wird automatisch freigegeben. Mit malloc() allokierter Speicher muss manuell freigegeben werden. alloca() ist ein Feature, welches nicht standardisiert ist. Daher lässt sich strongSwan nicht mit allen C-Bibliotheken übersetzen. Auf der Wiki-Seite über den Windows-Port wird erwähnt, dass nur die Kompilierung mittels MinGW-W64 unterstützt wird.

4.2 Bestehende Implementierung

Die bestehende Implementierung umfasst die eigentliche Bibliothek, eine Implementierung eines Kernel-Interface zwischen libipsec und ”charon”, sowie Code in libstrongswan um TUN-Geräte zu öffnen. Martin Willi hat 2014 bereits die gesamte Portierungsarbeit für version 5.2.0 von strongSwan gemacht. Der Port unterstützt Windows 7 / server 2008 R2 und neuer.

4.3 Unterstützte Kryptographie

Die unterstützte Kryptographie ist notwendigerweise von den deklarierten Identifikatoren für IKE beschränkt. strongSwan unterstützt mehr Verschlüsselungsalgorithmen als in den RFCs

deklariert wurde. Aus diesem Grund nutzt strongSwan Identifikatoren teilweise aus dem privaten Bereich, wenn die Identifikatoren für den eingesetzten Algorithmus nicht standardisiert sind.

strongSwan unterstützt eine große Anzahl von Algorithmen im Vergleich zu anderen Implementierungen, wie in den Tabellen in Abschnitt 6 dagelegt wird. Wenn libipsec genutzt wird, so können alle Algorithmen, die im Userspace implementiert sind, für die Absicherung von Verkehr genutzt werden.

4.4 Portierung

4.4.1 header

Die Bibliotheken und Plugins um libipsec nutzen diverse Datenstrukturen und Konstanten, die in C-Header-Dateien von Linux definiert sind. Diese sind unter Windows nicht verfügbar. Aus diesem Grund wurde eine Kopie der relevanten Definitionen in den Quellcode kopiert und fehlende Teile ergänz. Spezifisch wurde aus dem Quellcode von GLIBC die Definition eines IP-Headers kopiert und die fehlenden Protokollkonstanten per Hand ergänzt.

Folgend ist die Headerdatei inklusive der Lizenzheader.

Code 1: Header für libipsec

```
1  /* Copyright (C) 1991-2016 Free Software Foundation, Inc.
2     This file is part of the GNU C Library.
3
4     The GNU C Library is free software; you can redistribute it and/or
5     modify it under the terms of the GNU Lesser General Public
6     License as published by the Free Software Foundation; either
7     version 2.1 of the License, or (at your option) any later version.
8
9     The GNU C Library is distributed in the hope that it will be useful,
10    but WITHOUT ANY WARRANTY; without even the implied warranty of
11    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
12    Lesser General Public License for more details.
13
14    You should have received a copy of the GNU Lesser General Public
15    License along with the GNU C Library; if not, see
16    <http://www.gnu.org/licenses/>.  */
17
18 */
19
20 /*
21  * Copyright (c) 1982, 1986, 1993
22  *      The Regents of the University of California.  All rights reserved.
23  *
24  * Redistribution and use in source and binary forms, with or without
25  * modification, are permitted provided that the following conditions
26  * are met:
27  * 1. Redistributions of source code must retain the above copyright
28  *    notice, this list of conditions and the following disclaimer.
29  * 2. Redistributions in binary form must reproduce the above copyright
30  *    notice, this list of conditions and the following disclaimer in the
31  *    documentation and/or other materials provided with the distribution.
32  * 4. Neither the name of the University nor the names of its contributors
33  *    may be used to endorse or promote products derived from this software
```

```
34  *     without specific prior written permission.
35  *
36  * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
37  * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
38  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
39  * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
40  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
41  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
42  * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
43  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
44  * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
45  * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
46  * SUCH DAMAGE.
47  *
48  *      @(#)ip.h        8.1 (Berkeley) 6/10/93
49  */
50
51 /*
52  * Details about licensing:
53  * The definition of the IP header is from the headers of GLIBC, that come with
        Arch Linux
54  * It is subject to the license headers of GLIBC and Berkeley
55  * The IP protocol identifier constants are manually determined from
        /etc/protocols
56  * and hand written out.  They are subject to the GPLv2.
57  */
58
59 #ifndef WINDOWS_32_PROTOCOL_HEADERS
60 #define WINDOWS_32_PROTOCOL_HEADERS
61
62 /*
63  * Structure of an internet header, naked of options.
64  */
65 struct ip
66   {
67 #if __BYTE_ORDER == __LITTLE_ENDIAN
68     unsigned int ip_hl:4;               /* header length */
69     unsigned int ip_v:4;                /* version */
70 #endif
71     uint8_t ip_tos;                     /* type of service */
72     u_short ip_len;                     /* total length */
73     u_short ip_id;                      /* identification */
74     u_short ip_off;                     /* fragment offset field */
75 #define IP_RF 0x8000                    /* reserved fragment flag */
76 #define IP_DF 0x4000                    /* dont fragment flag */
77 #define IP_MF 0x2000                    /* more fragments flag */
78 #define IP_OFFMASK 0x1fff               /* mask for fragmenting bits */
79     uint8_t ip_ttl;                     /* time to live */
80     uint8_t ip_p;                       /* protocol */
81     u_short ip_sum;                     /* checksum */
82     struct in_addr ip_src, ip_dst;      /* source and dest address */
83   };
84
85 #
86 #define IPPROTO_IPIP 4
```

```
87 #define IPPROTO_IPv6 41
88 #define IPPROTO_NONE 59
89
90 #endif /* WINDOWS_32_PROTOCOL_HEADERS */
```

4.4.2 libipsec

libipsec implementiert die Verarbeitung von Paketen, das Erzwingen der SPs, das Verwalten der SPs, SAs, Routen und der TUN-Geräte. Die hierbei relevanten Dateien sind unter /src/libipsec/ zu finden.

Standardmäßig installiert strongSwan die IP-Adressen die per Config-Mode oder Configuration Payload (CP) empfangen wird auf dem ausgehenden Interface (Linux, Kernelspace processing), was für die Kommunikation mit dem Peer genutzt wird, oder auf dem Loopback-Adapter (Windows). Um die IP-Adresse für TUN-Geräte korrekt zu setzen, nutzt libipsec den Parameter "charon.install_virtual_ip_on" von strongswan.conf, der während der Initialisierung des Plugins gesetzt wird.

4.4.3 kernel-libipsec

Die hier zu portierenden Bestandteile waren ausschließlich Codeteile, die sich mit der Dateiein- und ausgabe beschäftigten.

Die primäre Aufgabe hier war die Installation der Routen in "kernel_libipsec_ipsec.c" für Windows anzupassen, da hier ein Gateway verwendet wird auf einem TAP-Gerät statt ein echtes TUN-Gerät, sowie die Anpassung des Codes für die Ein- und Ausgabe und einige Methoden in "kernel_libipsec_router.c", da das Multiplexen der Handles, sowie die Benachrichtigung von anderen Threads auf Windows anders abläuft als auf Linux und UNIX.

Für das Multiplexen der Eingabe stehen unter Windows zwei Verfahren zur Verfügung:

- WaitForMultipleObjects
- IOCompletionPorts

WaitForMultipleObjects[25] funktioniert mit einem Array aus Handles. In der Struktur des Handles ist das event-Attribut auf ein einzgartiges Event gesetzt, welches genutzt wird um das Handle zu finden, dessen Lesevorgang abgeschlossen oder abgebrochen wurde. Nach dem Kopieren des Handles in das Array und dem Setzen des Events wird ein asynchroner Lesevorgang gestartet. Wenn er sofort beendet wurde, wird das entsprechende Event gesetzt. Dadurch beendet der Aufruf von WaitForMultipleObjects() nach dem Aufruf direkt, falls ein Lesevorgang schon zuvor erfolgreich war und der Programmcode wird etwas kürzer.

IOCompletionPorts[26] funktionieren, indem man einen IOCompletionPort mit CreateIoCompletionPort() anlegt und Handles mit ihm asoziiert. Bei der Asoziierung wird ein einzigartiger Schlüssel übergeben, der bei der Signalisierung wieder zurückgegeben wird um das Handle identifizieren zu können. Der CompletionPort kann erst nach dem Schließen der asoziierten Handles geschlossen werden. Mit der Funktion GetQueuedCompletionStatus() kann der ausführende Thread dann auf abgeschlossene I/O-Operationen warten. Die Nachteile dieser Methode sind,

[25] 16i.
[26] 16b.

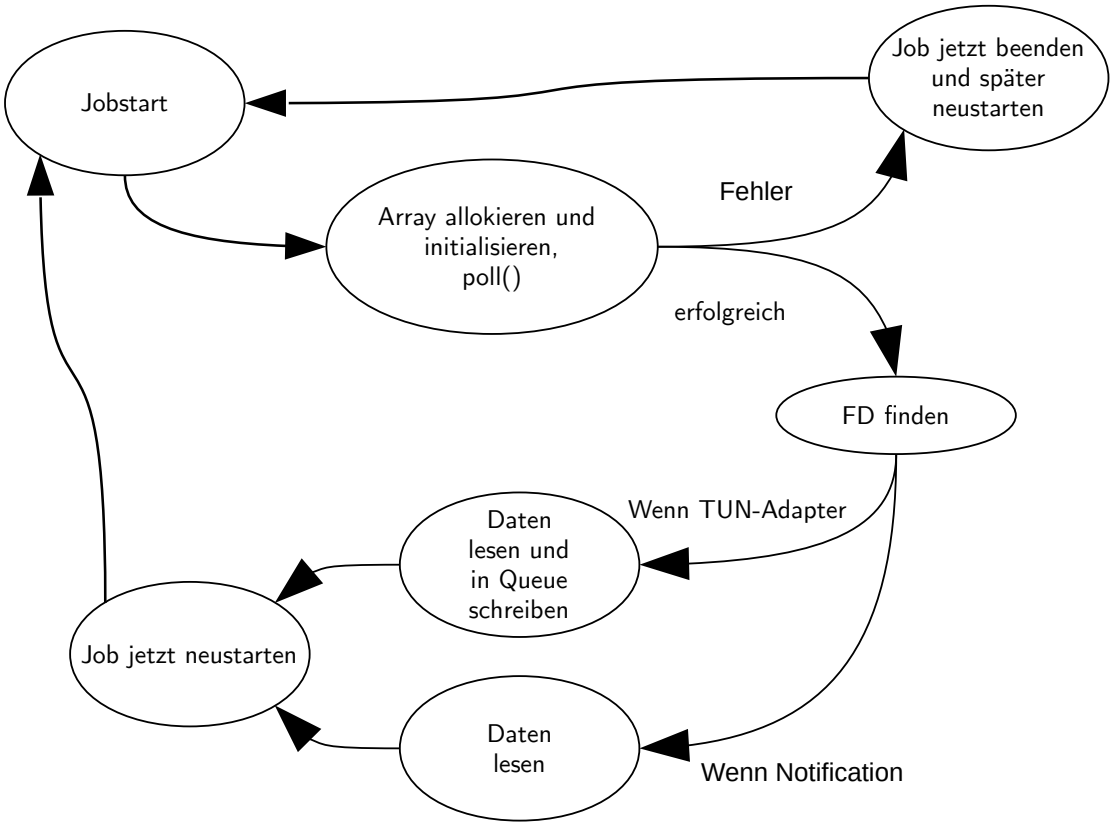

Abbildung 9: Zustände in handle_plain() mittels poll()

dass jegliche Operationen auf den Handles eine Benachrichtigung an GetQueuedCompletionStatus() generieren, obwohl Schreib-Vorgänge nicht von Interesse sind.

Des weiteren ähnelt die Nutzung von WaitForMultipleObjects() deren von poll() insoweit, dass die Handles/FDs auch nach der Benutzung mit der Funktion weiterhin einzeln genutzt werden können.

Da es relativ einfach ist WaitForMultipleObjects() zu nutzen, wurde diese Methode für die Implementierung von handle_plain() genutzt.

Die Zustände der Originalimplementierung sind in Abbildung 9 dargestellt.

Wie aus dem Diagramm hervorgeht, wird im Prinzip nur ein Array mit Strukturen des Typs ”pollfd” erstellt, dann mit einem Notification-FD und den FDs der TUN-Geräte befüllt und als gewünschte Events ”POLLIN” gesetzt. Das heißt, dass der Aufruf von poll() ohne Fehler beendet wird, wenn Daten auf dem FDs zur Verfügung stehen. Danach wird poll() auf die ”pollfd”-Datenstruktur ausgeführt und analysiert, auf welchen FDs Daten zur Verfügung stehen. Dieser Code ist relativ kurz im Vergleich zum Analogon mit WaitFor*Objects() Funktionen unter Windows. Dies geht aus den vergleichbaren Implementierungsmöglichkeiten, die sich aus den WaitFor*Objects()-Funktionen ergeben. Dies wird im Diagramm Abbildung 10 und Abbildung 11 hervor. Diese Diagramme haben weitaus mehr Zustände als Abbildung 9.

Der Unterschied zwischen der Implementierung von Abbildung 10 und Abbildung 11 ist, dass Abbildung 11 effektiver und schneller ist, da die Puffer nach jedem Lesevorgang beibehalten werden und IO-Operationen weiterlaufen können. In Abbildung 10 werden die Puffer und Operationen immer freigegeben und gestoppt. Das ist unnötig, bringt die Implementierung

jedoch näher an das Verhalten von Abbildung 9. Es wurde das Verfahren aus Abbildung 11 implementiert, da es schneller und effektiver ist.

In handle_plain werden zwei verschiedene Arrays benutzt.

bundle_array Ein Array aus Strukturen vom Typ "handle_overlapped_buffer_t".

event_array Ein Array aus Events (HANDLE).

Strukturen vom Typ "handle_overlapped_buffer_t" beinhalten jeweils ein Objekt vom Typ HANDLE, ein Objekt vom Typ *OVERLAPPED und ein Objekt vom Typ chunk_t. Das Handle gehört zu einem TUN-Handle, das OVERLAPPED-Objekt wird direkt für asynchrones IO benötigt und das Objekt vom Typ chunk_t beinhaltet den Pufferspeicher und dessen Länge.

Das "event_array" Array zum Multiplexen mittels WaitForMultiplEvents() genutzt und beinhaltet die gleichen Events, wie die Strukturen des Typs OVERLAPPED. Bei der Ausführung von WaitForMultipleObjects() auf das Array wird darauf gewartet, dass eine vorher gestartete Leseoperation auf einem Handle fertig gestellt wird. Das signalisiert ein Event im Array und dessen Position im Array verrät welche Position im bundle_array einen nun gefüllten Puffer hat. Die Position signalisiert hierbei nur die niedrigste Position im Array, deren Event signalisiert wurde. Ein im Array höher gelegenes Event könnte auch signalisiert sein. Der Puffer kann nun gelesen werden und an die Queue zu libipsec gehängt werden. Nach dem Lesen wird die OVERLAPPED-Struktur und das Event zurückgesetzt, sowie der Puffer mit Nullen initialisiert und die Länge zurückgesetzt. Danach wird ein neuer Lesevorgang auf dem oder den Handles gestartet, deren Leseoperationen fertig sind.

Der Code in auf Seite 52 zeigt die Implementierung von handle_plain() auf Windows. Die Funktionen zum Starten von asynchronen Leseoperationen auf Handles wurden in start_read() abgekapselt. Das Übersetzen von Fehlercodes in menschenlesbare Texte wurde in format_error() abgekapselt. Bei Fehlern beim Lesen startet sich der Job mit JOB_REQUEUE_FAIR neu, sodass der Job als letzter vom Job-Manager in "charon" neugestartet wird, wenn hoffentlich der Fehler nicht erneut auftritt.

libipsec setzt für die von ihr installierten Routen standardmäßig das "Gateway" oder "Next Hop"Feld nicht. Der Grund dafür ist, dass libipsec bisher nur auf Betriebssytemen genutzt wurde, die echte Layer-3-Geräte implementieren und daher keine Kollisionsdomäne über das Gerät erreichbar ist. Daher ergibt es keinen Sinn das "next hop"-Feld zu setzen. Auf Windows ist dies jedoch, wie zuvor erläutert, nicht der Fall, da der TAP-Windows-Treiber TUN-Geräte als Ethernet-Gerät emuliert.

Code 2: Patch für die Routen-Installation von libipsec

```
1 #ifdef __linux__
2 #elif defined(WIN32)
3         /* TODO: Complete */
4         /* Set out special gateway */
5         /* We also need to add a route to 169.254.0.0/16 via all our tun devices
              */
6         host_t *gw = host_create_from_string("169.254.128.128", 0);
7         route->gateway = gw;
8 #else
9         /* on Linux we cant't install a gateway */
```

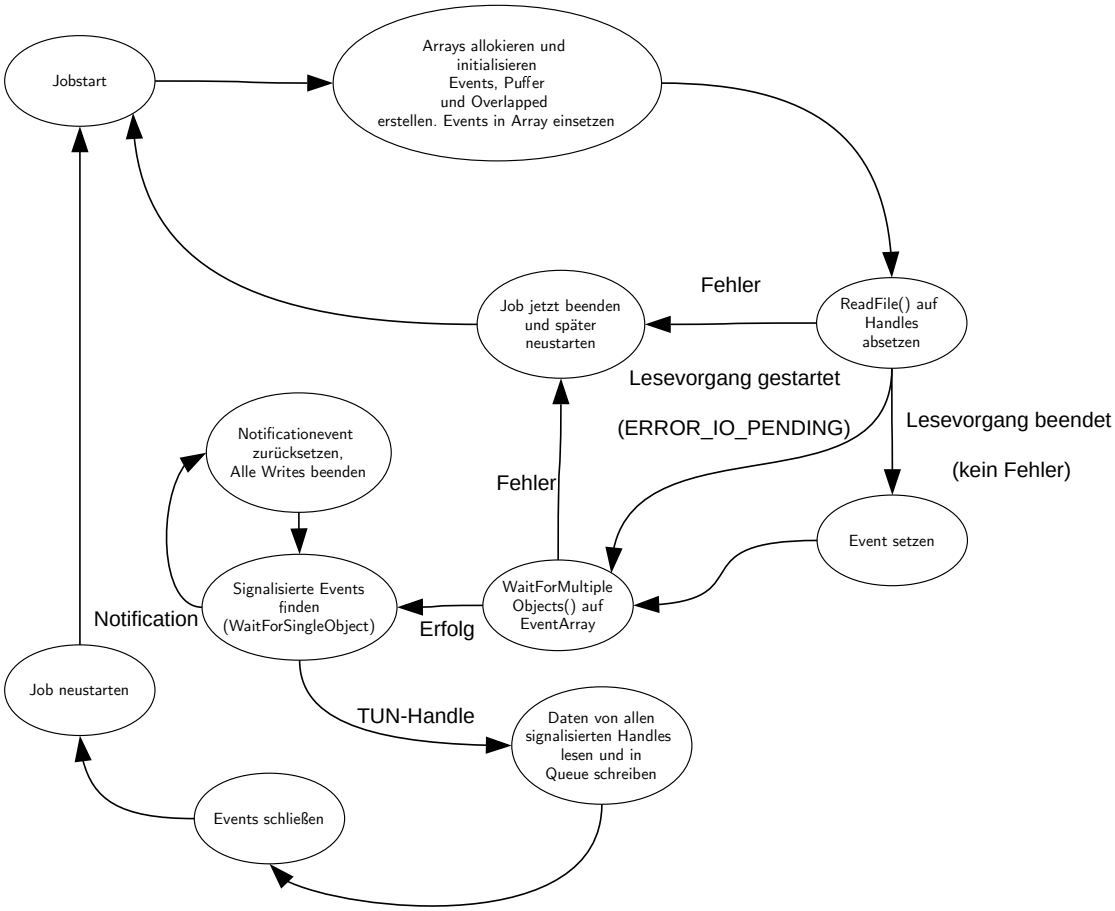

Abbildung 10: Zustände in handle_plain() mittels WaitForMultipleObjects(), Variante 1

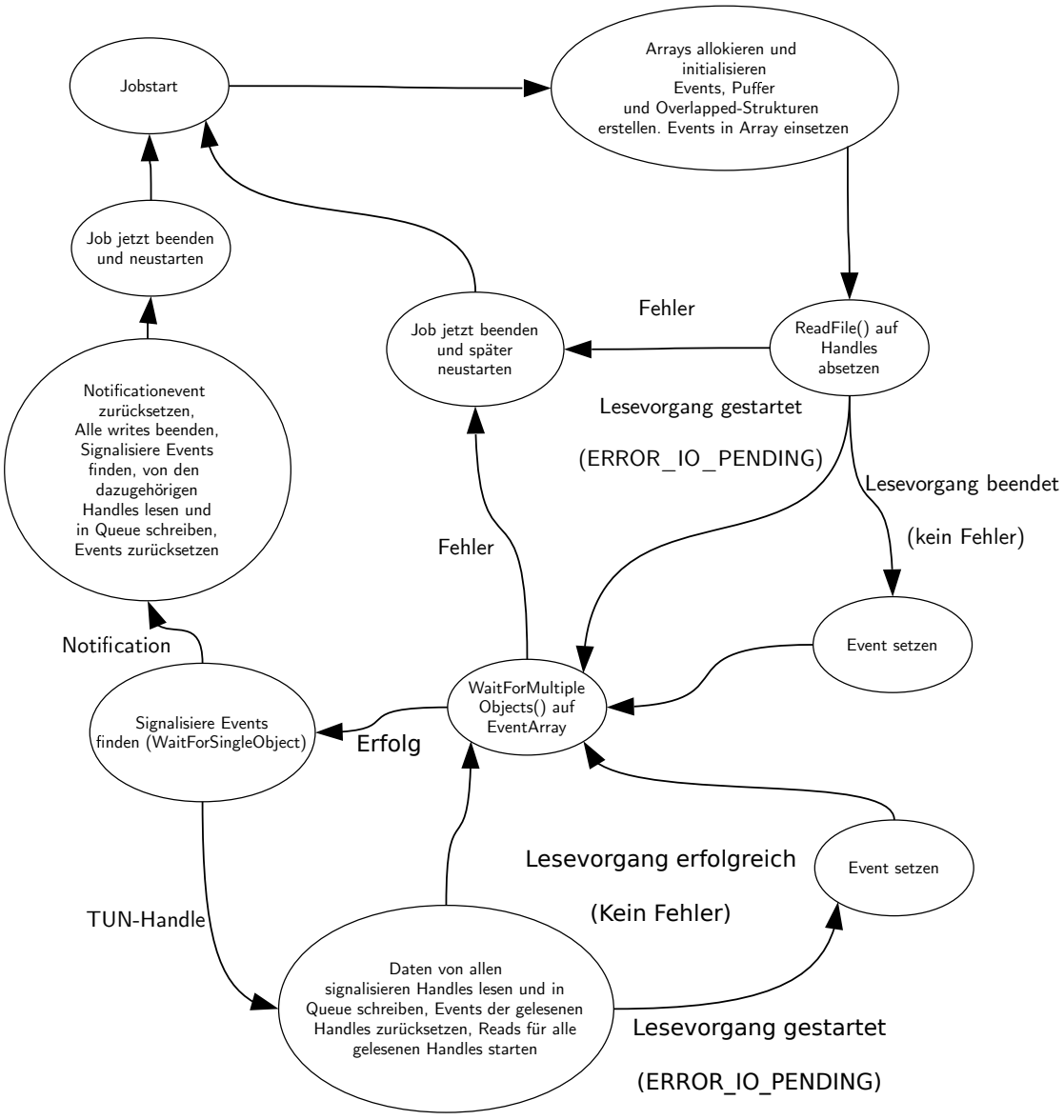

Abbildung 11: Zustände in handle_plain() mittels WaitForMultipleObjects(), Variante 2

```
10          route->gateway = charon->kernel->get_nexthop(charon->kernel, dst, -1,
                src);
11  #endif
```

Die Implementierung von handle_plain() auf Windows unterstütz, wie die Implementierung auf Linux und UNIX das Multiplexen von IO von mehreren TUN-Adaptern. Der Grund dafür ist, dass verschiedene TUN-Implementierungen gegebenenfalls verschiedene Adapter für mehrere Tunnel benötigen. Mit dem TAP-Windows-Treiber ist dies zwar nicht der Fall, es wurde der Vollständigkeit halber jedoch mitemplementiert.

4.4.4 libstrongswan

Hier galt es das Öffnen, Konfigurieren und Schließen von TUN-Geräten mit dem TAP-Windows-Treiber zu implementieren.

Der Treiber ist kompiliert auf der OpenVPN-Seite verfügbar[27] und der Quellcode auf Github[28]. Die Basis für die Implementierung war hier der existierende Programmcode in OpenVPN, spezifisch in der Datei "src/openvpn/tun.c.".

TAP-Geräte erstellen Das Erstellen von TAP-Geräten ist im Moment nicht unterstützt. Der Grund dafür ist, dass die Funktionalität von "devcon.exe" dafür nachgebaut werden müsste, was sehr zeitaufwendig ist. Des weiteren ist der Quellcode von "devcon.exe" nicht offen, weshalb es Lizensierungsprobleme damit geben könnte.

TAP-Geräte suchen Um das TAP-Gerät zu nutzen versucht der Code zuerst nach Netzwerkgeräten mit der ID "tap0901" zu suchen, welche die ID für TAP-Geräte ist. Dies wird über die Registry getan. Hierbei wird im Pfad

```
HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Control\\Class\\{4D36E972-E325-11CE-BFC1-08002BE10318
```

nach Verbindungen gesucht, deren "ComponentId"-Feld den Wert "tap0901" besitzen. Bei passenden Schlüsseln wird der Wert "NetCfgInstanceId" an eine linked list angehängt. Die NetCfgInstanceId identifiziert jeden Netzwerkadapter unter Windows mittels einer einzigartigen ID. Vor dem Beenden der Funktion wird die linked list zurückgegeben. Diese Funktionalität ist in der Funktion get_tap_reg() in der Datei "src/libstrongswan/networking/tun_device.c" gekapselt.

Der Code hierfür ist im Appendix unter Code 18 aufzufinden.

TAP-Geräte öffnen TAP-Geräte sind im Usermode über den Globalen Addressraum für Userspaceadapter unter "\\.\\Global\\$NetCfgInstanceId.tap" erreichbar. "$NetCfgInstanceId" steht hierbei für die vorher mit "get_tap_req()" gefundenen "NetCfgInstanceId"-Werte der gefundenen TAP-Adapter.

```
CreateFile(device_path, GENERIC_READ | GENERIC_WRITE, 0,
                    0, OPEN_EXISTING, FILE_ATTRIBUTE_SYSTEM |
                        FILE_FLAG_OVERLAPPED, 0);
```

[27] https://openvpn.net/index.php/open-source/downloads.html
[28] https://github.com/OpenVPN

Das Handle kann nun genutzt werden um Pakete vom TAP-Gerät zu lesen und zu schreiben. Mittels DeviceIoControl kann die Konfiguration des Geräts geändert werden, wie die IP des virtuellen Routers, den Status des Geräts sowie der Modus des Geräts.

Für die Portierung wurden die FDs für Handles ausgetauscht und für das Benachrichtigen über Änderungen in der Liste der TUN-Geräte wurde ein Event verwendet.

Da Eventbasiertes IO-Multiplexing genutzt wird, werden verschiedene Events zum Lesen und Schreiben verwendet. Die Namen der Events werden als Attribut des Objekts in "tun_device_t->read_event_name" "tun_device_t->write_event_name" gespeichert. Sie können mittels "tun_device_t->get_read_event_name()' und "tun_device_t->get_read_event_name()" gelesen werden.

Ändern der MTU eines TAP-Geräts Das Ändern der MTU ist im Moment nicht unterstützt. Der Treiber ermöglicht es, die MTU über die Windows-Registry zu ändern. Dort wird der Schlüssel "MTU" dafür genutzt. Er steht standardmäßig auf "1500", welches die Standard-MTU von Ethernet ist.

Konfiguration eines TAP-Geräts Die Konfiguration eines TAP-Geräts geschieht mit den IOCTL-Werten aus /autoreffig:tap-ioctls.

Bei der Konfiguration eines TAP-Geräts für die Benutzung mit strongSwan müssen die folgenden Schritte ausgeführt werden:

- Das Gerät in den TUN-Modus setzen. Dabei werden als Parameter die lokale IP (169.254.128.127), die IP des virtuellen Routers (169.254.128.128) und die Netzmaske des entfernten Netzwerks (255.255.255.255) gesetzt. (TAP_WIN_IOCTL_CONFIG_TUN)
- Deaktivierung der Überprüfung des "Sender protocol address"-Feldes. (TAP_WIN_IOCTL_CONFIG_SET_SRC_CHECK)
- Adapter in den "up"-Zustand setzen (einschalten). Nach dem Setzen des Zustands muss kurz gewartet werden um sicher zu gehen, dass das Gerät auch aktiv ist. (TAP_WIN_IOCTL_SET_MEDIA_STATUS)

Folgend ist der Code für das Konfigurieren eines TAP-Geräts:

Code 3: Konfiguration eines TAP-Geräts

```
1  /**
2   * Initialize the tun device
3   */
4  static bool init_tun(private_tun_device_t *this, const char *name_tmpl)
5  [...]
6  #elif defined(WIN32)
7          /* WIN32 TAP driver stuff*/
8          /* Check if there is an unused tun device following the IPsec name
               scheme*/
9          enumerator_t *enumerator;
10         char *guid;
11         BOOL success = FALSE;
12         linked_list_t *possible_devices = get_tap_reg();
13         memset(this->if_name, 0, sizeof(this->if_name));
14
15         this->read_event_name = malloc(WIN32_TUN_EVENT_LENGTH);
```

```
16          this->write_event_name = malloc(WIN32_TUN_EVENT_LENGTH);
17          /* Iterate over list */
18          enumerator = possible_devices->create_enumerator(possible_devices);
19          /* Try to open that device */
20          while(enumerator->enumerate(enumerator, &guid))
21          {
22              if (!success){
23                  /* Set mode */
24                  char device_path[256];
25                  /* Translate dev name to guid */
26                  /* TODO: Fix. device_guid should be */
27                  snprintf (device_path, sizeof(device_path), "%s%s%s",
                        USERMODEDEVICEDIR, guid, TAP_WIN_SUFFIX);
28
29                  this->tunhandle = CreateFile(device_path, GENERIC_READ |
                        GENERIC_WRITE, 0,
30                      0, OPEN_EXISTING, FILE_ATTRIBUTE_SYSTEM |
                            FILE_FLAG_OVERLAPPED, 0);
31                  if (this->tunhandle == INVALID_HANDLE_VALUE)
32                  {
33                      DBG1(DBG_LIB, "could not create TUN device %s", device_path);
34                  }
35                  else
36                  {
37                      memcpy(this->if_name, guid, strlen(guid));
38                      success = TRUE;
39                  }
40              }
41              else
42              {
43                  break;
44              }
45              /* device has been examined or used, free it */
46              free(guid);
47          }
48
49          /* possible_devices has been freed while going over the enumerator.
50           * Therefore it is not necessary to free the elements in the list now.
51           */
52          enumerator->destroy(enumerator);
53          possible_devices->destroy(possible_devices);
54          if (!success)
55          {
56              return FALSE;
57          }
58          /* set correct mode */
59          /* We set a fake gateway of 169.254.254.128 that we route packets over
60           The TAP driver strips the Ethernet header and trailer of the Ethernet
                frames
61           before sending them back to the application that listens on the handle
                */
62          struct in_addr ep[3];
63          ULONG status = TRUE;
64          DWORD len;
65          /* Local address (just fake one): 169.254.128.127 */
```

```
66          ep[0].S_un.S_un_b.s_b1 = 169;
67          ep[0].S_un.S_un_b.s_b2 = 254;
68          ep[0].S_un.S_un_b.s_b3 = 128;
69          ep[0].S_un.S_un_b.s_b4 = 127;
70          /*
71           * Remote network. The tap driver validates it by masking it with the
                remote_netmask
72           * and then comparing hte result against the remote network (this value
                here).
73           * If it does not match, an error is logged and initialization fails.
74           * (local & remote_netmask ? local)
75           * The driver does proxy arp for this network and the local address.
76           */
77          /* We need to integrate support for IPv6, too. */
78          /* Just fake a link local address for now (169.254.128.128) */
79          ep[1].S_un.S_un_b.s_b1 = 169;
80          ep[1].S_un.S_un_b.s_b2 = 254;
81          ep[1].S_un.S_un_b.s_b3 = 128;
82          ep[1].S_un.S_un_b.s_b4 = 128;
83          /* Remote netmask (255.255.255.255) */
84          ep[2].S_un.S_un_b.s_b1 = 255;
85          ep[2].S_un.S_un_b.s_b2 = 255;
86          ep[2].S_un.S_un_b.s_b3 = 255;
87          ep[2].S_un.S_un_b.s_b4 = 255;
88
89          if(!DeviceIoControl (this->tunhandle, TAP_WIN_IOCTL_CONFIG_TUN,
90                      ep, sizeof (ep),
91                      ep, sizeof (ep), &len, NULL))
92          {
93              DBG1 (DBG_LIB, "WARNING: The TAP-Windows driver rejected a
                    TAP_WIN_IOCTL_CONFIG_TUN DeviceIoControl call.");
94          }
95
96          ULONG disable_src_check = FALSE;
97          if(!DeviceIoControl(this->tunhandle, TAP_WIN_IOCTL_CONFIG_SET_SRC_CHECK,
98                      &disable_src_check, sizeof(disable_src_check),
99                      &disable_src_check, sizeof(disable_src_check), &len, NULL))
100         {
101             DBG1 (DBG_LIB, "WARNING: The TAP-Windows driver rejected a
                    TAP_WIN_IOCTL_CONFIG_SET_SRC_CHECK DeviceIoControl call.");
102         }
103         ULONG driverVersion[3] = {0 , 0, 0};
104         if(!DeviceIoControl(this->tunhandle, TAP_WIN_IOCTL_GET_VERSION,
105                     &driverVersion, sizeof(driverVersion),
106                     &driverVersion, sizeof(driverVersion), &len, NULL))
107         {
108             DBG1(DBG_LIB, "WARNING: The TAP-Windows driver rejected a
                    TAP_WIN_IOCTL_GET_VERSION DeviceIoControl call.");
109         }
110         else
111         {
112             DBG1(DBG_LIB, "TAP-Windows driver version %d.%d available.",
                    driverVersion[0], driverVersion[1]);
113         }
114         /* Set device to up */
```

```
115
116         if (!DeviceIoControl (this->tunhandle, TAP_WIN_IOCTL_SET_MEDIA_STATUS,
117                               &status, sizeof (status),
118                                 &status, sizeof (status), &len, NULL))
119         {
120             DBG1 (DBG_LIB, "WARNING: The TAP-Windows driver rejected a
                    TAP_WIN_IOCTL_SET_MEDIA_STATUS DeviceIoControl call.");
121         }
122
123             /* Give the adapter 2 seconds to come up */
124         /* Create event with special template */
125         snprintf(this->read_event_name, WIN32_TUN_EVENT_LENGTH,
                WIN32_TUN_READ_EVENT_TEMPLATE, this->if_name);
126         snprintf(this->write_event_name, WIN32_TUN_EVENT_LENGTH,
                WIN32_TUN_WRITE_EVENT_TEMPLATE, this->if_name);
127         sleep(2);
128         return TRUE;
129         [...]
130 }
```

Lesen und Schreiben vom Adapter Beim Öffnen des Adapters wurde ein Handle erstellt. Durch asynchrone Lese- und Schreiboperationen mittels ReadFile() und WriteFile() unter Benutzung von "OVERLAPPED"-Strukturen und Events lassen sich Pakete lesen und schreiben.

tun_device_t Folgend der Code für das Erstellen eines Objects vom Typ "tun_device_t".

Code 4: Code für das Erstellen von tun_device_t

```
1  /*
2   * Described in header
3   */
4  tun_device_t *tun_device_create(const char *name_tmpl)
5  {
6          private_tun_device_t *this;
7
8          INIT(this,
9                  .public = {
10                         .read_packet = _read_packet,
11                         .write_packet = _write_packet,
12                         .get_mtu = _get_mtu,
13                         .set_mtu = _set_mtu,
14                         .get_name = _get_name,
15                         /* For WIN32, that's a handle. */
16 #ifdef WIN32
17                         .get_handle = _get_handle,
18                         .get_write_event_name = _get_write_event_name,
19                         .get_read_event_name = _get_read_event_name,
20 #else
21                         .get_fd = _get_fd,
22 #endif /* WIN32 */
23                         .set_address = _set_address,
24                         .get_address = _get_address,
25                         .up = _up,
26                         .destroy = _destroy,
```

```
27                  },
28 #ifdef WIN32
29                  .tunhandle = NULL,
30 #else
31                  .tunfd = -1,
32                  .sock = -1,
33 #endif /* WIN32 */
34          );
35
36          if (!init_tun(this, name_tmpl))
37          {
38                  free(this);
39                  return NULL;
40          }
41          DBG1(DBG_LIB, "created TUN device: %s", this->if_name);
42
43 #ifdef WIN32
44 #else
45          this->sock = socket(AF_INET, SOCK_DGRAM, 0);
46          if (this->sock < 0)
47          {
48                  DBG1(DBG_LIB, "failed to open socket to configure TUN device");
49                  destroy(this);
50                  return NULL;
51          }
52 #endif /* WIN32 */
53          return &this->public;
54 }
55
56 #endif /* TUN devices supported */
```

Code 5: Relevanter code für tun_device_t->destroy()

```
1 METHOD(tun_device_t, destroy, void,
2          private_tun_device_t *this)
3 {
4 #ifdef WIN32
5          /* close file handle, destroy interface */
6          CloseHandle(this->tunhandle);
7          free(this->read_event_name);
8          free(this->write_event_name);
9 #else
10         [...]
11 #endif
12         DESTROY_IF(this->address);
13         free(this);
14 }
```

4.4.5 kernel-iph

kernel-iph ist ein Plugin für ”charon”, das das Kernel-Interface für die Verwaltung von IP-Adressen und Routen implementiert.

Ursprünglich war das Verwalten von IP-Adressen im Master-Zweig nicht implementiert. Eine vollständige Implementierung existiert im Zweig ”win-vip”, die in den Zweig 'windows-libipsec'

gemerged wurde, um eine vollständige Implementierung dort zu erhalten.

Nachgerüstet werden musste Code zum Beachten des ”charon.install_virtual_ip_on” Parameters der in der Datei ”strongswan.conf” definiert werden kann und den libipsec nutzt um die Installation der empfangenen IP-Adresse(n) auf den richtigen Adapter zu erzwingen. Der Code kopiert den Adapternamen während der Initialisierung des Plugins, daher ändert ein Ändern des Parameters während der Laufzeit nicht die Schnittstelle, auf die die IP-Adresse installiert werden würde.

Die bekannten IP-Adressen werden in kernel-iph in einer Datenstruktur gespeichert, um sie schnell nachschlagen zu können und Berechnungen zu tätigen. Die Struktur wird einerseits durch manuelles Einfügen und Entfernen von selbstinstallierten Adressen bewerkstelligt, als auch durch Events, die die Anwendung über ein Handle vom Kernel empfängt.

Code 6: Ergänzung zu private_kernel_iph_net_t

```
/**
 * Private data of kernel_iph_net implementation.
 */
struct private_kernel_iph_net_t {
        [...]
        /**
         * Whether to install virtual IPs
         */
        bool install_virtual_ip;

        /**
         * Where to install virtual IPs
         */

        char *install_virtual_ip_on;
};
```

Code 7: Code für add_ip

```
METHOD(kernel_net_t, add_ip, status_t,
        private_kernel_iph_net_t *this, host_t *vip, int prefix, char *name)
{
        if (!this->install_virtual_ip)
        {       /* disabled by config */
                return SUCCESS;
        }
        MIB_UNICASTIPADDRESS_ROW row;
        u_long status;
        iface_t *iface = NULL, *entry = NULL;

        /* Print out all known interfaces */
        enumerator_t *enumerator = this->ifaces->create_enumerator(this->ifaces);
        while(enumerator->enumerate(enumerator, &entry))
        {
            DBG1(DBG_KNL, "interface %s\n index: %d\n description %s\n type %d",
                entry->ifname, entry->ifindex, entry->ifdesc, entry->iftype);
        }
        enumerator->destroy(enumerator);
        /* name of the MS Loopback adapter */
```

```
20          if (!this->install_virtual_ip_on ||
                this->ifaces->find_first(this->ifaces, (void*)iface_by_name,
21                                                  (void**)&iface,
                                                        this->install_virtual_ip_on)
                                                        != SUCCESS)
22          {
23              name = "{DB2C49B1-7C90-4253-9E61-8C6A881194ED}";
24          }
25          else
26          {
27              name = this->install_virtual_ip_on;
28          }
29          host2unicast(vip, prefix, &row);
30
31          row.InterfaceIndex = add_addr(this, name, vip, TRUE);
32          if (!row.InterfaceIndex)
33          {
34                  DBG1(DBG_KNL, "interface '%s' not found", name);
35                  return NOT_FOUND;
36          }
37
38          status = CreateUnicastIpAddressEntry(&row);
39          if (status != NO_ERROR)
40          {
41                  DBG1(DBG_KNL, "creating IPH address entry failed: %lu", status);
42                  remove_addr(this, vip);
43                  return FAILED;
44          }
45          return SUCCESS;
46  }
```

Code 8: Code für del_ip

```
1  METHOD(kernel_net_t, del_ip, status_t,
2          private_kernel_iph_net_t *this, host_t *vip, int prefix, bool wait)
3  {
4          if (!this->install_virtual_ip)
5          {       /* disabled by config */
6                  return SUCCESS;
7          }
8
9          MIB_UNICASTIPADDRESS_ROW row;
10         u_long status;
11
12         host2unicast(vip, prefix, &row);
13
14         row.InterfaceIndex = remove_addr(this, vip);
15         if (!row.InterfaceIndex)
16         {
17                 DBG1(DBG_KNL, "virtual IP %H not found", vip);
18                 return NOT_FOUND;
19         }
20
21         status = DeleteUnicastIpAddressEntry(&row);
22         if (status != NO_ERROR)
```

```
23          {
24                  DBG1(DBG_KNL, "deleting IPH address entry failed: %lu", status);
25                  return FAILED;
26          }
27
28          return SUCCESS;
29  }
```

Code 9: Ergänzung zu kernel_iph_net_create()

```
1  *
2   * Described in header.
3   */
4  kernel_iph_net_t *kernel_iph_net_create()
5  {
6          [...}
7                  .install_virtual_ip = lib->settings->get_bool(lib->settings,
8                                                  "%s.install_virtual_ip", TRUE,
                                                        lib->ns),
9                  .install_virtual_ip_on = lib->settings->get_str(lib->settings,
10                                                 "%s.install_virtual_ip_on",
                                                        NULL, lib->ns),
11             [...]
12 }
```

Für das Implementieren des Codes für den TAP-Treiber werden Magic Numbers benötigt, welche sich im Quellcode von OpenVPN finden lassen. Diese wurden übernommen und in einer Header-Datei angelegt. Teilweise wurden auch die Konstanten für die Erzeugung der Namen der Events für das IO-Multiplexen dort abgelegt. Die Datei ist im Appendix unter Code 17 zu finden.

Der gesamte gepatchte Quellcode von strongSwan ist auf der CD unter strongSwan/ zu finden Der Patch ansich ist unter patches/strongSwan/tap-handling.patch zu finden.

4.5 TAP-Treiber

Im Zuge der Bachelorarbeit (BA) wurde ein Patch für den TAP-Windows-Treiber entwicklelt, um die Prüfung der Quell-IP der ARP-Requests zu deaktivieren. Das ist erforderlich, um mit der virtuellen IP des Clients mit dem entfernten IP-Netzwerk kommunizieren zu können.

4.5.1 Bauen und Installieren des Treibers

Um den Treiber zu bauen, wird zuallererst der Quellcode benötigt, sowie die Vorraussetzungen, welche auf der Github-Seite dort aufgelistet sind. Die aktuellen Minimalvorraussetzungen sind Python 2.7, das Microsoft Windows 7 WDK (Windows Driver Kit) und ein Windows Code signing certificate. Das Zertifikat muss nicht valid sein, wenn der Treiber nicht produktiv ausgerollt werden soll. Der Grund ist, dass unter Windows das Überprüfen der Treibersignatur deaktiviert werden kann, aber der Treiber muss trotzdem signiert sein. Der Trust-path muss aber nicht zu einem öffentlichen Trust Anchor (eine CA) führen. Um den Treiber signieren zu können, wird die gesamte Trust Chain benötigt, sowie der private Schlüssel des Zertifikats. Um ein Zertifikat mit dem dazugehörigen privaten Schlüssel importieren zu können, müssen die Dateien in einen

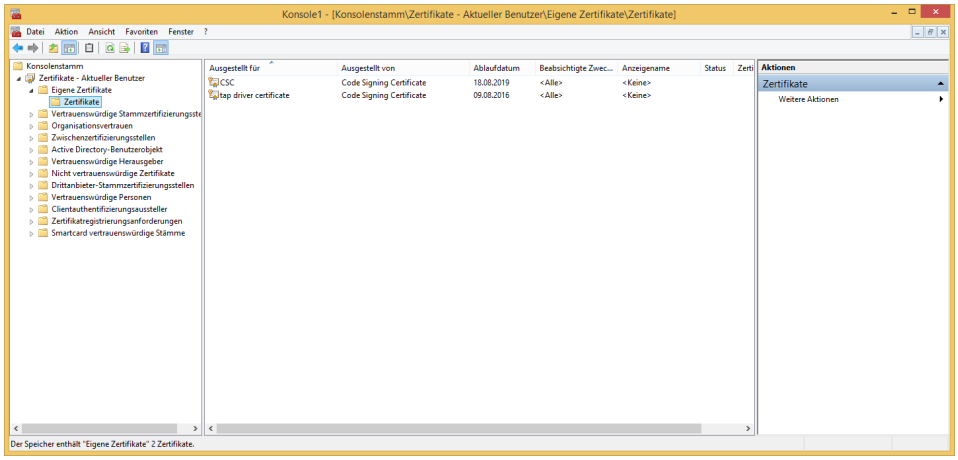

Abbildung 12: Eigene-Zertifikate-Menü

PKCS#12-Container gepackt werden. Dieser kann mit OpenSSL erstellt werden, zum Beispiel mit diesem Befehl:

Code 10: OpenSSL PKCS#12

```
1 openssl pkcs12 -export -in key key.key -in certificate.pem -out pkcs12.p12
```

Die mit diesem Befehl erstellte Datei pkcs12.p12 kann dann durch einen Doppelklick auf sie in den Windows importiert werden.

Der Befehl um den Treiber dann zu bauen kann so aussehen:

Code 11: TAP-Windows bauen

```
1 python buildtap.py -c -b -d --cert="csc" --sign
    --crosscert="C:\Users\Noel\certs\rootca.pem"
```

Die Ausgabe des Befehls listet die Kompilierung, sowie das Signieren der Dateien auf. Wie zu sehen ist, muss das Root-CA-Zertifikat als Datei vorhanden sein. Das Code Signing certificate wird nur mit seinem Common Name angegeben. Damit das WDK es findet, muss es in ”Eigene Zertifikate” zu finden sein.

Mit Windows im Testmodus kann der Treiber geladen werden und TAP-Geräte konnen mittels tapinstall.exe (Was eigentlich ”devcon.exe” ist). Die Datei ist im Quellcode des Treibers nicht mitenthalten. Sie kann jedoch durch die Installation des Treiber-Pakets von der OpenVPN-Webseite erhalten werden. Nach der Installation des Pakets ist die Datei unter *C:\Program Files\TAP-Windows\bin\tapinstall.exe* zu finden und kann zur Erstellung und zum Löschen von TAP-Geräten genutzt werden.

Der erste Parameter ist offensichtlich ”install” oder ”remove”. Der Pfad zur .inf-Datei über den Treiber muss beim Erstellen eines Geräts angegeben werden. Hier im Beispiel ist es die Datei, die mittels des Python-Skripts erstellt wurde. Bei der Installation dieses Treibers wird der von mir kompilierte Treiber installiert, welcher sich unter ”C:\Users\Noel\tap-driver\dist\amd64\OemVista.inf” befindet.

Code 12: Erstellung eines TAP-Geraets

```
1 "C:\Program Files\TAP-Windows\bin\tapinstall.exe" install
    "C:\Users\Noel\tap-driver\dist\amd64\OemVista.inf" tap0901
```

Code 13: Löschung aller TAP-Geraete

```
1 "C:\Program Files\TAP-Windows\bin\tapinstall.exe" remove tap0901
```

ABI Die ABI des Treibers ist undokumentiert. Im Folgenden ist eine Auflistung der Funktionen der ABI. Sie wurden aus der Datei "src/device.c" extrahiert und aufgeschlüsselt. Für mehr Informationen bezüglich des Verhaltens und der genauen Eingabeparameter sollte die Datei konsultiert werden.

Patch Bei der Entwicklung des Patchs wurde viel Zeit darauf verwendet die funktionalen Komponenten zu finden, die bei der Verarbeitung von Address Resolution Protocol (ARP)-Requests ausgeführt werden, sowie die Codepfade zu finden, die bei der Verarbeitung von Daten abgelaufen werden.

Der entwickelte Patch deaktiviert die Überprüfung der Quell-IP der ARP-Requests, die vom Treiber verarbeitet werden. Wenn die Überprüfung erfolgreich war, so wird der ARP-Request beantwortet und dem Kernel wird damit mitgeteilt, wie die MAC-Adresse des virtuellen Routers lautet.

Für die Implementierung der Routen wurde die Nutzung eines virtuellen Routers gewählt, da das Routen aller IP-Adressen über das Gerät die ARP-Tabelle des Clients mehr gefüllt hätte und eine ARP-Adressabfrage bei jeder Kommunikation mit einer neuen IP-Adresse zu Folge hätte.

Der Patch wurde entwickelt, indem nach der Behandlung von ARP-Paketen im Quellcode des Treibers gesucht wurde. Dafür wurde nach "ARP" und "arp" mittels "grep" im Quelltext gesucht. Dabei wurde die Funktion "ProcessARP" in "src/txpath.c" gefunden, in der ARP-Requests verarbeitet werden. In dieser Funktion werden verschiedene Felder des Pakets überprüft, unter anderem das "Sender protocol address"-Feld.

Die Überprüfung des "Sender protocol address"-Felds wurde optional gemacht, wobei es standardmäßig aktiviert ist. Die Kontrolle dieses Feldes kann über ein IOCTL-Wert gesteuert werden. Der momentane Zustand der Option wird im Adapter-Kontext des TAP-Adapters gespeichert.

Er wurde 2016-08-15 in der OpenVPN Community auf freenode im Kanale #openvpn-meeting besprochen und bekam ein feature-ack. Ein Code-ack steht noch aus. Das Ticket ist unter `https://community.openvpn.net/openvpn/ticket/721` zu finden.

Der komplette Patch ist auf der CD unter "patches/tap-windows/0001-implement-source-ip-check.patch" zu finden.

4.6 Test des Codes

Zum Konfigurieren und Bauen des Quellcodes wurde der folgende Befehl verwendet. Im Testszenario wurde der Quellcode auf Linux für Windows 64-Bit cross kompiliert und in einer Virtual Machine (VM) mit Windows 8.1 ausgeführt. Die gebauten Binärdateien wurden über ein geteiltes Verzeichnis der VM zugänglich gemacht. Danach wurden sie in "C:\\Users\\Thermi\\bin kopiert und in 'cmd' mit Administratorrechten ausgeführt. Die nötige Dateistruktur sollte wie in Abbildung 14 auf Seite 36 aussehen.

Im Unterordner "swanctl" befindet sich die Dateien "swanctl.conf", sowie die Ordner "bliss", "ecdsa", "pkcs8", "pkcs12", "pubkey", "rsa", "x509", "x509aa", "x509ac", "x509ca", "x509crl" und"x509ocsp". Im Ordner "rsa" befindet sich der private Schlüssel für den Client. Im Ordner "x509" befindet sich

IOCtl	Makro	Eingabe	Ausgabe	Zweck	Kommentar
1	TAP_WIN_IOCTL_GET_MAC	NULL	char[6]	Gibt die MAC-Adresse des Geräts zurück	
2	TAP_WIN_IOCTL_GET_VERSION	NULL	ULONG[3]	Gibt die Version des Treibers zurück	
3	TAP_WIN_IOCTL_GET_MTU	NULL	ULONG	Gibt die MTU des Geräts zurück	
4	TAP_WIN_IOCTL_GET_INFO	NULL	N/A	Gibt Informationen über den Adapter zurück	Ist laut Code für NDIS6 nicht implementiert
5	TAP_WIN_IOCTL_CONFIG_POINT_TO_POINT	IPADDR[2] (2*4 CHAR)	NULL	Setzt das Gerät in den Point-To-Point-Modus	
6	TAP_WIN_IOCTL_SET_MEDIA_STATUS	ULONG	NULL	Setzt das Gerät in den "Up" oder "Down"-Zustand	
7	TAP_WIN_IOCTL_CONFIG_DHCP_MASQ	IPADDR[4] (4*4 CHAR)	NULL	Aktiviert den internen DHCP-Server, DHCP-IP-Adresse, DHCP-Netzmaske, DHCP-Server-IP, Leasetime	
8	TAP_WIN_IOCTL_GET_LOG_LINE	allokierter String (char *)	NULL	Gibt eine Debug-Log-Zeile zurück	
9	TAP_WIN_IOCTL_CONFIG_DHCP_SET_OPT	char[256]	NULL	Setzt die DHCP-Optionen	
10	TAP_WIN_IOCTL_CONFIG_TUN	IPADDR[3] (3*4 char)	NULL	Setzt das Gerät in den TUN-Modus, lokale IP, entferntes Netzwerk, entfernte Netzmaske	
11	TAP_WIN_IOCTL_CONFIG_SET_SRC_CHECK	ULONG	NULL	Deaktiviert oder Aktiviert den ARP-Source-Check. "0" deaktiviert ihn. "1" aktiviert ihn (Standard)	

Tabelle 1: TAP-Windows-Treiber IOCtls

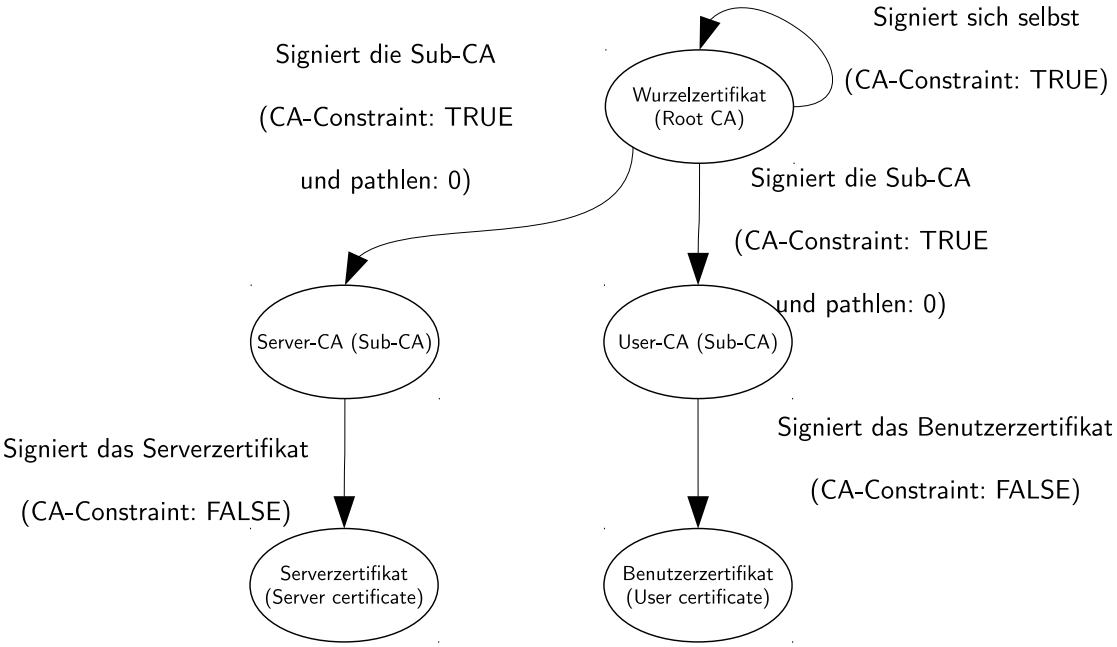

Abbildung 13: Eine exemplarische CA-Struktur

das Zertifikat des Clients und im Ordner "x509ca" alle Certificate Authority (CA)-Zertifikate vom Zertifikat des Clients bis zum selbstsignierten Wurzelzertifikat, sowie das Zertifikat der Server-CA.

Die Datei "this_log" wurde von der Logger-Konfiguration des Dienstes erstellt. Hierbei wurde zuerst "charon-svc.exe" in einem ersten Fenster ausgeführt. In einem zweiten Fenster wurde die Konfiguration geladen und der Tunnelaufbau mittels "swanctl.exe" gestartet. Hierbei wurden die folgenden Befehle verwendet: "swanctl.exe –load-all" und "swanctl –i –child bar". Nach dem Testen wurde der Tunnel, sollte "charon-svc.exe" nicht gecrasht sein, mit "swanctl -t –ike foo" beendet, sodass keine virtuellen IP-Adressen oder Routen für den nächsten Test zurückbleiben.

Wenn "charon-svc.exe" getötet wird, während ein Tunnel mit Routen und IP-Adressen aufgebaut ist, so verbleiben IP-Adressen und Routen auf dem Adapter, was verhindern kann dass dieselben Routen und IP-Adressen wieder installiert werden. Das führt dazu, dass der Aufbau der CHILD_SA fehlschlägt und ein neuer Tunnel nicht aufgebaut werden kann. In diesem Fall müssen sie manuell über die Nutzung von "route delete" oder "netsh" gelöscht werden oder Windows neu gestartet werden.

OpenSSL auf Windows wird für die Kryptographie benötigt und liegt nicht in den Standardsuchpfaden für Bibliotheken und Header. Aus diesem Grund werden die Header und Bibliotheken über LDFLAGs in Include-Pfade auf der Kommandozeile übergeben.

Die Bibliotheken von OpenSSL wurden durch die Installation von OpenVPN beschafft. Bei der Installation von OpenVPN werden direkt die Bibliotheken von openssl mitinstalliert.

Die unter Code 16 und Code 15 bereitgestellten Konfigurationen wurden zur Konfiguration des Dienstes genutzt.

Code 14: ./configure und make

```
1 ./configure --host=x86_64-w64-mingw32 --prefix=/ --libdir=/bin --bindir=/bin
    --sbindir=/bin --disable-defaults --enable-monolithic --enable-static
    --enable-svc --enable-ikev2
```

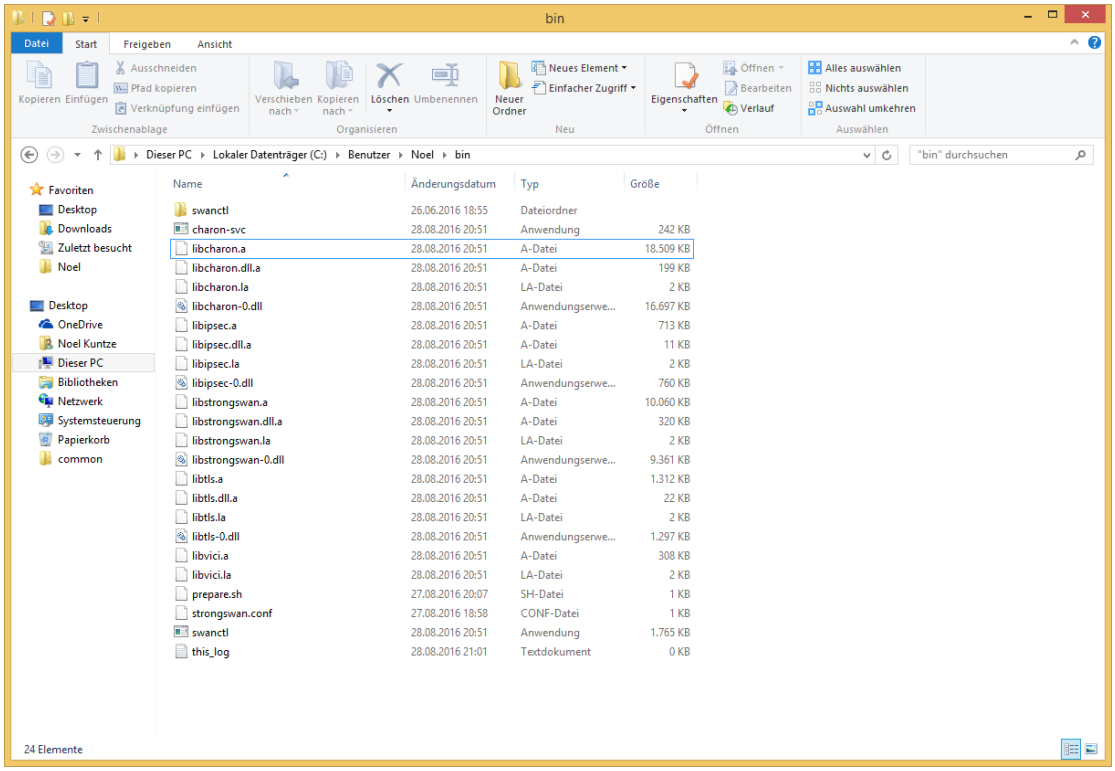

Abbildung 14: Ordnerstruktur nach dem Kopieren der Dateien

```
2 --enable-ikev1 --enable-nonce --enable-pem --enable-pkcs1 --enable-x509
      --enable-openssl --enable-socket-win --enable-kernel-wfp --enable-kernel-iph
      --enable-pubkey --enable-swanctl
3 --with-swanctldir=swanctl --with-strongswan-conf=strongswan.conf
      --enable-libipsec --enable-kernel-libipsec --enable-eap-tls
      --enable-mschapv2 --enable-eap-peap --enable-eap-gtc
4 --enable-eap-dynamic --enable-eap-identity --enable-md4 --enable-ipseckey
      --enable-dnscert --enable-files --enable-sha3 host_alias=x86_64-w64-mingw32
      CC=x86_64-w64-mingw32-gcc
5 CFLAGS=-g -O2 -Wall -Werror -Wno-pointer-sign -Wno-format-security -Wno-format
      -mno-ms-bitfields
      -I/home/thermi/FH-Stuff/UNITS-8/Bachelorarbeit/win32-headers/files/include/
6 LDFLAGS=-L/home/thermi/FH-Stuff/UNITS-8/Bachelorarbeit/win32-headers/files/lib/
      -L/home/thermi/FH-Stuff/UNITS-8/Bachelorarbeit/win32-headers/files/bin/
7 make clean && make
```

Der Befehl konfiguriert den strongSwan Quellcode zum Bauen der benötigten Plugins, sowie einiger Extras für die Authentifizierung und dazu den 64-Bit Compiler des MINGW32-Projekts zu nutzen.

Der Code wurde getestet, indem versucht wurde eine VPN-Verbindung zu einem IKE-Peer unter meiner Kontrolle aufzubauen. Der ausgehandelte Traffic Selector (TS) wurde so gewählt, dass die Beschränkung von Route based VPNs keine Rolle spielen. Die genutzte Konfiguration ist im Appendix unter Code 15 und Code 16 zu finden.

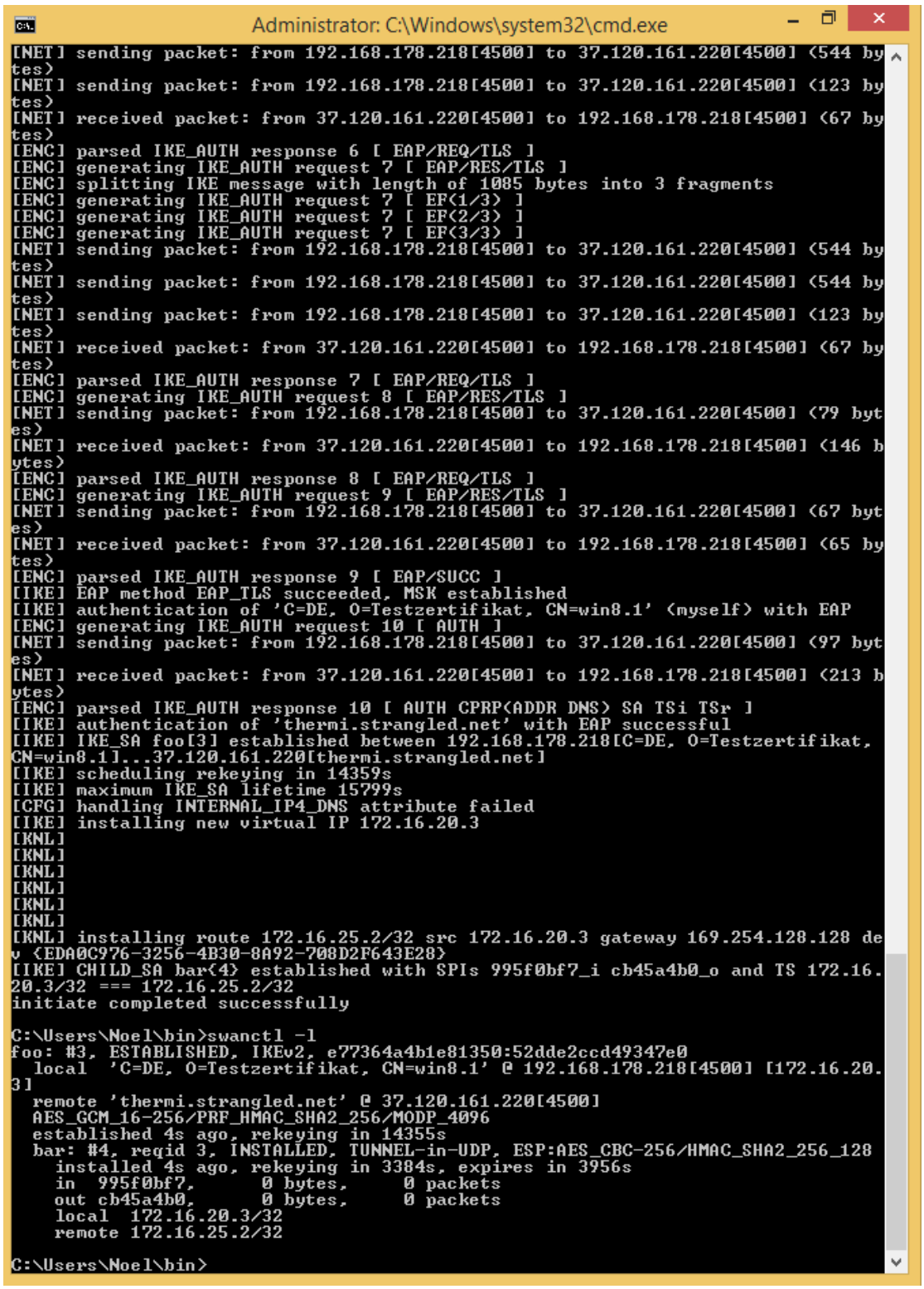

Abbildung 15: Ausgabe des Tunnelaufbaus und swanctl -l

4.7 Test der Verbindung

Das Testen der Verbindung wurde durch durch einfaches Pingen über die Verbindung bewerkstelligt. Dabei wird durch Dumpen der Pakete mit tcpdump und Wireshark überprüft, ob die ARP requests auf dem TUN-Adapter beantwortet werden und ob das Paket auf dem anderen Endpunkt auftaucht. Des weiteren werden die Traffic-Counter der SAs überprüft, um zu überprüfen, ob die Pakete jemals verarbeitet wurden.

4.8 Notwendige Features für Nutzung als RW auf Windows

Um strongSwan als RW auf Windows benutzbar zu machen werden neben dem Implementieren der Unterstützung von TAP-Geräten unter anderem die Unterstützung der Installation von DNS-Resolvereinträgen benötigt, da in der Regel interne DNS-Server an die RWs über Config Mode oder CPs gesendet werden, die sie nutzen sollen.

Des weiteren wird eine Möglichkeit benötigt, um über Versatile IKE Configuration Interface (VICI) nach Daten für dynamisch auftretende Passwortaufforderungen, die bei der Authentifizierung mittels XAUTH oder EAP auftreten können, zu fragen. Dies wird für die vollständige Unterstützen von MehrFaktorAuthentifizierung (MFA) benötigt.

Auf Windows benutzt das Tool ”swanctl”, welches dort zum Kontrollieren des Daemons genutzt wird, eine TCP-Verbindung über 127.0.0.1:4502 um mit dem Daemon zu kommunizieren. Dies ist im Vergleich zur Implementierung auf anderen Plattformen unsicher, da dort UNIX-Sockets genutzt werden, die standardmäßig aufgrund der Access Control Lists (ACLs) abgesichert werden und nur ”root” auf den Socket zugreifen kann. Der Wiki-Artikel macht auf die Unsicherheit des VICI-Protokolls, welches zwischen ”swanctl” und ”charon” gesprochen wird, besonders aufmerksam.

> The VICI protocol runs over a reliable transport protocol. As the protocol itself currently does not provide any security or authentication properties, it is recommended to run it over a UNIX socket with appropriate permissions.

[16h]

4.9 Probleme

Die implementierte Lösung funktioniert im Moment noch nicht. Das Problem ist, dass die Windows-eigenen Funktionen nicht wie dokumentiert funktionieren. Wie erläutert wird WaitForMultipleObjects() genutzt, um die IO-Operationen zu multiplexen. Diese Funktion beendet im Programmcode mit dem Wert ”1”, was bedeuted dass das zweite Objekt (In C werden Arrays von 0 auf indexiert.) im Array signalisiert wurde. Bei der Überprüfung mit WaitForSingleObject() ist das jedoch nicht der Fall. Wenn das Event signalisiert wäre, dann würde der folgende Aufruf den Wert ”WAIT_OBJECT_0” (0x0) zurückgeben:

```
1 WaitForSingleObject(event_array[offset], 0)
```

] Das ist jedoch nicht der Fall. Der Aufruf beendet mit ”WAIT_TIMEOUT” (0x102, 258), was bedeuted, dass das Event nicht signalisiert ist. Wenn der Rückgabewert ignoriert wird, dann befindet sich im Puffer kein valides IPv4-Packet.

Das Problem ist besonders evident bei der Betrachtung des Debug-Logs, welches sich auf Seite 58 finden lässt. Es wird vom Code in auf Seite 52 ausgegeben.

Des weiteren gibt die Funktion ”GetLastError()” den Fehlerwert 0 zurück, wenn sie nicht direkt nach dem Funktionsaufruf, der den Fehlerstatus setzte, aufgerufen wird. Es ist irrelevant, ob zwischen dem Aufruf von ”GetLastError()” nur eine einfache Zuweisung oder eine If-Abfrage steht. Dies ergibt keinen Sinn, da sich der Fehlerstatus scheinbar ändert, ohne dass eine Funktion aufgerufen wird, die das tun könnte.

5 Fazit

In der Arbeit wurden die nötigen Änderungen für strongSwan präsentiert, um libipsec auf Windows lauffähig zu machen. Dies inkludiert die Implementierung der Routinen für das Öffnen und Konfigurieren von TAP-Geräten, die vom TAP-Windows-Treiber bereitgestellt werden. Des weiteren wurde die Installation von IP-Adressen getestet und erweitert, sowie Code zum Lesen und Schreiben von Paketen auf Handles auf Windows.

Die Lösung ist unvollständig, da das Empfangen und Senden von Paketen noch nicht funktioniert. Nichtsdestotrotz wurde mit der BA ein Großteil der Basis geschaffen um "strongSwan" dazu zu befähigen TAP-Geräte zu nutzen und der TAP-Windows-Treiber in seiner Funktionalität erweitert.

Die Probleme, die die Implementierung schlussendlich scheitern ließen lassen es zu die Frage zu stellen, ob Anwendungen, die mit MinGW-W64 kompiliert wurden überhaupt fehlerfrei auf Handles von Usermodedevice-Treiber zugreifen können und ob nicht weitere Fehler in MinGW-W64 existieren, die das Implementieren der Kompatibilität mit TAP-Geräten verhindern. Die Alternativanwendung "OpenVPN" benutzt den Treiber problemlos, wobei sie jedoch mit Visual Studio kompiliert wird, was den kritischen Unterschied im Verhalten ausmachen könnte. Von mingw64 ist bekannt, dass es ab und zu Probleme mit falschen Headern und ähnlichem gibt. Es steht offen, ob es tatsächlich ein Problem mit dem eigenen geschriebenen Code aus der Bachelorarbeit ist, oder ob es tiefer herrürht.

Die weitere Verfolgung der Probleme in tiefere Schichten des Systems wurden aus Zeitmangel nicht weiter verfolgt.

Die gesamte Arbeit ist gemeinfrei und unterliegt der GPLv3. Somit kann sie, sowie der Quellcode, frei vertrieben werden, in der Hoffnung dass sie jemandem nutzt. Der Quellcode der Arbeit, sowie komplette Git-Bäume und diffs vom jeweiligen Root-Commit der jeweiligen Softwareprojekte sind auf der CD mitenthalten.

Es bleibt offen, ob es zu schaffen ist den Treiber Problemlos in strongSwan zu integrieren. Mit der Arbeit wurde auf jeden Fall eine gute Basis gelegt und die ABI des TAP-Windows-Treibers dokumentiert.

Literatur

[13] *Shrew Soft VPN Client Administrators Guide*. 2013. URL: https://www.shrew.net/static/help-2.1.x/vpnhelp.htm (besucht am 13. 08. 2016).

[16a] *bintec Secure IPSec Client*. 14. Apr. 2016. URL: http://pim.bintec-elmeg.com/common/ajax.php?modul_id=101&klasse=produkte_pdf&bereich=admin&com=detail&kanal=xml&system_id=871&sprache=en.

[16b] *CreateIoCompletionPort function (Windows)*. 2016. URL: https://msdn.microsoft.com/en-us/library/windows/desktop/aa363862(v=vs.85).aspx (besucht am 09. 08. 2016).

[16c] *IKEv1CipherSuites - strongSwan*. 2016. URL: https://wiki.strongswan.org/projects/strongswan/wiki/IKEv1CipherSuites (besucht am 13. 08. 2016).

[16d] *IKEv2CipherSuites - strongSwan*. 2016. URL: https://wiki.strongswan.org/projects/strongswan/wiki/IKEv2CipherSuites (besucht am 13. 08. 2016).

[16e] *IPsec Key Exchange*. 2016. URL: https://technet.microsoft.com/en-us/library/cc731752(v=ws.11).aspx (besucht am 25. 08. 2016).

[16f] *IpsecStandards - strongSwan*. 30. März 2016. URL: https://wiki.strongswan.org/projects/strongswan/wiki/IpsecStandards (besucht am 27. 07. 2016).

[16g] *PluginList - strongSwan*. 2016. URL: https://wiki.strongswan.org/projects/strongswan/wiki/PluginList (besucht am 29. 08. 2016).

[16h] *Vici - strongSwan*. 2016. URL: https://wiki.strongswan.org/projects/strongswan/wiki/Vici (besucht am 29. 08. 2016).

[16i] *WaitForMultipleObjects function (Windows)*. 2016. URL: https://msdn.microsoft.com/en-us/library/windows/desktop/ms687025(v=vs.85).aspx (besucht am 10. 08. 2016).

[16j] *Windows7 - strongSwan*. 12. Juli 2016. URL: https://wiki.strongswan.org/projects/strongswan/wiki/Windows7 (besucht am 31. 07. 2016).

[Abr+01] Abraham Shacham u. a. *RFC 3173 - IP Payload Compression Protocol (IPComp)*. Sep. 2001. URL: https://tools.ietf.org/html/rfc3173 (besucht am 01. 08. 2016).

[And11] Andreas Steffen. *The strongSwan IPsec Solution*. 15. Juni 2011. URL: https://www.strongswan.org/tcg/tcg_munich_2011.pdf.

[Cha+14] Charlie Kaufman u. a. *RFC 7296 - Internet Key Exchange Protocol Version 2 (IKEv2)*. Okt. 2014. URL: https://tools.ietf.org/html/rfc7296 (besucht am 27. 07. 2016).

[DB98] Daniel L. McDonald und Bao G. Phan. *RFC 2367 - PF_KEY Key Management API, Version 2*. Juli 1998. URL: https://tools.ietf.org/html/rfc2367 (besucht am 25. 07. 2016).

[Dou+98] Douglas Maughan u. a. *RFC 2408 - Internet Security Association and Key Management Protocol (ISAKMP)*. Nov. 1998. URL: https://tools.ietf.org/html/rfc2408 (besucht am 27. 07. 2016).

[Hil98] Hilarie K. Orman. *RFC 2412 - The OAKLEY Key Determination Protocol*. Nov. 1998. URL: https://tools.ietf.org/html/rfc2412 (besucht am 30. 07. 2016).

[JM05] Jan Hutter und Martin Willi. „strongSwan II Eine IKEv2-Implementierung für Linux“. Diplomarbeit. Rapperswil: Hochschule für Technik Rapperswil, 16. Dez. 2005. 186 S. URL: http://security.hsr.ch/theses/DA_2005_IKEv2.pdf (besucht am 28. 07. 2016).

[Jür16] Jürgen Honig. *Datenblatt NCP Secure Entry Client Windows*. 26. Jan. 2016. URL: https://www.ncp-e.com/fileadmin/pdf/library/datasheets/NCP_DB_Entry_Client_Win32_64.pdf (besucht am 14. 08. 2016).

[Mar+05] Markus Stenberg u. a. *RFC 3948 - UDP Encapsulation of IPsec ESP Packets*. Jan. 2005. URL: https://tools.ietf.org/html/rfc3948 (besucht am 27. 07. 2016).

[Mar14] Martin WIlli. *git.strongswan.org Git - WIP: Windows virtual IP notes win-vip*. 16. Sep. 2014. URL: https://git.strongswan.org/?p=strongswan.git;a=commit;h=d50fd962a9bfea277de9131375e793deb80874a5 (besucht am 25. 08. 2016).

[PRJ15] Petri Jokela, Robert Moskowitz und Jan Melen. *RFC 7402 - Using the Encapsulating Security Payload (ESP) Transport Format with the Host Identity Protocol (HIP)*. Apr. 2015. URL: https://tools.ietf.org/html/rfc7402 (besucht am 27. 07. 2016).

[SK05] Stephen Kent und Karen Seo. *RFC 4301 - Security Architecture for the Internet Protocol*. Dez. 2005. URL: https://tools.ietf.org/html/rfc4301 (besucht am 27. 07. 2016).

[Ste05a] Stephen Kent. *RFC 4302 - IP Authentication Header*. Dez. 2005. URL: https://tools.ietf.org/html/rfc4302 (besucht am 27. 07. 2016).

[Ste05b] Stephen Kent. *RFC 4303 - IP Encapsulating Security Payload (ESP)*. Dez. 2005. URL: https://tools.ietf.org/html/rfc4303 (besucht am 27. 07. 2016).

[TM14] Tobias Brunner und Martin WIlli. *Kernel-wfp - strongSwan*. 7. Okt. 2014. URL: https://wiki.strongswan.org/projects/strongswan/wiki/Kernel-wfp (besucht am 25. 08. 2016).

6 Appendix

6.1 Featurematrix

Die Daten aus diesen Tabellen wurden unter Nutzung der öffentlich zugänglichen Dokumente auf den entsprechenden Herstellerwebseiten erstellt.

Wenn bei einem symmetrischen Verschlüsselungsalgorithmus kein Modus angegeben war, wurde Cipher Block Chaining (CBC) angenommen.

Wenn bei einem Authentifizierungsmodus nicht alle unterstützen Permutationen angegeben waren, wurden alle standardisierten Permutationen als unterstützt angenommen.

Wenn ein Feature nicht explizit als unterstützt angegeben wurde, so wurde angenommen dass es nicht unterstützt wird.

Offenbar ist der ”bintec Secure IPsec Client” nur ein Rebranding des ”NCP Secure Entry Client”, wenn man vom Graphical User Interface (GUI) ausgehen kann. Ein weiteres Indiz ist, dass im Installer des ”bintec Secure IPsec Client” der String ”NCP engineering GmbH” auftaucht. Daher wäre es zu verstehen, wenn aus den Dokumentationen der beiden Produkte Featureparität hervorkäme. Dem ist aber nicht so, wie aus den Tabellen hervorgeht.

Diese Tabellen beziehen sich nur auf die Fähigkeiten, die ab Windows 7 unterstützt sind. Sie machen keine Aussage über die Unterstützung der Software auf anderen Platformen und hat keinen Anspruch auf Vollständigkeit.

Symbolik

x Unterstützt

o Nicht unterstützt

? unbekannt

Referenzen zur Bibliographie

- strongSwan[29][30]
- Windows Agile VPN Client[31]
- Shrewsoft VPN Client[32]
- NCP Secure Entry Client[33]
- bintec Secure IPsec Client[34]

[29] 16c.
[30] 16d.
[31] 16j.
[32] 13.
[33] Jür16.
[34] 16a.

Software	IKE-Versionen
strongSwan	IKEv1, IKEv2
Windows Agile VPN Client	IKEv1+L2TP, IKEv2
Shrewsoft VPN Client	IKEv1
NCP Secure Entry Client	IKEv1, IKEv2
bintec Secure IPsec Client	IKEv1, IKEv2

Tabelle 2: Unterstützte IKE-Versionen der IPsec-Implementierungen

Software	Lizenz
strongSwan	MIT/GPLv2
Windows Agile VPN Client	Proprietär
Shrewsoft VPN Client	Shareware
NCP Secure Entry Client	Proprietär
bintec Secure IPsec Client	Proprietär

Tabelle 3: Lizenzen der IPsec-Implementierungen

6.2 Testkonfiguration

Code 15: Testkonfiguration - swanctl.conf

```
connections {
        foo {
            version = 2
            dpd_delay = 10
            dpd_timeout = 60
            fragmentation = yes
            send_cert = always
            remote_addrs = 37.120.161.220
            proposals = aes256gcm16-prfsha256-modp4096
            vips = 0.0.0.0
                        mobike = no
                        encap = yes
            local {
                auth = eap-tls
                certs = certificate.pem
            }
            remote {
                auth = pubkey
                id = thermi.strangled.net
                cacerts = serverca.pem
            }
            children {
                    bar {
                        dpd_action = restart
                        start_action = none
                        esp_proposals = aes256-sha256-ecp521
                        remote_ts = 172.16.25.2/32
                    }
            }
        }
}
```

Software / Modus	strongSwan	Windows	Shrewsoft	NCP	bintec
AES-128-CBC	x	x	x	x	x
AES-192-CBC	x	x	x	x	x
AES-256-CBC	x	x	x	x	x
AES-128-GCM-8	x	o	o	o	o
AES-128-GCM-12	x	o	o	o	o
AES-128-GCM-16	x	o	o	o	o
AES-192-GCM-8	x	o	o	o	o
AES-192-GCM-12	x	o	o	o	o
AES-192-GCM-16	x	o	o	o	o
AES-256-GCM-8	x	o	o	o	o
AES-256-GCM-12	x	o	o	o	o
AES-256-GCM-16	x	o	o	o	o
AES-128-CTR	x	o	o	o	o
AES-192-CTR	x	o	o	o	o
AES-256-CTR	x	o	o	o	o
AES-128-CCM-8	x	o	o	o	o
AES-128-CCM-12	x	o	o	o	o
AES-128-CCM-16	x	o	o	o	o
AES-192-CCM-8	x	o	o	o	o
AES-192-CCM-12	x	o	o	o	o
AES-192-CCM-16	x	o	o	o	o
AES-256-CCM-8	x	o	o	o	o
AES-256-CCM-12	x	o	o	o	o
AES-256-CCM-16	x	o	o	o	o
DES-CBC	o	o	x	o	o
3DES-CBC	x	x	x	x	x
BLOWFISH-128-CBC	x	o	x	x	x
BLOWFISH-192-CBC	x	o	x	x	x
BLOWFISH-256-CBC	x	o	x	x	x
CAMELLIA-128-CBC	x	o	o	o	o
CAMELLIA-192-CBC	x	o	o	o	o
CAMELLIA-256-CBC	x	o	o	o	o
CAMELLIA-128-CCM-8	x	o	o	o	o
CAMELLIA-128-CCM-12	x	o	o	o	o
CAMELLIA-128-CCM-16	x	o	o	o	o
CAMELLIA-192-CCM-8	x	o	o	o	o
CAMELLIA-192-CCM-12	x	o	o	o	o
CAMELLIA-192-CCM-16	x	o	o	o	o
CAMELLIA-256-CCM-8	x	o	o	o	o
CAMELLIA-256-CCM-12	x	o	o	o	o
CAMELLIA-256-CCM-16	x	o	o	o	o
SERPENT-128-CBC	x	o	o	o	o
SERPENT-192-CBC	x	o	o	o	o
SERPENT-256-CBC	x	o	o	o	o
TWOFISH-128-CBC	x	o	o	o	o
TWOFISH-192-CBC	x	o	o	o	o
TWOFISH-256-CBC	x	o	o	o	o
CAST-128-CBC	x	o	x	o	o
chacha20poly1305	x	o	o	o	o

Tabelle 4: Unterstützte Algorithmen für Vertraulichkeit der IPsec-Implementierungen

Modus \ Software	strongSwan	Windows	Shrewsoft	NCP	bintec
MD5	x	o	x	x	x
SHA-1	x	x	x	o	x
SHA-256	x	x	o	x	x
SHA-384	x	x	o	x	x
SHA-512	x	o	o	x	x
SHA-256-96	x	x	o	o	o
AES-XCBC	x	o	o	o	o
AES-128-GMAC	x	o	o	o	o
AES-192-GMAC	x	o	o	o	o
AES-256-GMAC	x	o	o	o	o

Tabelle 5: Unterstützte Algorithmen für Authentizität der IPsec-Implementierungen

Modus \ Software	strongSwan	Windows	Shrewsoft	NCP	bintec
MODP-768	x	o	x	x	x
MODP-1024	x	x	x	x	x
MODP-1536	x	o	x	x	x
MODP-2048	x	o	x	x	x
MODP-3072	x	o	x	x	x
MODP-4096	x	o	x	x	x
MODP-6144	x	o	x	x	x
MODP-8192	x	o	x	x	x
MODP-1024s160	x	o	o	o	o
MODP-2048s224	x	o	o	o	o
MODP-2048s256	x	o	o	o	o
ECP-192	x	o	o	x	o
ECP-224	x	o	o	x	o
ECP-256	x	o	o	x	o
ECP-384	x	o	o	x	o
ECP-521	x	o	o	x	o
ECP-224BP	x	o	o	o	o
ECP-256BP	x	o	o	o	o
ECP-384BP	x	o	o	o	o
ECP-512BP	x	o	o	o	o
NTRU-112	x	o	o	o	o
NTRU-128	x	o	o	o	o
NTRU-192	x	o	o	o	o
NTRU-256	x	o	o	o	o
NEWHOPE-128	x	o	o	o	o

Tabelle 6: Unterstützte Schlüsselaustauschprotokolle der IPsec-Implementierungen

Code 16: Testkonfiguration - strongswan.conf

```
1 charon-svc {
2         filelog {
3                 this_log.txt{
4                         default=2
5                         job = 1
6                         mgr = 0
7                         enc = 0
8                         asn = 0
```

Modus \ Software	strongSwan	Windows	Shrewsoft	NCP	bintec
Hybrid	x	x	x	x	x
PSK	x	o	x	x	o
PSK + XAUTH	x	o	x	x	o
X.509	x	x	x	x	x
EAP-MD5	x	o	o	x	o
EAP-PAP	x	o	o	x	o
EAP-CHAP	x	o	o	x	o
EAP-MSCHAPv2	x	x	o	x	o
EAP-GTC	x	o	o	o	o
EAP-TLS	x	x	o	x	o
EAP-TTLS	x	o	o	o	o
EAP-AKA	x	o	o	o	o
EAP-TNC	x	o	o	o	o
TNC-IMC	x	o	o	o	o
TNC-IMV	x	o	o	o	o

Tabelle 7: Unterstützte Authentifizierungsmethoden der IPsec-Implementierungen

Modus \ Software	strongSwan	Windows	Shrewsoft	NCP	bintec
CRL	x	x	o	x	x
OCSP	x	?	o	x	x

Tabelle 8: Unterstützte Mechanismen zum Zurückziehen von Zertifikaten der IPsec-Implementierungen

Modus \ Software	strongSwan	Windows	Shrewsoft	NCP	bintec
Tunnel-Modus	x	x	x	x	x
Transport-Modus	x	x	o	o	o
BEET-Modus	x	o	o	o	o

Tabelle 9: Unterstützte Tunnel-Modi der IPsec-Implementierungen

Modus \ Software	strongSwan	Windows	Shrewsoft	NCP	bintec
Main Mode	x	x	x	x	?
Aggressive Mode	x	x	x	x	?
Quick Mode	x	o	x	x	?
Config Mode	x	x	x	x	x

Tabelle 10: Unterstützte IKE-Modi der IPsec-Implementierungen

```
9                               flush_line = yes
10                              ike_name = yes
11                              append = no
12                      }
13              }
14      }
```

Code 17: Code von win32.h

```
1 /*
```

Feature \ Software	strongSwan	Windows	Shrewsoft	NCP	bintec
NAT-T	x	x	x	x	x
DPD	x	x	x	x	x
MOBIKE	x	x	o	o	o
GUI	o	x	x	x	x
IPsec über TCP	o	o	o	x	x
IPv6	x	x	x	x	x
Attribut-Zertifikate	x	?	o	o	?
PFS	x	o	x	x	x
IKE-Fragmentierung	x	Nur IKEv1	x	o	o
Komprimierung	x	o	o	x	o

Tabelle 11: Unterstützte Features der IPsec-Implementierungen

```
2  * Copyright (C) 2016 Noel Kuntze
3  *
4  * Permission is hereby granted, free of charge, to any person obtaining a copy
5  * of this software and associated documentation files (the "Software"), to deal
6  * in the Software without restriction, including without limitation the rights
7  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
8  * copies of the Software, and to permit persons to whom the Software is
9  * furnished to do so, subject to the following conditions:
10 *
11 * The above copyright notice and this permission notice shall be included in
12 * all copies or substantial portions of the Software.
13 *
14 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
15 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
16 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
17 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
18 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
19 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
20 * THE SOFTWARE.
21 */
22
23 #ifndef WIN32_H
24 #define WIN32_H
25
26 #define WIN32_TUN_READ_EVENT_TEMPLATE "WIN32-libipsec-read-device-%d"
27 #define WIN32_TUN_WRITE_EVENT_TEMPLATE "WIN32-libipsec-write-device-%d"
28 #define WIN32_TUN_EVENT_LENGTH 80
29 #define TAP_WIN_COMPONENT_ID "tap0901"
30
31 #define ADAPTER_KEY
       "SYSTEM\\CurrentControlSet\\Control\\Class\\{4D36E972-E325-11CE-BFC1-08002BE10318}"
32 #define NETWORK_CONNECTIONS_KEY
       "SYSTEM\\CurrentControlSet\\Control\\Network\\{4D36E972-E325-11CE-BFC1-08002BE10318}"
33
34 /*
35  * ==========================
36  * Filesystem prefixes
37  * ==========================
38  */
```

```
39
40 #define USERMODEDEVICEDIR "\\\\.\\Global\\"
41 #define SYSDEVICEDIR      "\\Device\\"
42 #define USERDEVICEDIR     "\\DosDevices\\Global\\"
43 #define TAP_WIN_SUFFIX    ".tap"
44
45 /*
46  * TAP IOCTL constants and macros.
47  *
48  */
49 #define TAP_WIN_CONTROL_CODE(request,method) \
50   CTL_CODE (FILE_DEVICE_UNKNOWN, request, method, FILE_ANY_ACCESS)
51
52 /* Present in 8.1 */
53
54 #define TAP_WIN_IOCTL_GET_MAC               TAP_WIN_CONTROL_CODE (1,
      METHOD_BUFFERED)
55 #define TAP_WIN_IOCTL_GET_VERSION           TAP_WIN_CONTROL_CODE (2,
      METHOD_BUFFERED)
56 #define TAP_WIN_IOCTL_GET_MTU               TAP_WIN_CONTROL_CODE (3,
      METHOD_BUFFERED)
57 #define TAP_WIN_IOCTL_GET_INFO              TAP_WIN_CONTROL_CODE (4,
      METHOD_BUFFERED)
58 #define TAP_WIN_IOCTL_CONFIG_POINT_TO_POINT TAP_WIN_CONTROL_CODE (5,
      METHOD_BUFFERED)
59 #define TAP_WIN_IOCTL_SET_MEDIA_STATUS      TAP_WIN_CONTROL_CODE (6,
      METHOD_BUFFERED)
60 #define TAP_WIN_IOCTL_CONFIG_DHCP_MASQ      TAP_WIN_CONTROL_CODE (7,
      METHOD_BUFFERED)
61 #define TAP_WIN_IOCTL_GET_LOG_LINE          TAP_WIN_CONTROL_CODE (8,
      METHOD_BUFFERED)
62 #define TAP_WIN_IOCTL_CONFIG_DHCP_SET_OPT   TAP_WIN_CONTROL_CODE (9,
      METHOD_BUFFERED)
63
64 /* Added in 8.2 */
65
66 /* obsoletes TAP_WIN_IOCTL_CONFIG_POINT_TO_POINT */
67 #define TAP_WIN_IOCTL_CONFIG_TUN            TAP_WIN_CONTROL_CODE (10,
      METHOD_BUFFERED)
68 #define TAP_WIN_IOCTL_CONFIG_SET_SRC_CHECK  TAP_WIN_CONTROL_CODE (11,
      METHOD_BUFFERED)
69
70
71 #endif /* WIN32_H */
```

Code 18: Code für das Suchen eines TAP-Geräts

```
1 /*
2  * Searches through the registry for suitable TAP driver interfaces
3  * On Windows, the TAP interface metadata is stored and described in the
       registry.
4  * It returns a linked list that contains all found guids. The guids describe
       the interfaces.
5  */
6
```

```
7  linked_list_t *get_tap_reg()
8  {
9      HKEY adapter_key;
10     LONG status;
11     DWORD len;
12     linked_list_t *list = linked_list_create();
13     int i = 0;
14
15     /*
16      * Open parent key. It contains all other keys that
17      * describe any possible interfaces.
18      */
19     status = RegOpenKeyEx(
20             HKEY_LOCAL_MACHINE,
21             ADAPTER_KEY,
22             0,
23             KEY_READ,
24             &adapter_key);
25
26     if (status != ERROR_SUCCESS)
27     {
28         DBG2(DBG_LIB, "Error opening registry key: %s", ADAPTER_KEY);
29     }
30
31     while (true)
32     {
33         char enum_name[256];
34         char unit_string[256];
35         HKEY unit_key;
36         char component_id_string[] = "ComponentId";
37         char component_id[256];
38         char net_cfg_instance_id_string[] = "NetCfgInstanceId";
39         char net_cfg_instance_id[256];
40         DWORD data_type;
41
42         len = sizeof (enum_name);
43         status = RegEnumKeyEx(
44                 adapter_key,
45                 i,
46                 enum_name,
47                 &len,
48                 NULL,
49                 NULL,
50                 NULL,
51                 NULL);
52         if (status == ERROR_NO_MORE_ITEMS)
53         {
54             break;
55         }
56         else if (status != ERROR_SUCCESS)
57         {
58             DBG2(DBG_LIB, "Error enumerating registry subkeys of key: %s",
59                     ADAPTER_KEY);
60         }
61
```

```
62          snprintf(unit_string, sizeof (unit_string), "%s\\%s",
63                  ADAPTER_KEY, enum_name);
64
65          status = RegOpenKeyEx(
66                  HKEY_LOCAL_MACHINE,
67                  unit_string,
68                  0,
69                  KEY_READ,
70                  &unit_key);
71
72          if (status != ERROR_SUCCESS)
73          {
74              DBG2(DBG_LIB, "Error opening registry key: %s", unit_string);
75          }
76          else
77          {
78              len = sizeof (component_id);
79              status = RegQueryValueEx(
80                      unit_key,
81                      component_id_string,
82                      NULL,
83                      &data_type,
84                      component_id,
85                      &len);
86
87              if (status != ERROR_SUCCESS || data_type != REG_SZ)
88              {
89                  DBG2(DBG_LIB, "Error opening registry key: %s\\%s",
90                          unit_string, component_id_string);
91              }
92              else
93              {
94                  len = sizeof (net_cfg_instance_id);
95                  status = RegQueryValueEx(
96                          unit_key,
97                          net_cfg_instance_id_string,
98                          NULL,
99                          &data_type,
100                         net_cfg_instance_id,
101                         &len);
102
103                 if (status == ERROR_SUCCESS && data_type == REG_SZ)
104                 {
105                     if (!strcmp(component_id, TAP_WIN_COMPONENT_ID))
106                     {
107                         /* That thing is a valid interface key */
108                         /* link into return list */
109                         char *guid = malloc(sizeof(net_cfg_instance_id));
110                         memcpy(guid, net_cfg_instance_id,
                                sizeof(net_cfg_instance_id));
111                         list->insert_last(list, guid);
112                     }
113                 }
114             }
115             RegCloseKey(unit_key);
```

```
116             }
117             ++i;
118         }
119
120         RegCloseKey(adapter_key);
121         return list;
122 }
```

Code 19: Code für handle_plain auf Windows

```
1  /**
2   * Job handling outbound plaintext packets
3   */
4  static job_requeue_t handle_plain(private_kernel_libipsec_router_t *this)
5  {
6  #ifdef WIN32
7          DBG2(DBG_ESP, "entered handle_plain.");
8          void **key = NULL;
9          bool oldstate;
10         uint32_t length, event_status = 0, i = 0, j = 0, offset;
11         handle_overlapped_buffer_t *bundle_array = NULL, dummy,
               tun_device_handle_overlapped_buffer, *structures = NULL;
12         OVERLAPPED *overlapped = NULL;
13         HANDLE *event_array = NULL, tun_device_event;
14         tun_device_t *tun_device = this->tun.tun;
15         enumerator_t *tuns_enumerator;
16
17         memset(&tun_device_handle_overlapped_buffer, 0,
               sizeof(handle_overlapped_buffer_t));
18         /* Reset synchronisation event */
19         ResetEvent(this->event);
20
21         length = this->tuns->get_count(this->tuns);
22
23         this->lock->read_lock(this->lock);
24         /* Read event for this->tun */
25
26         /* allocate arrays for all the structs we need */
27         /* events, overlapped structures and bundles. */
28         /* event_array holds all the HANDLE structures for the events that are
29          * used for notifying the thread of finished reads and writes.
30          */
31
32         overlapped = alloca((length+2)*sizeof(OVERLAPPED));
33         event_array = alloca((length+2)*sizeof(HANDLE));
34         bundle_array = alloca((length+2)*sizeof(handle_overlapped_buffer_t));
35
36         memset(overlapped, 0, (length+2)*sizeof(OVERLAPPED));
37         memset(bundle_array, 0, (length+2)*sizeof(handle_overlapped_buffer_t));
38
39         DBG2(DBG_ESP, "Allocated arrays, opened events");
40
41         /* These are the arrays we're going to work with */
42
43         /* first position is the event we use for synchronisation  */
```

```
44          /* Insert notification event */
45          event_array[i] = this->event;
46          DBG2(DBG_ESP, "put notification event into index %d", i);
47          /* Insert dummy structure */
48          bundle_array[i] = dummy;
49          i++;
50
51          /* second position is this->tun */
52          /* insert event object for this->tun device */
53          tun_device_event = CreateEvent(NULL, FALSE, FALSE,
                this->tun.tun->get_read_event_name(this->tun.tun));
54          if (!tun_device_event)
55          {
56              char *error_message = format_error(GetLastError());
57              DBG2(DBG_ESP, "Error: %s", error_message);
58              free(error_message);
59              return JOB_REQUEUE_FAIR;
60          }
61          event_array[i] = tun_device_event;
62          DBG2(DBG_ESP, "Put TUN %s event in index %d",
                this->tun.tun->get_name(this->tun.tun), i);
63          ResetEvent(event_array[i]);
64          /* bundle for the read on this->tun */
65          /* Reserve memory for the buffer*/
66          tun_device_handle_overlapped_buffer.buffer =
                chunk_alloc(tun_device->get_mtu(tun_device));
67
68          DBG2(DBG_ESP, "Allocated buffer.");
69          tun_device_handle_overlapped_buffer.fileHandle =
                tun_device->get_handle(tun_device);
70          DBG2(DBG_ESP, "Allocated file handle.");
71          tun_device_handle_overlapped_buffer.overlapped = overlapped;
72          DBG2(DBG_ESP, "Allocated overlapped..");
73          /* Fill in code */
74          /*
75          tun_device_handle_overlapped_buffer.overlapped->hEvent =
                OpenEvent(EVENT_ALL_ACCESS, FALSE,
                this->tun.tun->get_read_event_name(this->tun.tun));
76          */
77          tun_device_handle_overlapped_buffer.overlapped->hEvent= tun_device_event;
78          if (tun_device_handle_overlapped_buffer.overlapped->hEvent == NULL)
79          {
80              char *error_message = format_error(GetLastError());
81              DBG2(DBG_ESP, "Error: %s", error_message);
82              free(error_message);
83              return JOB_REQUEUE_FAIR;
84          }
85          DBG2(DBG_ESP, "Created event");
86          bundle_array[i] = tun_device_handle_overlapped_buffer;
87
88          i++;
89
90          /* Start ReadFile for this->tun.handle */
91          if (!start_read(&tun_device_handle_overlapped_buffer,
                tun_device_handle_overlapped_buffer.overlapped->hEvent))
```

```
92          {
93                  // TODO: Cleanup heap
94                  this->lock->unlock(this->lock);
95                  return JOB_REQUEUE_FAIR;
96          }
97          /* pad bundle_array with two empty structures */
98          /* iterate over all our tun devices, create event handles, reset them,
                queue read operations on all handles */
99
100         DBG2(DBG_ESP, "Enumerating tun devices ...");
101
102         tuns_enumerator = this->tuns->create_enumerator(this->tuns);
103         while(tuns_enumerator->enumerate(tuns_enumerator, key, &tun_device))
104         {
105             DBG2(DBG_ESP, "TUN device %s", *key);
106             /* Allocate structure and buffer */
107
108             structures[j].buffer = chunk_alloc(tun_device->get_mtu(tun_device));
109             memset(structures[j].buffer.ptr, 0, structures[j].buffer.len);
110             structures[j].fileHandle = tun_device->get_handle(tun_device);
111             /* Allocate and initialise OVERLAPPED structure */
112             structures[j].overlapped = alloca(sizeof(OVERLAPPED));
113             (*structures[j].overlapped) = overlapped[j];
114             memset(&structures[j].overlapped, 0, sizeof(OVERLAPPED));
115             /* Create unique name for that event. */
116             /* Create unique event for read accesses on that device
117              * No security attributes, no manual reset, initial state is
                    unsignaled,
118              * name is the special name we created
119              */
120             structures[j].overlapped->hEvent = CreateEvent(NULL, FALSE, FALSE,
                    tun_device->get_read_event_name(tun_device));
121             // event_array[i] = OpenEvent(EVENT_ALL_ACCESS, FALSE,
                    tun_device->get_read_event_name(tun_device));
122             event_array[i] = structures[j].overlapped->hEvent;
123             bundle_array[i] = structures[j];
124
125             if (event_array[i] == NULL)
126             {
127                 char *error_message = format_error(GetLastError());
128                 DBG2(DBG_ESP, "Error: %s", error_message);
129                 free(error_message);
130                 return JOB_REQUEUE_FAIR;
131             }
132             i++;
133
134             /* Initialise read with the allocate overwrite structure */
135             DBG2(DBG_ESP, "Reading on %s", tun_device->get_name(tun_device));
136             if (!start_read(&structures[j], structures[j].overlapped->hEvent))
137             {
138                     // TODO: Cleanup heap
139                     this->lock->unlock(this->lock);
140                     return JOB_REQUEUE_FAIR;
141             }
142             j++;
```

```
143         }
144         tuns_enumerator->destroy(tuns_enumerator);
145
146         this->lock->unlock(this->lock);
147         while (TRUE)
148         {
149             /* Wait for a handle to be signaled */
150             /* In the mingw64 sources, MAXIMUM_WAIT_OBJECTS is defined as 64.
                    That means we can wait for a maximum of 64 event handles.
151              * This translates to 63 tun devices. I think this is sufficiently
                    high to not have to implement a mechanism for waiting for more
152              * events /support more TUN devices */
153             oldstate = thread_cancelability(FALSE);
154             DBG2(DBG_ESP, "Waiting for events...");
155             event_status = WaitForMultipleObjects(i, event_array, FALSE,
                    INFINITE);
156             thread_cancelability(oldstate);
157             DBG2(DBG_ESP, "Event triggered with event_status %d", event_status);
158             offset = event_status - WAIT_OBJECT_0;
159             DBG2(DBG_ESP, "offset == %d", offset);
160
161             /* A handle was signaled. Find the tun handle whose read was
                    successful */
162
163             /* We can only use the event_status of indication for the first
                    completed IO operation.
164              * After the event was signaled, we need to test the OVERLAPPED
                    structure in the other array
165              * to find out what event was signaled.
166              */
167             /*
168              * Probably broken?
169              */
170             /* Check if an event in the array was signaled. (That is the case if
171              * the event_status is between WAIT_OBJECT_0 and WAIT_OBJECT_0 +
                    nCount -1)
172              */
173             if ((WAIT_OBJECT_0 < event_status) && event_status < ((WAIT_OBJECT_0
                    + length - 1)))
174             {
175                 /* the event at event_array[event_status - WAIT_OBJECT_0] has
                        been signaled */
176                 /* It is possible that more than one event was signalled. In
                        that case, (event_status - WAIT_OBJECT_0)
177                  * is the index with the lowest event that was signalled. More
                        signalled events can be found higher
178                  *
179                  * According to the documentation, WAIT_OBJECT_0 is defined as 0
180                  */
181                 if (offset == 0)
182                 {
183                     /* Notification about changes regarding the tun devices.
184                      * Or the object is destroyed.
185                      * We need to rebuild the array. So exit and rebuild. */
186                     DBG2(DBG_ESP, "cleanup.");
```

```
187                     /* Cleanup
188                      *  Starts with 1 to skip over the dummy
189                      */
190                     for(uint32_t k=1;k<i;k++)
191                     {
192                         /* stop all asynchronous IO */
193                         CancelIo(bundle_array[k].fileHandle);
194                         CloseHandle(bundle_array[k].overlapped->hEvent);
195                         memset(bundle_array[k].buffer.ptr, 0,
                                bundle_array[k].buffer.len);
196                         free(bundle_array[k].buffer.ptr);
197                         ResetEvent(event_array[k]);
198                         CloseHandle(event_array[k]);
199                     }
200                     /* exit */
201                     DBG2(DBG_ESP, "Cleanup done.");
202                     return JOB_REQUEUE_DIRECT;
203                 }
204                 for(uint32_t k=1;k<i; k++)
205                 {
206                     DBG2(DBG_ESP, "position %d in array", k);
207                     /* Is the object signaled? */
208                     DBG2(DBG_ESP, "checking if event is signaled.");
209                     DWORD WaitResult =
                            WaitForSingleObject(bundle_array[k].overlapped->hEvent,
                            0);
210                     DBG2(DBG_ESP, "WaitForSingleObject returned %d", WaitResult);
211                     if (WaitResult == WAIT_OBJECT_0)
212                     {
213                         /* The arrays have the same length and the same
                                positioning of the elements.
214                          * Therefore, if event_array[k] is signaled, the read on
                                bundle_array[i].fileHandle has succeeded
215                          * and bundle_array[k].buffer has our data now. */
216                         DBG2(DBG_ESP, "Event is signaled. Processing packet.");
217                         /* Do we need to copy the chunk before we enqueue it? */
218                         char *foo = alloca((bundle_array[k].buffer.len *4)/3 +
                                1);
219                         memset(foo, 0, (bundle_array[k].buffer.len *4)/3 + 1);
220                         DBG2(DBG_ESP, "Length of buffer: %u",
                                bundle_array[k].buffer.len);
221                         chunk_to_base64(bundle_array[k].buffer, foo);
222                         DBG2(DBG_ESP, "Content of Buffer: %s", foo);
223
224                         ip_packet_t *packet;
225                         /* clone the buffer */
226                         chunk_t buffer_clone = chunk_clone
                                (bundle_array[k].buffer);
227                         packet = ip_packet_create(buffer_clone);
228                         DBG2(DBG_ESP, "Packet contents: %B",
                                packet->get_encoding(packet));
229                         if (packet)
230                         {
231                                 ipsec->processor->queue_outbound(ipsec->processor,
                                        packet);
```

```
232                         }
233                         else
234                         {
235                                 DBG2(DBG_ESP, "invalid IP packet read from TUN
                                        device");
236                         }
237                         /* Reset the overlapped structure, event and buffer */
238                         /* Print out the package for debugging */
239                         memset(&bundle_array[k].overlapped, 0,
                                sizeof(OVERLAPPED));
240                         /* Don't leak packets */
241                         memset(bundle_array[k].buffer.ptr, 0,
                                bundle_array[k].buffer.len);
242
243                         bundle_array[k].overlapped->hEvent = event_array[k];
244
245                         if (!start_read(&bundle_array[k],
                                bundle_array[k].overlapped->hEvent))
246                         {
247                            /* Cleanup
248                             *  Starts with 1 to skip over the dummy
249                             */
250                             for(uint32_t k=1;k<i;k++)
251                             {
252                                 /* stop all asynchronous IO */
253                                 CancelIo(bundle_array[k].fileHandle);
254                                 CloseHandle(bundle_array[k].overlapped->hEvent);
255                                 memset(bundle_array[k].buffer.ptr, 0,
                                        bundle_array[k].buffer.len);
256                                 free(bundle_array[k].buffer.ptr);
257                                 ResetEvent(event_array[k]);
258                                 CloseHandle(event_array[k]);
259                             }
260                             this->lock->unlock(this->lock);
261                             return JOB_REQUEUE_FAIR;
262                         }
263                     }
264                     else
265                     {
266                         DBG2(DBG_ESP, "Event is not signaled.");
267                     }
268                 }
269
270             }
271             /* Function failed */
272             else
273             {
274                 DBG2(DBG_ESP, "waiting for events on the tun device reads
                        failed.");
275
276                 /* Cleanup
277                  *  Starts with 1 to skip over the dummy
278                  */
279                 for(uint32_t k=1;k<i;k++)
280                 {
```

```
281                         /* stop all asynchronous IO */
282                         CancelIo(bundle_array[k].fileHandle);
283                         CloseHandle(bundle_array[k].overlapped->hEvent);
284                         memset(bundle_array[k].buffer.ptr, 0,
                                bundle_array[k].buffer.len);
285                         free(bundle_array[k].buffer.ptr);
286                         ResetEvent(event_array[k]);
287                         CloseHandle(event_array[k]);
288                     }
289                     return JOB_REQUEUE_FAIR;
290
291                 }
292             }
293             return JOB_REQUEUE_DIRECT;
294             #else
295             [...]
296             #endif
297 }
```

Code 20: Debug-Log; Zeigt Problematik mit WaitForSingleObject()

```
 1 00[DMN] Starting IKE service charon-svc (strongSwan 5.4.1dr1, Windows Client
       6.2.9200 (SP 0.0)
 2 00[LIB] plugin 'sha3': loaded successfully
 3 00[LIB] plugin 'md4': loaded successfully
 4 00[LIB] plugin 'nonce': loaded successfully
 5 00[LIB] plugin 'x509': loaded successfully
 6 00[LIB] plugin 'pubkey': loaded successfully
 7 00[LIB] plugin 'pkcs1': loaded successfully
 8 00[LIB] plugin 'dnscert': loaded successfully
 9 00[LIB] plugin 'ipseckey': loaded successfully
10 00[LIB] plugin 'pem': loaded successfully
11 00[LIB] plugin 'openssl': loaded successfully
12 00[LIB] plugin 'files': loaded successfully
13 00[LIB] Error opening registry key:
       SYSTEM\CurrentControlSet\Control\Class\{4D36E972-E325-11CE-BFC1-08002BE10318}\Properties
14 00[LIB] TAP-Windows driver version 9.22 available.
15 00[LIB] created TUN device: {EDA0C976-3256-4B30-8A92-708D2F643E28}
16 00[LIB] plugin 'kernel-libipsec': loaded successfully
17 00[LIB] plugin 'kernel-wfp': loaded successfully
18 00[LIB] plugin 'kernel-iph': loaded successfully
19 00[LIB] plugin 'socket-win': loaded successfully
20 00[LIB] plugin 'vici': loaded successfully
21 00[LIB] plugin 'eap-identity': loaded successfully
22 00[LIB] plugin 'eap-gtc': loaded successfully
23 00[LIB] plugin 'eap-dynamic': loaded successfully
24 00[LIB] plugin 'eap-tls': loaded successfully
25 00[LIB] plugin 'eap-peap': loaded successfully
26 00[LIB] feature CUSTOM:kernel-ipsec in plugin 'kernel-wfp' failed to load
27 00[LIB] feature PUBKEY:DSA in plugin 'pem' has unmet dependency: PUBKEY:DSA
28 00[LIB] feature CUSTOM:dnscert in plugin 'dnscert' has unmet dependency: RESOLVER
29 00[LIB] feature CUSTOM:ipseckey in plugin 'ipseckey' has unmet dependency:
       RESOLVER
30 00[LIB] feature PRIVKEY:DSA in plugin 'pem' has unmet dependency: PRIVKEY:DSA
31 00[LIB] feature PRIVKEY:BLISS in plugin 'pem' has unmet dependency: PRIVKEY:BLISS
```

```
32 00[LIB] feature CERT_DECODE:PGP in plugin 'pem' has unmet dependency:
       CERT_DECODE:PGP
33 00[LIB] feature CERT_DECODE:OCSP_REQUEST in plugin 'pem' has unmet dependency:
       CERT_DECODE:OCSP_REQUEST
34 00[LIB] unloading plugin 'dnscert' without loaded features
35 00[LIB] unloading plugin 'ipseckey' without loaded features
36 00[LIB] unloading plugin 'kernel-wfp' without loaded features
37 00[LIB] loaded plugins: charon-svc sha3 md4 nonce x509 pubkey pkcs1 pem openssl
       files kernel-libipsec kernel-iph socket-win vici eap-identity eap-gtc
       eap-dynamic eap-tls eap-peap
38 00[LIB] unable to load 8 plugin features (7 due to unmet dependencies)
39 00[JOB] spawning 16 worker threads
40 00[LIB] created thread 4016
41 00[LIB] created thread 3032
42 00[LIB] created thread 3216
43 00[LIB] created thread 3876
44 00[LIB] created thread 3132
45 00[LIB] created thread 1580
46 00[LIB] created thread 1244
47 00[LIB] created thread 4212
48 00[LIB] created thread 3152
49 00[LIB] created thread 4208
50 00[LIB] created thread 2560
51 00[LIB] created thread 2764
52 00[LIB] created thread 1604
53 00[LIB] created thread 4220
54 00[LIB] created thread 1304
55 00[LIB] created thread 972
56 08[ESP] entered handle_plain.
57 08[ESP] Allocated arrays, opened events
58 08[ESP] put notification event into index 0
59 08[ESP] Put TUN {EDA0C976-3256-4B30-8A92-708D2F643E28} event in index 1
60 08[ESP] Allocated buffer.
61 08[ESP] Allocated file handle.
62 08[ESP] Allocated overlapped..
63 08[ESP] Created event
64 08[ESP] ReadFile() returned 0
65 08[ESP] Error 997
66 08[ESP] Read on tun device is pending.
67 08[ESP] Enumerating tun devices ...
68 08[ESP] Waiting for events...
69 08[ESP] Event triggered with event_status 1
70 08[ESP] offset == 1
71 08[ESP] position 1 in array
72 08[ESP] checking if event is signaled.
73 08[ESP] WaitForSingleObject returned 258
74 08[ESP] Event is not signaled.
75 08[ESP] Waiting for events...
76 [...]
```

Code 21: Ausgabe von ipconfig und route -4 print

```
1 C:\Users\Noel\bin>ipconfig
2
3 Windows-IP-Konfiguration
```

```
4
5
6  Ethernet-Adapter THIS IS A TAP DEVICE:
7
8     Verbindungsspezifisches DNS-Suffix:
9     Verbindungslokale IPv6-Adresse  . : fe80::596b:bf92:963f:63a3%8
10    IPv4-Adresse  . . . . . . . . . . : 172.16.20.2
11    Subnetzmaske  . . . . . . . . . . : 255.255.255.255
12    Standardgateway . . . . . . . . . :
13
14 Ethernet-Adapter Ethernet:
15
16    Verbindungsspezifisches DNS-Suffix: thermicorp.lan
17    IPv6-Adresse. . . . . . . . . . . : 2a02:8071:9282:e600:3031:9c6b:6485:3cc9
18    Temporäre IPv6-Adresse. . . . . . : 2a02:8071:9282:e600:5181:db51:f4d7:2afa
19    Temporäre IPv6-Adresse. . . . . . : 2a02:8071:9282:e600:7154:ac74:30c4:cbee
20    Verbindungslokale IPv6-Adresse  . : fe80::3031:9c6b:6485:3cc9%3
21    IPv4-Adresse  . . . . . . . . . . : 192.168.178.218
22    Subnetzmaske  . . . . . . . . . . : 255.255.255.0
23    Standardgateway . . . . . . . . . : fe80::a96:d7ff:fe85:e002%3
24                                        192.168.178.1
25
26 Tunneladapter isatap.{EDA0C976-3256-4B30-8A92-708D2F643E28}:
27
28    Medienstatus. . . . . . . . . . . : Medium getrennt
29    Verbindungsspezifisches DNS-Suffix:
30
31 Tunneladapter Teredo Tunneling Pseudo-Interface:
32
33    Medienstatus. . . . . . . . . . . : Medium getrennt
34    Verbindungsspezifisches DNS-Suffix:
35
36 Tunneladapter isatap.thermicorp.lan:
37
38    Medienstatus. . . . . . . . . . . : Medium getrennt
39    Verbindungsspezifisches DNS-Suffix: thermicorp.lan
40
41
42 C:\Users\Noel\bin>route -4 print
43 ===========================================================================
44 Schnittstellenliste
45   8...00 ff ed a0 c9 76 ......TAP-Windows Adapter V9
46   3...08 00 27 ef 0b 0e ......Intel(R) PRO/1000 MT-Desktopadapter
47   1...........................Software Loopback Interface 1
48   4...00 00 00 00 00 00 00 e0 Microsoft-ISATAP-Adapter
49   5...00 00 00 00 00 00 00 e0 Teredo Tunneling Pseudo-Interface
50   6...00 00 00 00 00 00 00 e0 Microsoft-ISATAP-Adapter #2
51 ===========================================================================
52
53 IPv4-Routentabelle
54 ===========================================================================
55 Aktive Routen:
56       Netzwerkziel    Netzwerkmaske          Gateway     Schnittstelle Metrik
57            0.0.0.0          0.0.0.0    192.168.178.1  192.168.178.218     10
58          127.0.0.0        255.0.0.0    Auf Verbindung         127.0.0.1    306
```

```
59          127.0.0.1   255.255.255.255    Auf Verbindung          127.0.0.1     306
60    127.255.255.255   255.255.255.255    Auf Verbindung          127.0.0.1     306
61       172.16.20.2    255.255.255.255    Auf Verbindung       172.16.20.2      276
62       172.16.25.2    255.255.255.255  169.254.128.128        172.16.20.2       30
63     192.168.178.0     255.255.255.0     Auf Verbindung    192.168.178.218     266
64    192.168.178.218   255.255.255.255    Auf Verbindung    192.168.178.218     266
65    192.168.178.255   255.255.255.255    Auf Verbindung    192.168.178.218     266
66          224.0.0.0         240.0.0.0    Auf Verbindung          127.0.0.1     306
67          224.0.0.0         240.0.0.0    Auf Verbindung       172.16.20.2      276
68          224.0.0.0         240.0.0.0    Auf Verbindung    192.168.178.218     266
69    255.255.255.255   255.255.255.255    Auf Verbindung          127.0.0.1     306
70    255.255.255.255   255.255.255.255    Auf Verbindung       172.16.20.2      276
71    255.255.255.255   255.255.255.255    Auf Verbindung    192.168.178.218     266
72  ===========================================================================
73  Ständige Routen:
74    Keine
```

6.3 Lizensierung der Arbeit

Diese BA steht unter der GNU General Public License version 3 (GPLv3) und darf unter Berücksichtigung der Lizensvereinbarung der GPLv3 verfielfältigt und verteilt werden. Der Lizenztext ist auf der offiziellen Webseite[35] zu finden.

Das Logo der Hochschule Offenburg (HSO) steht unter seiner eigenen Lizenz. Es steht nicht unter der GPLv3. Der gesamte Code steht unter der URL `https://github.com/Thermi/Bachelorarbeit` zur Verfügung.
Diese Arbeit wurde mittels LaTeXerstellt

[35] `https://www.gnu.org/licenses/gpl.html`

Eidesstattliche Erklärung

Hiermit versichere ich eidesstattlich, dass die vorliegende Bachelor-Thesis von mir selbstständig und ohne unerlaubte fremde Hilfe angefertigt worden ist, insbesondere, dass ich alle Stellen, die wörtlich oder annähernd wörtlich oder dem Gedanken nach aus Veröffentlichungen, unveröffentlichten Unterlagen und Gesprächen entnommen worden sind, als solche an den entsprechenden Stellen innerhalb der Arbeit durch Zitate kenntlich gemacht habe, wobei in den Zitaten jeweils der Umfang der entnommenen Originalzitate kenntlich gemacht wurde. Ich bin mir bewusst, dass eine falsche Versicherung rechtliche Folgen haben wird.

Haslach, den ....

Noel Kuntze