Studienarbeit Informatik

SwanTamer
Das GUI für strongSwan

Thomas Boksberger
Marco Gruber

Wintersemester 2006/2007
1. Februar 2007

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

Inhaltsverzeichnis

1 Abstract

Abteilung	Informatik
Name der Studenten	Marco Gruber, Thomas Boksberger
Semester	WS 06/07
Titel der Studienarbeit	GUI für Linux strongSwan
Examinator	Prof. Dr. Andreas Steffen

1.1 Ziel der Arbeit

StrongSwan, eine populäre IPsec Implementation wird üblicherweise mit einer ASCII Datei (/etc/ipsec.conf) konfiguriert. Die Benutzer von Linux-Installationen werden immer jünger und unerfahrener. Sie konfigurieren ihr System immer mehr mit den von der Distribution zur Verfügung gestellten grafischen Tools, und nicht mehr über die Komandozeile und die Konfigurationsdateien. Um auch diesen Benutzern den einfachen Zugang zu strongSwan zu ermöglichen, soll im Rahmen dieser Semesterarbeit ein einfach zu bedienendes Grafisches Konfigurations-Tool erstellt werden.

1.2 Ergebnisse

Im Laufe dieser Semesterarbeit ist ein Tool (SwanTamer) entstanden, das es ermöglicht, einfache Verbindungen von Rechnern und Netzen zu realisieren. Dabei können die grundlegenden Eigenschaften eingestellt werden.

Wenn eine komplexere Konfigurationsdatei existiert und diese mit SwanTamer eingelesen wird, können die grundlegenden Einstellungen verändert werden, die komplexeren werden ohne Veränderung wieder in die Datei geschrieben.

Ein Benutzer, der die grundlegenden Funktionsweisen von IPsec verstanden hat, ist mit SwanTamer in der Lage eine Verbindung zwischen verschiedenen Rechnern/Netzen zu erstellen und zu betreiben.

1.3 Ausblick

Eine denkbare Erweiterung zum SwanTamer wäre eine integrierte Zertifikatsverwaltung mit der Möglichkeit solche auch zu erstellen und zu verteilen.

Ebenfalls wäre es möglich SwanTamer so zu erweitern, dass auch komplexere Konfigurationen im Tool machbar und editierbar wären.

2 Management Summary

2.1 Ausgangslage

2.1.1 Motivation

Die Benutzer von Linux-Installationen werden immer jünger und unerfahrener. Sie konfigurieren ihr System immer mehr mit den von der Distribution zur Verfügung gestellten Grafischen Tools. Dabei sind sie immer seltener in der Lage die Dienste und Programme über die ursprünglichen Konfigurationsdateien zu konfigurieren. SwanTamer soll genau diesen Leuten einen möglichst einfachen Zugang zu strongSwan geben.

2.1.2 Vergleichbare Arbeiten

Es gibt schon einige Programme die helfen eine VPN Verbindung aufzubauen. Die meisten davon arbeiten aber mit andern Programmen zusammen und sind nicht in der Lage direkt mit strongSwan zu kommunizieren.

2.1.3 Ziele

Wir haben dieses Projekt ausgewählt, weil wir eine Herausforderung in der Programmiersprache und den dazugehörigen Bibliotheken sahen, und weil uns die Thematik von IPsec interessiert.

SwanTamer muss in der Lage sein einfache Verbindungen zu konfigurieren, und diese zu starten. Wichtig ist dabei Stabilität, auf keinen Fall darf SwanTamer abstürzen.

2.2 Vorgehen

2.2.1 Planung und Arbeit

Um einen Überblick zu behalten und das Programm in seine logischen Teilbereiche zu unterteilen, haben wir SwanTamer in einer Dreischichtenarchitektur organisiert. Dabei haben wir uns zusammen von unten nach oben gearbeitet. Das heisst:

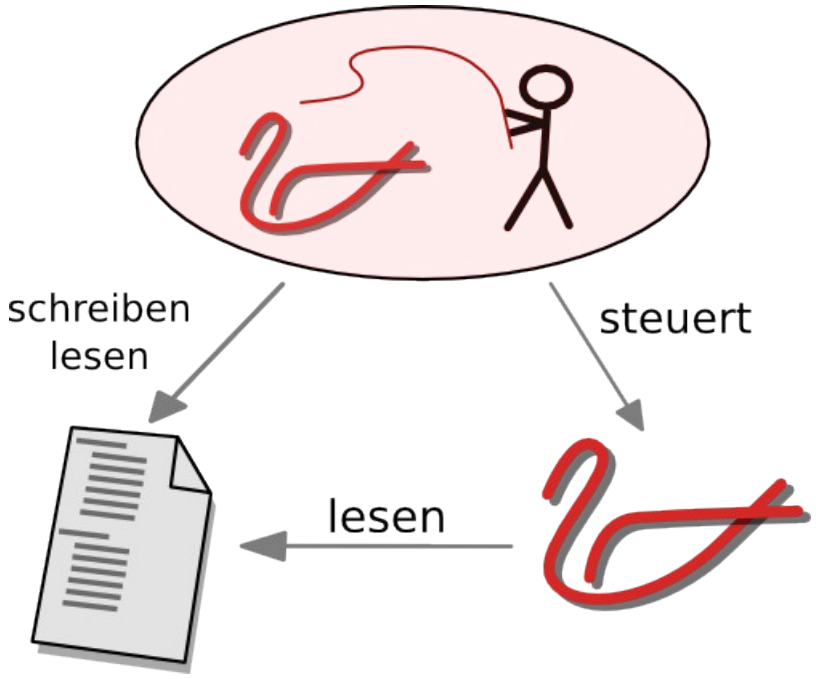
- Am Anfang haben wir uns darum gekümmert, dass SwanTamer die Konfigurationsdatei lesen und schreiben kann. (DH)
- In einem weiteren Schritt haben wir SwanTamer beigebracht die Konfiguration zu verstehen. (PD)
- Zum Schluss haben wir SwanTamer ein Aussehen gegeben. (UI)

2.2.2 Involvierte Personen

Betreuer:	Prof. Dr. Andreas Steffen
	Martin Willi
Studenten:	Marco Gruber
	Thomas Boksberger

2.3 Ergebnis

SwanTamer ist eine neue eigenständige Applikation. Diese liest und schreibt die Konfigurationsdatei. Zusätzlich wird mit strongSwan kommuniziert, um sich mit den einzelnen VPN Endpunkten zu verbinden oder zu trennen. Es wird sichergestellt, dass strongSwan auch nach dem Editieren der Konfiguration ohne SwanTamer problemlos funktioniert. Dies erreichen wir, indem wir die Konfiguration aus der normalen Datei mit der gleichen Syntax auslesen und keine zusätzliche Datei erzeugen.



2.3.1 Zielereichung

Bis auf den optionalen Punkt der interaktiven Passwortabfrage für einen geschützten Key, wurden in SwanTamer alle in der Aufgabenstellung beschriebenen Punkte implementiert.

2.3.2 Lernpunkte

Während dieser Arbeit haben wir uns einerseits mit neuen Bibliotheken (GTKmm, Glibmm) auseinander gesetzt, andererseits haben wir unser theoretisches Wissen über die Programmiersprache C++ in der Praxis einsetzen können.

2.4 Ausblicke

Eine denkbare Erweiterung zum SwanTamer wäre eine integrierte Zertifikatsverwaltung mit der Möglichkeit solche auch zu erstellen und zu verteilen.

Ebenfalls wäre es möglich SwanTamer so zu erweitern, dass auch komplexere Konfigurationen im Tool machbar und editierbar wären.

3 Auftrag

3.1 Einführung

Die populäre Linux strongSwan IPsec Software (www.strongswan.org) wird traditionell über die ASCII-Dateien /etc/ipsec.conf und /etc/ipsec.secrets konfiguriert. Um die Benutzerfreundlichkeit bei VPN Desktop-Anwendungen zu erhöhen, soll ein einfach-zu-bedienendes, grafisches Konfigurationstool auf der Basis der GTK+ Grafikbibliothek realisiert werden. Im Rahmen der Studienarbeit soll zuerst ein Konzept erarbeitet und anschliessend ein typischer VPN Remote-Access Konfigurationsablauf praktisch implementiert werden.

3.2 Aufgabenstellung

- Vertiefung in die Funktionsweise der strongSwan Software, insbesondere die Konfiguration via ipsec.conf und ipsec.secrets. Studium der typischen VPN Szenarien.
- Erarbeitung eines Konzepts für eine ergonomische, grafische Benutzerführung. Im Minimum solllen folgende Features unterstützt werden:
  - IKEv1 und IKEv2 Key Exchange Modus
  - IPsec Tunnel Modus (ESP)
  - NAT Traversal immer aktiviert
  - IKE und ESP Verschlüsselungsalgorithmen: AES-128/192/256, 3DES
  - IKE und ESP Hashalgorithmen: SHA-1, SHA-256, MD5
  - Perfect Forward Secrecy (PFS): yes|no
  - Host-to-Host, Host-Net, Net-to-Net und Net-to-Host Tunnel
  - IPv4 und IPv6 Adressen (Gateways und Subnetze)
  - IPv4 oder IPv6 Virtuelle IP Adresse: Konfiguration statisch oder via ModeConfig
  - RSA Authentisierung mit einem X.509 Zertifikat
  - PSK Authentisierung mit einem Pre-Shared Key
  - Eigene und Peer Identität: Distinguished Name, Hostname oder Email-Adresse
  - Aktivierung der Dead-Peer-Detection: none|clear|hold|restart
  - Starten und Stoppen von strongSwan via ipsec start|stop
  - Aufdatieren der Konfiguration via ipsec update
  - Starten und Stoppen von Verbindungen via ipsec up|down
  - Statusanzeige (rot/gelb/grün) auf der Basis von ipsec status
- Optional soll der RSA Private Key über eine Passphrase geschützt werden können, die interaktive abgefragt werden kann.
- Konfigurationsänderungen sollen nach ipsec.conf, respektive ipsec.secrets zurückgeschrieben werden.

4 Projektplan

Wir haben uns während des Projekts nur eine Skizze gemacht, auf der wir unsere Meilensteine definiert haben. Diese Skizze hatte allerdings während der ganzen Semesterarbeit auch nur den Charakter einer Skizze, an die wir uns mehr oder weniger

Arbeit \Woche	1	2	3	4	5	6	7	8	WF	9	10	11	12	13	14
Technologiestudium															
Datenhaltung															
Problemdomain															
Userinterface															
Dokumentieren															
Sitzung															
Summe	32,5	32,7	39,4	33	32,9	35	27	21	46	28	26,1	25,3	30,4	50,1	25

Wenig:
Mittel:
Viel:

orientierten.

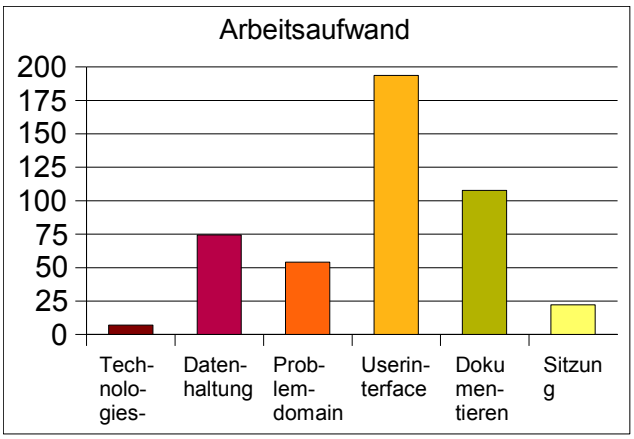
Der Oben dargestellte Fahrplan, ist der „Ist“ Zustand. Auf diesem Fahrplan ist deutlich eine Arbeitsweise von oben nach unten zu sehen.

Dieser Fahrplan entspricht ziemlich genau unseren Vorstellungen. Wie erwartet, haben wir für die Problemdomain nicht viel Zeit benötigt. Für das User Interface allerdings umso mehr.

In Woche 7 und 8 hatten wir ein Problem mit Speicherlecks & double Free's. Dies hat uns in diesen beiden Wochen viel Zeit gekostet. Schlussendlich haben wir dann auf Smartpointer umgestellt. So ist der „Ausrutscher“ mit dem Mittleren Zeitaufwand in der Datenhaltung zu erklären.

Am Schluss der Arbeit in Woche 13, haben wir festgestellt, dass wir einen Punkt der Projektanforderung noch nicht implementiert hatten. So mussten wir in dieser Woche noch einmal quer durch den Code (von der DH bis zum UI), um dieses Feature zu realisieren.

4.1 Zeiterfassung



Eine detaillierte Zeiterfassung von Thomas Boksberger und Marco Gruber ist in den entsprechenden OpenOffice Calc Dateien zu finden.

5 Konzept

Dieses GUI sollte Leute ansprechen, welche sich weder mit Linux/Unix noch mit IPsec besonders gut auskennen. Es soll eine Hilfe bei der Konfiguration und dem Starten bzw. Stoppen der einzelnen Tunnel sein. Da es nicht für den HighEnd User gedacht ist, hält sich der Funktionsumfang auch in Grenzen, damit die weniger versierten User nicht vor lauter Funktionsumfang den Überblick verlieren.

Eine weitere wichtige Eigenschaft sollte sein, dass die Datenhaltung bei Änderungen der Syntax des Konfigurationsfiles einfach anzupassen ist. Besonders bezüglich den zwei Konzepten wie die Konfiguration in strongSwan gehandhabt werden.

Eine Connection besteht ist im „pluto“ aus einem Objekt. Dieses entspricht dem Connection Block in der heutigen Konfigurationsdatei. Anders ist dies im „charon“. Dieser liest zwar von der gleichen Konfigurationsdatei teilt den Connection Block jedoch in eine Connection und mehrere Policies auf. Möglich wäre nun, dass mehrere Policies in einer Connection existieren (was zurzeit von der Konfigurationsdatei nicht unterstützt wird).

Unser Ziel ist es hier, intern bereits mehrere Policies zu unterstützen. Dies wollen wir zurzeit jedoch nicht bis ins GUI übernehmen. Der Grund hierfür ist, dass dieses Tool für User gedacht ist, für die das editieren der Konfigurationsdatei zu schwierig erscheint. In unseren Augen ist der Connection/Policy Ansatz komplizierter und somit für unsere Zielpersonen schwieriger zu verstehen.

6 Designentscheide

6.1 Schichten Modellierung

6.1.1 Kontext

Die Klassen, die für das Lesen / Schreiben von ipsec.conf ipsec.secrets und für das Starten und Stoppen der Verbindungen zuständig sind, sollten einfach austauschbar sein. So kann in einer späteren Version von strongSwan auch ein XML basierte Konfigurationsdatei problemlos eingelesen werden.

Die Eigenschaften (Key, Value Paare) sollten direkt in verschiedenen Enums oder ganzen Klassen abgelegt werden. Diese Klassen sind sehr zentral und werden von überall her (User Interface, Datenhaltung) verwendet. Es sollte also ein Design erstellt werden, dass diese Klassen ausserhalb der Datenhaltung (also die Lese-/Schreibklassen) definiert.

6.1.2 Variante Datenhaltung unterteilen

Die Datenhaltung wird in zwei Teile unterteilt. Ein Teil Lesen / Schreiben / Parsen und in ein Teil, welcher die Konfigurationsdatei repräsentiert und somit die Key/Value Paare dieser Datei beinhaltet. Einlesen und Parsen kann von der Problemdomain bequem auf die eingelesenen Einträge zugreifen. Da diese als Objekte abgelegt sind, lässt sich der Wert ganz einfach mit Methodenaufrufe verändern.

Da die Datenhaltung weder Connection, Policies oder andere Problemdomain Konstrukte kennen sollte, muss hier eine Neutrale Struktur für die Problemdomain angeboten werden.

Vorteil: Die Datenhaltung kann alle ihre Aufgaben wahrnehmen (Lesen, Schreiben und Parsen) und die Problemdomain kann bequem auf diese Daten zugreifen.

6.1.3 Variante Parsen in der Problemdomain

Die Datenhaltung liest die Konfigurationsdatei ein und erstellt daraus einen Baum. Jeder Knoten ist ein Key, Value Paar (beides Strings). Diese Strings werden dann erst in der PD validiert und in Klassen und Enums umgewandelt.

Vorteil: Die Software ist „schön geschichtet“. Das heisst die Datenhaltung greift nicht auf die Problemdomain zu und übergibt was sie einliesst. Weiterhin lassen sich die Informationen direkt in die Problemdomain Klassen (Connection, Policy, ...) laden und müssen nicht in einer neutralen, bereits geparsten Form übergeben werden.

Nachteil: Die Problemdomain muss sich um das Parsen kümmern, was eigentlich nicht ihre Aufgabe ist.

6.1.4 Entscheid

Weil wir keine Nachteile gefunden haben, entschlossen wir uns die Datenhaltung zu unterteilen.

6.2 Umgang mit „Policies“ von Charon

6.2.1 Kontext

In der Konfigurationsdatei werden für die Definition einer Verbindung die „conn“ Blöcke verwendet. In Pluto entspricht dies auch dem internen Aufbau.

In IKEv2 bzw. Charon wird das nicht so gelöst. Dort beinhaltet eine Connection mindestens eine Policy. Dabei wird in der Connection definiert, wohin man eine VPN Verbindung aufbaut. Die Policy entspricht einem Filter, welcher definiert was durch diese Connection geschickt wird.

Die Frage ist, wie diese Daten in SwanTamer abgespeichert werden. Mit anderen Worten, ob wir nur mit einer Connections oder mit Policies und Connections arbeiten.

6.2.2 Variante nur Connections

Wie im Kontext beschrieben, würden wir mit dieser Variante keine Trennung von diesen Informationen machen.

Vorteil: Der Vorteil dabei ist, dass die interne Struktur eins zu eins der Konfigurationsdatei entspricht. Somit muss weder beim Einlesen noch beim Abspeichern der Konfiguration ein Aufwand für ein solches Mapping betrieben werden.

Nachteil: Ein Nachteil ist, dass der IKEv1 Teil – also der Pluto Daemon – irgendwann verschwinden wird (zumindest hofft man das). Ein Refactoring um die interne Struktur auf eine Connection/Policy Struktur zu ändern, würde einiges mehr Aufwand generieren, als wenn man dies zur jetzigen zeit machen würde.

6.2.3 Variante Connection und Policy

Das Gegenteil von der obigen Variante wäre, wenn wir die interne Struktur von Charon (IKEv2 Daemon) nachbilden.

Dies würde bedeuten, dass die Konfigurationsdatei beim Einlesen in diese Connection / Policy Struktur umgewandelt werden muss. Beim Schreiben ist natürlich ein umgekehrter Mechanismus nötig.

Vorteil: Der Vorteil von dieser Variante ist natürlich, dass sie „Zukunftssicherer“ ist, da sie dem IKEv2 Modell entspricht.

Nachteil: Ein Nachteil ist sicherlich der Mehraufwand. Sofern diese Variante jedoch erst später umgesetzt wird, würde dies einiges an Mehraufwand bedeuten.

Weiterhin besteht ein Problem beim Schreiben der Policies. Um die Policies in Format der aktuellen Konfigurationsdatei abzuspeichern muss man mehre „conn“ Blöcke erstellen. Beim Einlesen einer solchen Konfigurationsdatei sind jedoch die Informationen der Zusammengehörigkeit der Policies nicht mehr vorhanden. Man müsste dann zusätzliche Logik implementieren, welche alle Connection Key-Wörter in den „conn“ Blöcken vergleicht und diese entsprechend nach gleichen Connections zusammenfasst.

6.2.4 Entscheid

Bei einem der wöchentlichen Meetings mit den Betreuern haben wir entschieden, dass die interne Struktur auf der zweiten Variante (Connection und Policies) beruhen soll.

Jedoch soll das GUI selbst zurzeit nicht die Trennung Connection und Policies machen. Somit erübrigt sich auch das Problem des Einlesens und Vergleichens der „conn“ Blöcke um die Policies zusammenzufassen.

6.3 Kommentare in der Konfigurationsdatei

6.3.1 Beschreibung

In der Manualpage von strongSwan (5 ipsec.conf v1.1) findet man folgendes:

> The file is a text file, consisting of one or more *sections*. White space followed by **#** followed by anything to the end of the line is a comment and is ignored, as are empty lines which are not within a section.

Das bedeutet, dass ein Kommentar mit einem Leerzeichen gefolgt von einem # beginnt. Üblicherweise unterscheidet man als User das Leerzeichen und den Tabulator nicht. Deshalb macht es Sinn, diese Kombination auch zu unterstützen.

6.4 Fehlerhafter Configeintrag

6.4.1 Beschreibung

Eine momentane Konfigurationsdatei sieht folgendermassen aus:

```
config setup
        plutostart=no

conn %default
        ikelifetime=60m
        keylife=20m
        rekeymargin=3m
        keyingtries=1
        keyexchange=ikev2
        authby=secret

conn home
        left=192.168.0.100
        leftnexthop=%direct
        leftid=carol@strongswan.org
        leftfirewall=yes
        right=192.168.0.1
        rightid=@moon.strongswan.org
        rightsubnet=10.1.0.0/16
        auto=add

conn rw
        leftsubnet=10.2.0.0/16
        right=%any
        auto=add
```

Dabei ist stets definiert, womit eine Sektion beginnen muss (nicht eingerückte Teile). Weiterhin ist definiert, welche Keys in welchen Sektionen gültig sind. Es stellt sich nun die Frage, wie das Programm Keywörter/Values bzw. Sektionen handhaben soll und welche

Paare ungültig sind (ob falsche Schlüsselwörter oder in einer falschen Sektion).

6.4.2 Variante ignorieren

Der configReader ignoriert solche Einträge. Dabei werden falsche Sektionen komplett ignoriert, bei falschen Keywörtern/Values wird diese Zeile entfernt.

Vorteil: Die weitergegebenen Daten der Datenhaltung enthalten nur validierte Informationen. Somit muss in der PD nicht darauf geachtet werden, dass diese falsche Informationen enthält.

Nachteil: Sofern zukünftig weitere Keywörter hinzukommen, müssen diese auch in unserem Programm in die Liste hinzugefügt werden, damit diese schlussendlich wieder in der veränderten Konfigurationsdatei landen.

6.4.3 Variante Daten einlesen

Der configReader versucht jegliche Daten einzulesen. Er validiert weder die Sectiontypen noch die Keywörter/Values.

Vorteil: Die Datenhaltung wird viel einfacher, da sie keine Validierung vornimmt. Weiterhin können diese Daten wieder in ein verändertes Konfigurationsfile geschrieben werden.

Nachteil: Die Validierung muss nun die Problemdomain machen. Diese kann jedoch nur Daten validieren, welche sie kennt bzw. für diejenigen Schlüsselwörter für welche eine entsprechende Validierung implementiert wurde. Daten welche sie nicht kennt, werden ignoriert (nicht entfernt) und können am Schluss wieder in die Konfigurationsdatei geschrieben werden. Dabei können fehlerhafte Konfigurationsdateien entstehen.

6.4.4 Variante Einlesen abbrechen

Sobald ein ungültiges Keyword/Value Paar bzw. ein ungültiger Sectiontyp auftaucht, wird das Einlesen mit einer Fehlermeldung beendet.

Vorteil: Im configReader müssen keine Eventualitäten speziell behandelt werden. Bei einem Fehler beendet er sich einfach. Dies ergibt die einfachste Implementierung von allen.

Nachteil: Nicht sehr User friendly

6.4.5 Entscheid

Entschieden haben wir uns für die Variante mit dem Ignorieren der Einträge. Der configReader muss bei einem ignorierten Eintrag eine Ausgabe auf dem StandardError machen. Weiterhin muss er eine Abfragemöglichkeit bieten, ob er Einträge beim Einlesen ignorieren musste.

6.5 SmartPointer

Aufgrund von Problemen mit Memory leaks und Zugriffen auf nicht mehr reservierten Speicher haben wir uns für den Einsatz von SmartPointern entschieden.

Nachfolgend werden die einzelne Implementationen solcher SmartPointer vorgestellt, um

schlussendlich den für uns Besten auszuwählen.

6.5.1 Variante „auto_ptr“

Da auto_ptr Bestandteil von Standard C++ Library ist, ist dieser auf allen Systemen verfügbar und wir generieren keine zusätzliche Abhängigkeit einer weiteren Library. Jedoch findet man folgendes unter http://www.cppreference.com/cppmisc/auto_ptr.html:

> **Warning**: It is generally a **bad idea** to put auto_ptr objects inside C++ STL containers. C++ containers can do funny things with the data inside them, including frequent reallocation (when being copied, for instance). Since calling the destructor of an auto_ptr object will free up the memory associated with that object, any C++ container reallocation will cause any auto_ptr objects to become invalid.

Da die Datenhaltung an einigen Orten mit Vektoren und Listen der STL arbeitet, fällt diese Möglichkeit weg.

6.5.2 Variante Boost's „shared_ptr“

Dieser Container ist für das Objekt im SmartPointer Objekt völlig transparent. Auch der Zugriff mit dem „->“ Operator geschieht wie bis an hin.

Die Boost Library ist jedoch nicht Bestandteil einer Standard C++ Installation und müsste somit nachinstalliert werden. Man hört jedoch immer mal wieder, dass die Boost Library in den nächsten C++ Standard mit einfliessen wird. Wie lange es geht, bis Boost wirklich Bestandteil von ANSI C++ ist, kann man jedoch nicht sagen und von daher müssen wir dies als eine normale zusätzliche Library ansehen.

6.5.3 Variante Glib's „RefPtr“

Auch diese Implementation eines SmartPointers befindet sich in einer Externen Library. Da wir jedoch GTK (welches Glib voraussetzt) für die GUI Oberfläche nutzen, hätten wir mit dieser Variante keine zusätzlich benötigte Library.

Der Nachteil dieser Variante ist, dass das Objekt in diesem Container selbst für den Referenzcounter bzw. das Löschen von sich selbst zuständig ist. Dieses Problem lässt sich jedoch leicht mit Hilfe einer Vererbung lösen. Da C++ Mehrfachvererbung unterstützt (im Gegensatz zu vielen anderen Sprachen), fallen Probleme mit Klassen, welche bereits von anderen Klassen geerbt haben weg.

6.5.4 Entscheid

Wir haben uns für den SmartPointer von Glib entschieden (RefPtr), da wir nicht wollen, dass unser Programm von 1000 anderen Libraries abhängig ist.

Wie oben beschrieben, ist der Nachteil der zusätzlichen Logik, welche eine solche Klasse mitbringen muss einfach in den Griff zu bekommen. Diese Logik ist bei uns in der

„smartPointerHelper“ Klasse implementiert. Sofern eine Klasse von dieser Klasse erbt, kann sie ohne weitere Modifikationen mit dem SmartPointer verwendet werden.

6.6 GTK Einbindung

6.6.1 Variante Glade generiert C

Das im Glade Designer entworfene User Interface wird aus dem Designer immer als C Code extrahiert.

Vorteil: Es sind uns keine Vorteile bekannt.

Nachteil: SwanTamer ist als C++ Applikation geplant. Daher ist es etwas umständlich immer wieder auf C Funktionen zurückzugreifen.

6.6.2 Variante Glade generiert C++

Das im Glade Designer entworfene User Interface wird aus dem Designer immer als C++ Code extrahiert. Dabei wird für jedes Window eine Klasse erzeugt.

Vorteil: Glade nimmt einem einige Arbeit ab, so muss man anschliessend weniger Code schreiben.

Nachteil: Galde überschreibt die generierten Klassen immer wieder, was es schwierig macht, diese zu erweitern. Man müsste für jede Klasse eine Wrapperklasse schreiben, die die generierte Klasse verwaltet.

6.6.3 Variante Glade generiert XML

Das im Glade Designer entworfene User Interface wird aus dem Designer als XML Datei gespeichert. Dieses kann mit Hilfe der libGlademm zur Laufzeit eingelesen und interpretiert werden.

Vorteil: Da das XML File zur Laufzeit gelesen wird, muss bei einer Änderung am User Interface der Quellcode nicht neu übersetzt werden. Zudem ist es die gängige Art unter Gnome, GUIs auf diese Weise zu erstellen. Das schönste an dieser Variante ist, dass man das Aussehen nicht im Quellcode definieren muss.

Nachteil: Das XML File muss zur Laufzeit eingelesen werden. Das heisst, es muss irgendwie im System gefunden werden.

6.6.4 Entscheid

Wir haben uns für die Variante 3 entschieden. Einerseits weil es die gängige Art ist, wie man Gnome GUI's baut. Andererseits weil uns die Variante zwei zu mühsam erschien.

6.7 Threads

Da wir mehrere Threads benötigen, stellt sich die Frage, wie sichergestellt wird, dass nicht beide gleichzeitig auf die gleichen GTK Daten zugreifen.

6.7.1 Variante „Eigner Mutex“

Mit einzelnen Glib::Mutex werden die gewünschten Bereiche von Hand abgesichert. Mit anderen Worten, für das ganze locking ist der Programmierer selbst verantwortlich und es werden keine eventuell vorhandenen Möglichkeiten von GTK benutzt.

Leider funktioniert dieses Verfahren nur, wenn man nur einen Globalen Mutex nutzt. Bereits bei einer Verwendung von einem Mutex pro Fenster, hatten wir bei unseren Tests immer wieder mal Abstürze beobachtet. Es scheint, als ob GTK intern gewisse Abhängigkeiten zwischen einzelnen Fenstern besitzt und somit die Veränderung eines Fensters (z.B. Update von Tabelleninhalten) Probleme bereitet.

Da diese Variante scheinbar nicht funktioniert ist sie bereits bei den Tests ausgeschieden.

6.7.2 Variante „GTK's gdk_threads_enter()/gdk_threads_leave()“

Wie unter http://developer.gnome.org/... beschrieben, gibt es für das Locking in GTK die zwei Methoden gdk_threads_enter() bzw. gdk_threads_leave(). Etwas Entsprechendes wird von GTKmm nicht zu Verfügung gestellt.

Nichtsdestotrotz lassen sich diese Funktionen auch in einer C++ Umgebung nutzen.

Das Konzept dahinter ist ein globaler Mutex, welcher vor jeder Veränderung gelockt werden muss.

6.7.3 Variante „Signale“

Signale sind eine völlig andere Herangehensweise als die beiden obigen Vorschläge.

Das Konzept dahinter ist, dass der „Main Thread” alle GUI Änderungen erledigt. „Worker Threads“ holen die Daten und senden dem „Main Thread” ein Signal um die Anzeige zu aktualisieren.

Ein Nachteil dabei ist, dass beim Verarbeiten eines Signals der „Main Thread“ beschäftigt ist. Während dieser Zeit ist somit keine Bedienung des gesamten User Interfaces möglich.

Es wird davon ausgegangen, dass solche Signale nicht in Massen an diesen „Main Thread“ geschickt werden. Weiterhin ist darauf zu achten, dass im „Main Thread“ nur das Nötigste gemacht wird. Im „Worker Thread“ muss all die restliche Arbeit getan werden, so dass der „Main Thread“ das User Interface nicht zulange blockiert.

6.7.4 Entscheid

Aufgrund der Einfachheit und schnellen Implementation, wurde die Variante mit den Signalen gewählt.

Da mehrere Threads erst nach zwei Drittel der Projektzeit auftauchten, müsste man für GTK Variante mit einem grossen Zeitaufwand für das Updaten des vorhandenen Codes rechnen.

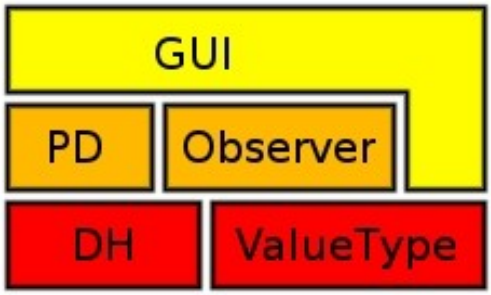
Inzwischen hat sich gezeigt, dass es nur in seltenen Fällen zum Blockieren des GUI's aufgrund dieses Entscheides führt. Diese einzelnen Fälle lassen sich zudem mit geringem Aufwand beheben.

Ein weiterer Punkt ist, dass dies auch die Standard Variante von Mircrosoft/WinForms und

Java/Swing ist.

7 Aufbau

Um die Konfigurationsdatei bestmöglich abstrahieren zu können haben wir uns für eine drei Schichten Architektur entschieden. In dieser Architektur ist die Datenhaltung (DH) für das Lesen, bzw. Schreiben der Konfigurationsdatei verantwortlich. Die Problemdomain (PD) repräsentiert die logische Darstellung der Konfiguration. Und das User Interface (UI oder GUI) zeigt die Fenster am Bildschirm an. Um diese Architektur auch im Code zu implementieren haben wir diese Schichten in Namensräume aufgeteilt.



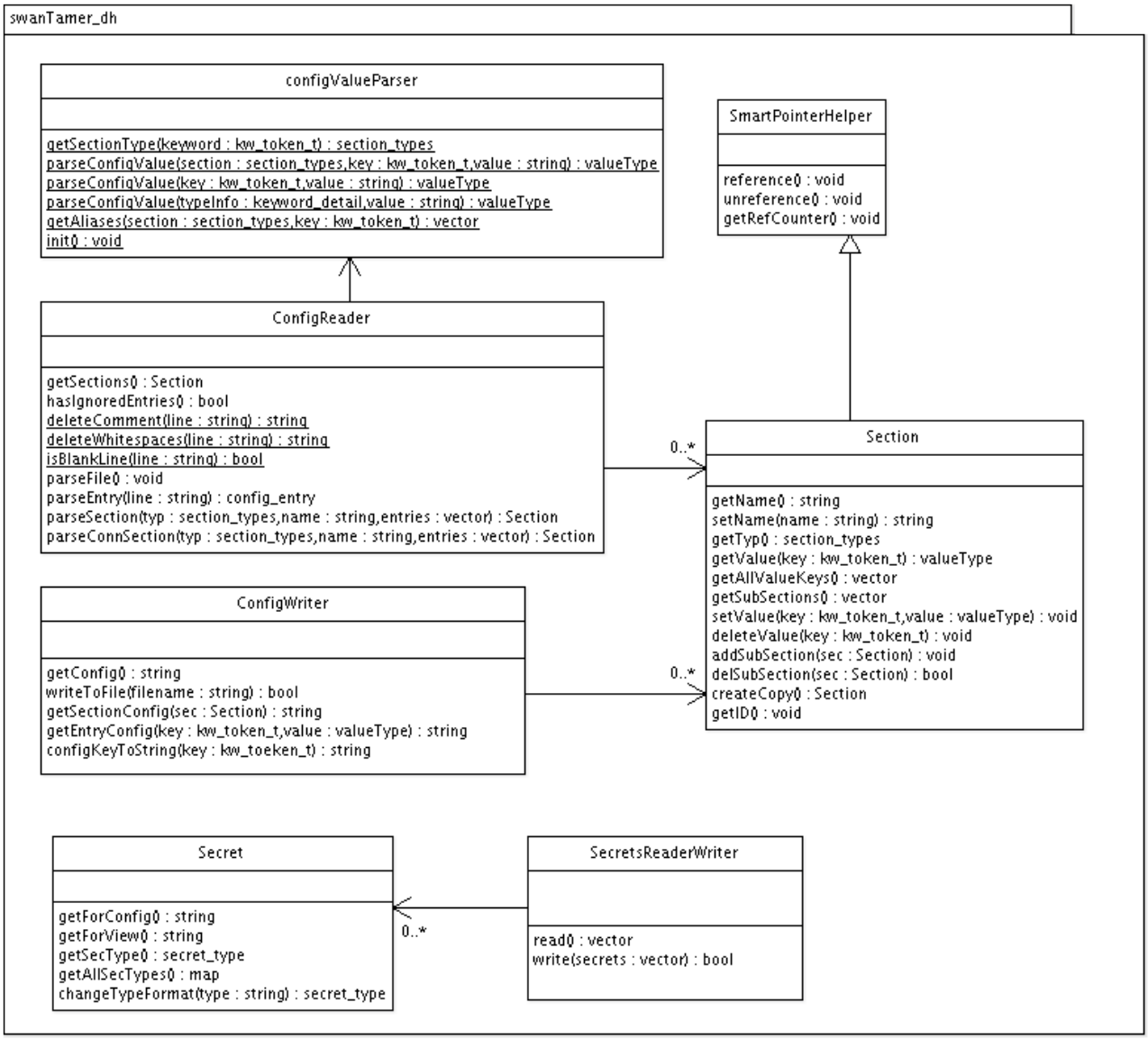
Der einzige Ort an dem das Schichtenmodell nicht eingehalten wird, ist das GUI. Dieses arbeitet stark mit den ValueTypes zusammen resp. verwendet diese. Das ist zwingend notwendig, da wir im GUI auch die Syntax validieren und so die genauen Typen brauchen. Dasselbe wird auch in der Datenhaltung gemacht, beim Einlesen der Konfiguration. Um keine Abhängigkeiten nach oben zu haben, sondern nur nach unten, haben wir beschlossen den ValueType in der untersten Schicht bei der DH zu platzieren.

7.1 Verzeichnisstruktur

src	Enthält nur die main.cpp Datei, welche die main() Funktion beinhaltet und somit für das Starten der Applikation zuständig ist
src/dh	Beinhaltet die Klassen, welche für das Ein-, Auslesen und Verwalten der Datenhaltung zuständig sind. In diesem Verzeichnis findet man die Klassen des Namespaces „swanTamer_dh“.
src/dh/valueTypes	Beinhaltet die Klassen, welche die internen Datentypen für die Schlüsselwörter repräsentieren. In diesem Verzeichnis findet man die Klassen des Namespaces „swanTamer_dh_valueTypes“.
src/pd	In diesem Verzeichnis findet man die Klassen des Namespaces „swanTamer_pd“.
src/pd/opserver	In diesem Verzeichnis findet man die Klassen des Namespaces „sswanTamer_pd_observer“.
src/ui	In diesem Verzeichnis findet man die Klassen des Namespaces „swanTamer_ui“.
src/strongswan	Enthält kopierte Dateien vom strongSwan Projekt welche von SwanTamer benötigt werden.

7.2 Namespace „swanTamer_dh“

Die Datenhaltung ist in zwei Namespaces aufgeteilt. Der eine Teil (genauere Beschreibung weiter unten) beinhaltet die ValueTypes. Der andere Teil (welcher nachfolgend beschrieben wird) enthält die restlichen Klassen, welche zur Datenhaltung gehören.



7.2.1 Section

Das wichtigste Element in diesem Namensraum ist die „Section“. Diese wird dazu verwendet, die eingelesenen Konfigurationsdaten an die PD weiterzugeben.

Der Aufbau einer solchen Section ist dabei wie der Knoten eines Baumes, welcher auf weitere Knoten (Sections) verweist und Key/Value Paare beinhaltet. Der Rekursion ist hierbei keine Grenze gesetzt, jedoch wird es momentan nur bis zur ersten Rekursion genutzt. Ein Blatt, welches man häufig bei Bäumen antrifft, verwenden wir hier nicht. Die Rekursion ist am Ende, wenn auf keine zusätzlichen Knoten verwiesen wird.

Eine Section in der Datenhaltung wird in der Problemdomain durch eine PDSection abgebildet. Das heisst, dass eine Connection und eine Policy welche von PDSection erben, immer eine Section aus der Datenhaltung verwalten.

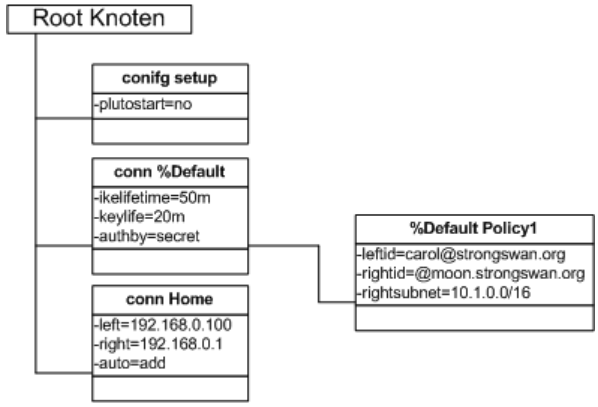
Der Root Knoten stellt zurzeit immer eine Liste von Connections („conn" Block und nicht Connection Klasse der PD), CA oder die „globalen Konfigurationseigenschaften" („config setup" Block) dar. Zurzeit existieren einzig in Connections Verweise auf weitere Sections. Diese stellen in der PD mögliche Policies oder andere Sections, welche zu dieser Connection gehören, dar. Es ist zwar möglich, unlogische Bäume wie etwa eine „Connection in einer Connection" oder eine „Policy in einer CA" anhand der Section Definition zu erstellen, jedoch wird dies beim Einlesen der Konfigurationsdatei abgefangen.

Die nachfolgende Abbildung einer strongSwan Konfiguration wird in der Datenhaltung eingelesen. Dabei wird diese in den daneben abgebildeten Baum abgebildet. Dabei entsprechen die Blöcke mit den Key/Value Paaren einem Section Objekt.

```
config setup
        plutostart=no

conn %default
        ikelifetime=60m
        keylife=20m
        authby=secret

conn home
        left=192.168.0.100
        leftid=carol@strongswan.org
        right=192.168.0.1
        rightid=@moon.strongswan.org
        rightsubnet=10.1.0.0/16
        auto=add
```



7.2.2 Keywords

Intern wird in SwanTamer mit den gleichen Schlüsselwörtern wie in strongSwan gearbeitet. Somit entsprechen sie einem „kw_token_t" Typ, welcher in der Datei „src/strongswan/keywords.h" definiert ist.

Das Mapping vom Keyword als String in diesen kw_token_t Datentypen geschieht also, wie auch in strongSwan selbst, mit den Files „keywords.*" im src/strongswan Verzeichnis. Bei neuen Keywords können diese vom strongSwan Projekt kopiert werden.

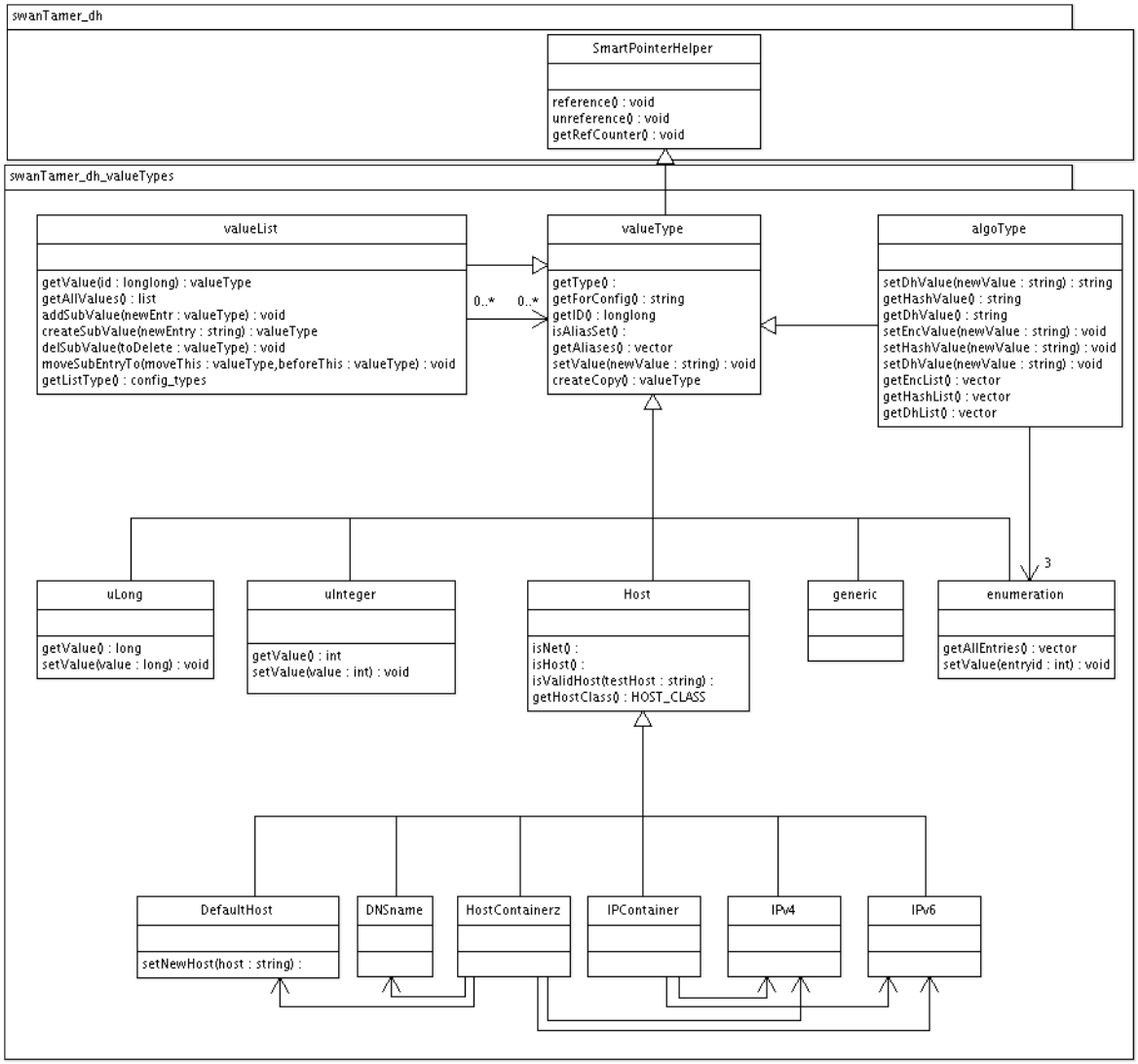
Das Mapping vom kw_token_t auf den entsprechenden valueType geschieht in der Klasse `swanTamer_dh::conigValueParser`. Hier müssen bei neuen Keywords entsprechende Einträge in die entsprechenden Vektoren eingetragen werden, damit ein Keyword einem Datentypen zugeordnet werden kann. Schlüsselwörter, welche nicht in diesen Vektoren definiert sind, werden nicht automatisch als String interpretiert. Es wird somit ein Fehler ausgelöst.

7.3 Namespace „swanTamer_dh_valueTypes"

Die Klasse swanTamer_dh_valueTypes::valueType stellt eine Abstrakte Klasse dar. Über den ValueType können die Werte der Konfigurationsdatei angesprochen und geändert werden. Für jede Art von Wert (also eine Zahl, Ipadresse, String, ja/nein, ...) gibt es einen entsprechenden ValueType der den Wert verwaltet.

Aufgrund von C++ Eigenheiten, lässt sich jedoch ein Objekt von valueType erstellen. Grundsätzlich sollten jedoch nie Objekte, welche nicht via Downcast auf valueType gecastet wurden, existieren.

Den nötigen Typ für einen Upcast erhält man mit der getType() Funktion, welche ein Keyword (dieses Keyword hat nichts mit der Konfiguration zu tun, sondern ist ein „swanTamer_dh_valueTypes::config_types“) zurückgibt, welches einem implementierten Typ von valueType entspricht.



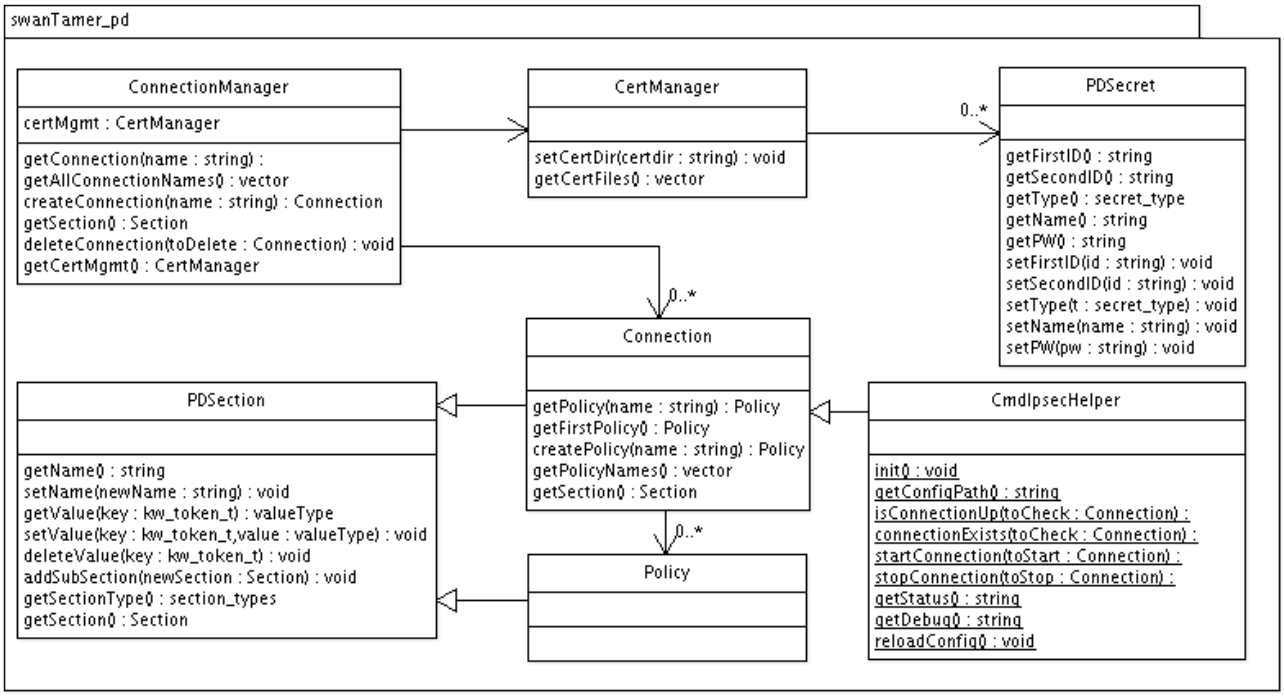
7.4 Namespace „swanTamer_pd“

Die Aufteilung von Connection und Policy ist im Pluto nicht bekannt. Charon hingegen arbeitet immer mit den Policies. Die Konfigurationsdatei ist aber für beide die selbe. In Anlehnung an den neueren Dämon Charon, ist die Problemdomain nach dessen Vorgehen gestaltet. Charon ist grundsätzlich in der Lage einer Connection mehrere Policies hinzuzufügen. Die Konfigurationsdatei kann ein solches Konstrukt allerdings nicht darstellen.

Darum haben wir das Design so gestaltet, dass wir grundsätzlich das neue Konzept von

Charon übernehmen. Das heisst eine Connection kann „im Prinzip“ beliebig viele Policies verwalten. Im Prinzip darum, weil im Userinterface zurzeit davon ausgegangen wird, dass nur eine Policy vorhanden ist.

Um mehrere Policies in einer Connection zu definieren müsste die Konfigurationsdatei umgestellt werden. Weiterhin müsste das GUI den neuen Ansprüchen angepasst werden.



7.4.1 Kommunikation mit strongSwan

Für die Kommunikation mit strongSwan ist die Klasse swanTamer_pd::CmdIpsecHelper zuständig. Sie stellt alle nötigen Funktionen zur Kommunikation mit strongSwan zu Verfügung.

Zur Zeit wird mit strongSwan via Kommandozeile bzw. system()-calls kommuniziert. Dabei ist sicherzustellen, dass "ipsec", "grep", "which" in Verzeichnissen, auf welche die PATH variable zeigt, zu finden sind.

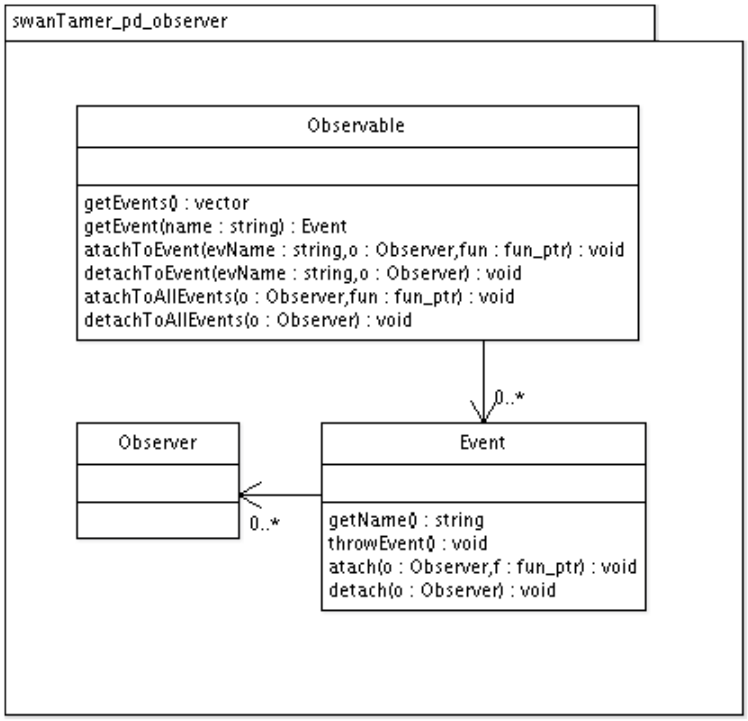
Der Grund, dass "ipsec" auf der Kommandozeile benötigt wird, ist auch der Grund, warum SwanTamer zurzeit mit Root Rechten gestartet werden muss.

In Zukunft könnte man sich vorstellen, dass eine direkte Kommunikation zwischen den strongSwan Daemons und SwanTamer stattfindet und somit nicht über die system()-calls kommuniziert werden muss.

7.5 Namespace „swanTamer_pd_observer“

Leider haben wir kein passendes Observer Paket unter C++ gefunden. Auch GTK scheint nichts Derartiges im Angebot zu haben. Daher haben wir schnell eine eigene Observer Implementierung programmiert. Dieser ist eine Mischung zwischen dem Observer Pattern (das auch in Java implementiert ist) und der Art wie C# das Problem löst. So muss der

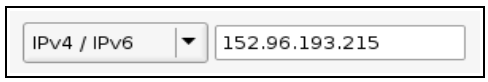
Observer einen parameterlosen Funktionspointer beim Observable registrieren. Dieser wird dann beim Auftreten des Events ausgeführt. Ein Observable kann dabei verschiedene Events haben, auf die individuell reagiert werden kann.



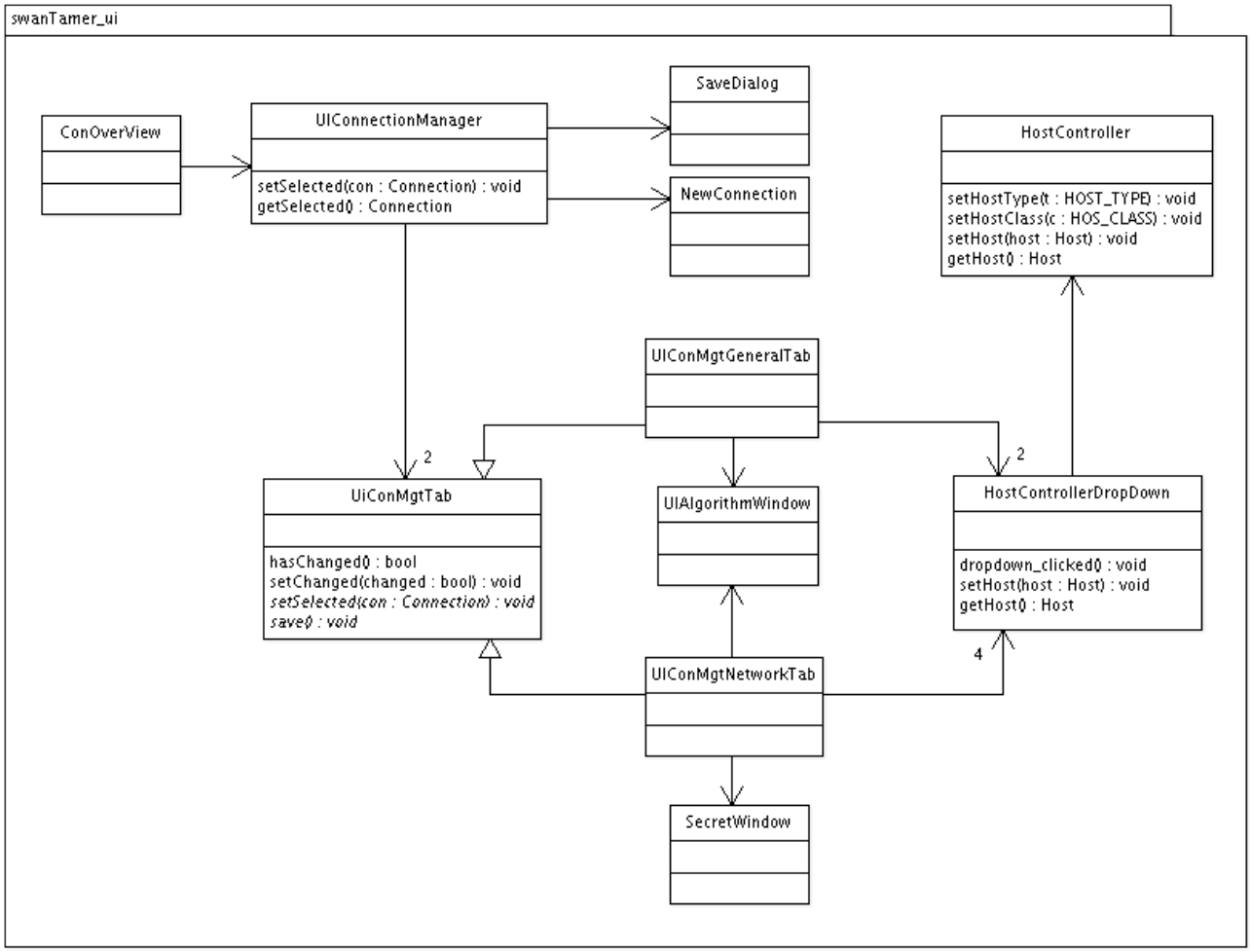
7.6 Namespace „swanTamer_ui“

Grundsätzlich verwaltet immer eine Klasse ein Fenster (ein Gtk::Window). Die Klasse UIConnectionManager (Konfigurationsfenster) ist dabei ein Spezialfall. Aufgrund der Tabs welche dieses Fenster beinhaltet, würde diese Klasse so gross und somit unübersichtlich, dass es nicht mehr angenehm gewesen wäre. Somit werden diese beiden Tabs von je einer eigenen Klasse verwaltet der UIConMgtGeneralTab und UIConMgtNetworkTab.

Selbstverständlich wurden einzelne Elemente, welche an mehreren Orten verwendet werden, aus diesen Fenster-Klassen in eigene Klassen exportiert. So konnte sinnloses copy & paste vermieden werden.



Ein Beispiel hierfür ist der HostControllerDropDown, welcher ein Dropdown-/Input-Box Paar verwaltet. Dieses Paar wird z.B. für die Eingabe von Ipv4 oder Hostnamen verwendet. Das Textfeld wird dabei übersichtshalber mit der Klasse HostController verwaltet.



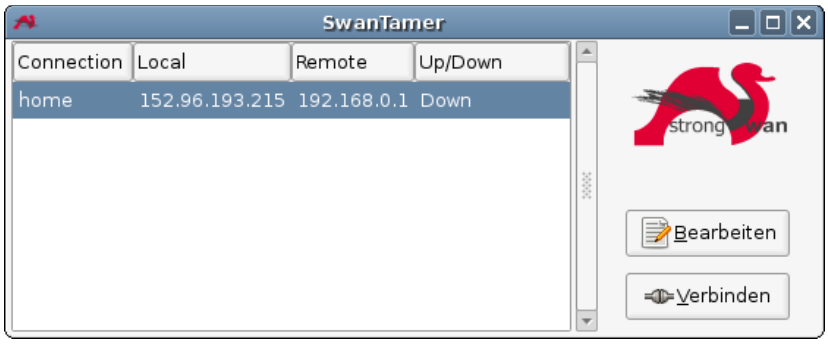
7.6.1 Secret - Manager

Der Secret – Manager kann im momentanen Stand leider nur PSK „Schlüssel“ verwalten. Die Zertifikate können im UIConnectionManager ausgewählt werden. Konfigurieren kann man sie aber nicht. Diese müssen von Hand in die vorgesehenen Verzeichnisse kopiert werden, und im ipsec.secrets eingetragen werden.

Dies haben wir zusammen mit Herr Steffen beschlossen. Da die vollständige Verwaltung der Zertifikate und Schlüssel ein Parsen dieser Dateien bedeutet hätte. Dies würde aber den Rahmen und die Zeitvorgaben dieser Arbeit sprengen.

7.7 Das Userinterface

7.7.1 Hauptfenster



Dieses Fenster ist das Hertz der Applikation. Es ist das Erste Fenster das geöffnet wird.

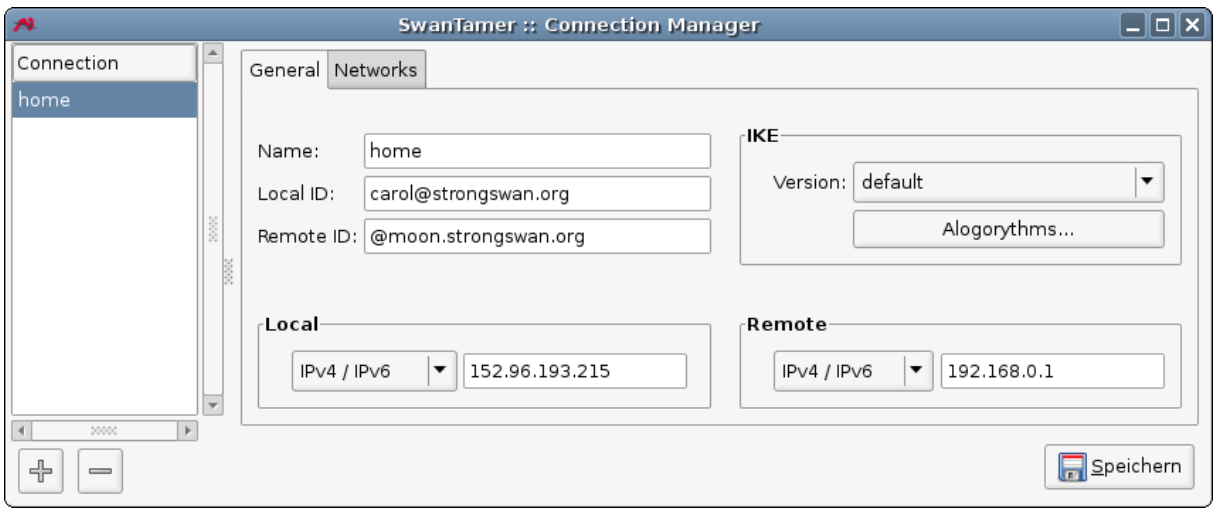
Das Hauptfenster ist so gestaltet, dass man alle wichtigen Informationen auf einen Blick hat. Genauso sollen alle wichtigen Funktionen direkt von hier ausgeführt werden können. Alle anderen Funktionen sind in die anderen Fenster ausgelagert.

Die Zwei häufigsten Usecases in diesem Projekt sind eine Verbindung öffnen und schliessen. Das Anlegen weiterer Verbindungen ist in diesem Fenster nicht vorgesehen, da man das nur selten machen muss.

7.7.2 Connectionmanager

Der ConnectionManager wird seinem Namen gerecht. Er verwaltet, erstellt und löscht Verbindungen.

Für den einfacheren Umgang sind im linken Bereich nochmals alle Verbindungen aufgelistet. So muss das Fenster nicht jedes Mal geschlossen werden, wenn eine andere Verbindung editiert werden soll.



Im rechten Bereich werden alle Eigenschaften einer Verbindung angezeigt. Für die einfachere und sichere Bedienung kann man hier so viel wie möglich mit Auswahllisten auswählen. So können sicher keine falschen Eingaben gemacht werden, und man sieht schnell was alles zur Auswahl steht.

Damit auch die IP-Adressen überprüft werden können, muss beim Eintragen eines Hosts immer angegeben werden, um was für eine Art von Host es sich handelt. Denn im Fall einer falschen IP Eingabe würde diese automatisch als Hostnamen erkannt und validiert.

7.8 SmartPointer

In der Datenhaltung werden ausschliesslich SmartPointer anstelle von Pointern verwendet. In den Bereichen der PD und UI werden zurzeit noch Pointer eingesetzt.

Der Grund für eine uneinheitliche Lösung ist, dass die Datenhaltung anfangs auch mit Pointern implementiert war. Nach und nach zeigten sich jedoch verschiedene Fehler wie Pointer auf Memory-Bereiche, welche bereits freigegeben waren. Nach analysieren des Fehlers zeigte sich, dass die Fehlerbehebung sehr Zeitraubend sein würde. Aufgrund der nicht vorhandenen Erfahrung haben wir uns deshalb entschieden ein Refactoring auf SmartPointer durchzuführen. Zurzeit ist dieses Refactoring mangels Zeit noch nicht bis zur PD durchgedrungen und somit nicht überall umgesetzt.

7.9 Threads

Das SwanTamer GUI besteht zurzeit aus zwei Threads.

Thread 1 (der Start Thread) beinhaltet die ganze Verwaltung des GUI's. Weiterhin werden die ganzen Signale in diesem Thread abgearbeitet. Dieser Thread wird hier in der Dokumentation zum Teil auch als „Main Thread“ bezeichnet.

Thread 2 - ein „worker Thread“ - ist für die Aktualisierung der Statusanzeigen der Connections verantwortlich. Er besteht aus einem endlos Loop, welcher jede Sekunde ein Signal zum Updaten der Statusanzeigen für die Connections schickt.

7.10 Makefile

Am Anfang dieser Semesterarbeit versuchten wir uns mit Autoconfig anzufreunden. Leider mussten wir feststellen, dass Autoconfig und Automake einige Einarbeitungszeit benötigen. So haben wir uns entschieden, das Makefile selber zu schreiben.

Folgende Targets unterstützt das Makefile zurzeit:

all	Dies ist das „default Target“. Übersetzt SwanTamer aus dem Sourcecode. Das generierte Binary wird dabei in das „bin“ Verzeichnis abgelegt. Bereits übersetzte Dateien werden bei einem zweiten Aufruf nicht erneut übersetzt.
install	Installiert die nötigen Dateien im System.
uninstall	Entfernt SwanTamer komplett vom System.
clean	Löscht alle generierten Dateien (Dokumentation, Binary, Objectfiles, ...), welche von „make“ kreiert wurden
doxygen	Generiert die Sourcecode Dokumentation und legt diese im „doc“ Verzeichnis ab.

Um den PREFIX für die Installation anzupassen, existieren im Makefile unter anderem folgende Zeilen:

```
1: PREFIX := /usr/local
2: SYSBINDIR := $(PREFIX)/bin
3: SYSCONFDIR := $(PREFIX)/share/swanTamer
```

Diese drei Variablen müssen vor dem Kompilieren geändert werden, ansonsten muss zuerst ein „make clean“ ausgeführt werden.

7.11 Sourcecode Dokumentation

Für die Sourcecode Dokumentation haben wir uns für Doxygen entschieden.

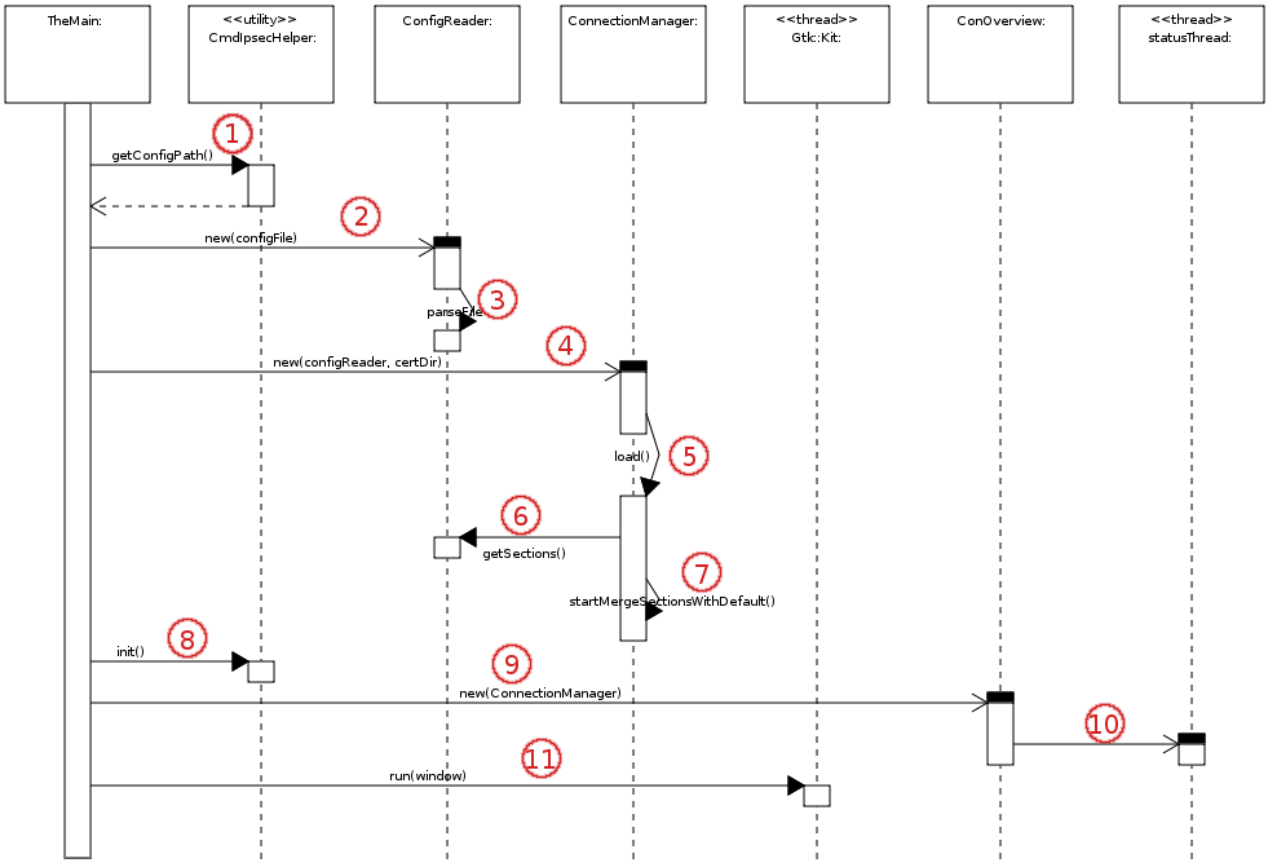
Für die Dokumentation haben wir dabei eine JavaDoc ähnliche Syntax gewählt, da diese für uns am einfachsten erschien.

Für die Klassendokumentation der UI Klassen, haben wir jeweils Bilder in diese Klassen generiert und in der Dokumentation integriert. Die Bilder selbst, sind dabei im „doc/img“ Verzeichnis zu hinterlegen, damit Doxygen diese auch findet.

8 Startup Routine

Um einen Überblick über die Logik hinter dem GUI zu erhalten, wollen wir die Startroutine etwas genauer ansehen.

8.1 Sequenzdiagramm



8.2 Beschreibung

1) Ermittelt den Pfad zur Konfigurationsdatei
2) Erstellt einen neuen ConfigReader...
3) ...welcher das angegebene File sofort einliest und es in seine Bestandteile (Section & ValueTypes) zerlegt.
4) In diesem Schritt wird die PD initialisiert. Ein ConnectionManager wird erstellt...
5) ... dieser lädt als erstes die Konfiguration.
6) Das macht er mit Hilfe des ConfigReaders.
7) Anschliessend werden die Konfigurationsdaten in Connections und Policies umgewandelt. Nachdem alle Sections umgewandelt worden sind, wird die Default Connection (sofern Vorhanden) auf alle anderen Connections angewendet und anschliessend gelöscht. Diese Funktion ist rekursiv und geht durch alle Section und deren Subsection hindurch.

8) Anschliessend wird getestet, ob alle verfügbaren Kommandozeilentools verfügbar sind und die nötigen Rechte existieren.
9) In diesem Schritt wird das UI initialisiert indem ein ConOverview Window erzeugt wird. Dieses Fenster wird als Schnittstelle zur PD dem ConnectionManager mitgegeben.
10) Das ConOverview Window startet noch in der Initialisierung einen Thread der jede Sekunde den Status aller Verbindungen aktualisiert. Dieser Kindthread kommuniziert dabei über einen Signalhandler mit dem Mainthread.
11) Startet mit Hilfe des Gtk::kit das User Interface und bringt es zur Anzeige.

9 Libraries & Programme

9.1 Das Programm

Kurz gesagt, hängt SwanTamer nur von einer Library ab, der GTKmm.

Verschiedene Distributionen teilen jedoch gtkmm in mehrere Subpakete auf. Dabei sind dann die Subpakete glibmm, gtkmm, libglademm nötig.

SwanTamer wurde mit der GTKmm in Version 2.4 entwickelt. Es wurde nicht getestet, ob das Programm mit anderen Versionen lauffähig ist. Leider ist die Versionsvielfalt bei GTKmm nicht sehr übersichtlich, daher ist die Homepage (http://www.gtkmm.org/download.shtml) und folgendes Zitat daraus zu beachten.

> ....
> gtkmm 2.4
>
> Note that in this context, gtkmm 2.4 refers to the API / ABI compatibility version. So don't be surprised if your distribution actually provides a later version such as gtkmm 2.8.x when you install the gtkmm 2.4 package.
> ....

9.2 Das Projekt

- Um das UI zu erstellen haben wir mit dem „Glade Interface Designer“ gearbeitet.
- Um die UML Diagramme zu erstellen haben wir mit dem Tool ArgoUML gearbeitet. Dieses Tool ist unter http://argouml.tigris.org/ frei (unter BSD Lizenz) erhältlich. Leider ist es in den uns bekannten Distributionen nicht enthalten.
- Um die Dokumentation zu erstellen haben wir mit OpenOffice v.2.0 gearbeitet.
- Die Grafiken sind in gängigen Formaten gespeichert. Um Pixelbilder in ihrer „Rohform“ zu editieren haben wir GIMP verwendet.
- Für Vectorbilder (*.svg) haben wir mit Inkspace gearbeitet.
- Die SourceCode Dokumentation wurde mit Hilfe von Doxygen (http://www.stack.nl/~dimitri/doxygen/) erstellt. Als Syntax für die Dokumentation haben wir eine Java-ähnliche Möglichkeit ausgewählt.

10 Implementationsbesonderheiten

In diesem Kapitel ist alles festgehalten was vom grundlegenden Aufbau abweicht, und eine Ausnahme macht. Aber auch Implementierungen die man nicht auf den ersten Blick versteht, oder besonders genial gelöst sind, sind hier aufgelistet.

10.1 DH: configValueParser

In dieser Klasse ist definiert, welchem Typ ein Schlüsselwort der Konfiguration entspricht und in welchen Block es gehört (CA, Global, Connection, Policy).

Dabei sind viele Schlüsselwörter mit dem valueType ARG_STR bzw. ARG_MISC definiert. Es ist einzig bei den Schlüsselwörtern, welche man mit SwanTamer verändern kann gewährleistet, dass ein entsprechendes valueType Objekt erstellt wird, damit diese einfach verändert werden können.

Auf diese Weise werden nicht alle verfügbaren ValueType-Klassen verwendet (z.B. TimePeriod). Allfällige vorhandene Fehler in diesen Implementierungen können so umschifft werden.

Z.B. wird für das Schlüsselwort „plutodebug“ (entspricht einer Auflistung der gewünschten Debug-ausgaben) nicht ein Listenobjekt, sondern ein Stringobjekt erstellt.

10.2 PD: strongSwan Kommunikation

Bei der Implementation der Klasse, welche mit strongSwan kommuniziert, tauchten Probleme auf. Diese zeigten sich dadurch, dass ein Aufruf von system() zum Starten einer Connection die IPsec Verbindung nicht immer erfolgreich aufbauen konnte. Es schien, als ob dieser Aufruf mittendrin abgebrochen wird. Der Grund, warum das so ist, können Thomas und ich auch heute noch nicht benennen.

Beim Testen ist uns aufgefallen, dass bei einem erneuten Klicken gleich nach einem misslungenen Verbindungsversuch die Erfolgsrate für diesen zweiten Versuch erheblich gestiegen ist. Somit erweiterten wir den Sourcecode, damit gleich nach dem ersten system() Aufruf, um den Tunnel zu starten, nochmals der gleiche Systemaufruf an das Betriebsystem geschickt wird.

Zu sehen ist dies in der CmdIpsecHelper::startConnection() Funktion.

10.3 UI: UIConnectionManager

Die Methode setSelected (die von save aufgerufen wird) „simuliert“ einen Klick in die Tabelle. Dabei wird der gleiche Event ausgelöst, wie wenn real mit der Maus eine Connection in der Tabelle ausgewählt wird. Darum leitet diese Methode den Aufruf nicht an die beiden Tabs weiter. Dies wird dann durch die Methode on_treeview_row_activated erledigt.

11 Lokalisierung

Zurzeit sind nur die meisten Buttons lokalisiert. Für restliches, wie Textlabels, besteht zurzeit keine Lokalisierung. Dies aus dem Grund, da GTK für Standardbuttons automatisch eine Lokalisierung anbietet. Um weitere Teile zu lokalisieren, reichte die Zeit nicht aus.

Momentan sind beinahe alle Texte im glade xml-File definiert. Ausnahmen sind hierbei die Errormeldungen, einzelne Buttons und die Spaltenüberschriften bei den Tabellen.

In der GTKmm Dokumentation widmet sich ein Kapitel der Lokalisierung. Dort wird beschrieben, wie man „gettext“ für die Lokalisierung verwendet. Der Englische Text befindet sich in den Beispielen der GTKmm Dokumentation bereits im Sourcecode. Um die Texte im xml-File auch übersetzen zu können, müsste man jedem Textfeld und Co einen Namen geben. Im Konstruktor des Fensters würde man alle Textfelder auslesen und neu Setzen.

12 Vorhandene Bugs

- Beim Start von Ipsec können verschiedene Fehler auftreten. SwanTamer kann diese zurzeit nicht unterscheiden. Als Beispiel: Wenn sich ein Fehler in der Konfigurationsdatei befindet, wird als Fehlermeldung gefreagt, ob sie SwanTamer als Root ausgeführt haben.
- Wenn sich in den Sourcen nur in einem *.h File etwas geändert hat, so erkennt make die Abhängigkeiten nicht und übersetzt diese nicht neu. In diesem Fall muss zuerst ein make clean ausgeführt werden, und anschliessend ein make all.

13 Ausblicke

Wie in jedem Projekt (auch wenn man seine Ziele erreicht) gibt es Arbeiten die man noch tun müsste, um das Programm abzurunden und zu perfektionieren. Was uns am Ende dieses Projekts aufgefallen ist, was noch fehlt, haben wir hier aufgelistet. Die Liste ist nach Prioritäten geordnet, was wir denken was wichtig ist, und welche Features noch etwas warten können.

1. Update thread -> cmd Abfrage in „worker thread“ auslagern
2. Starten einer Connection dauert zu lange -> GUI friert ein
   2.1. Anzeigen der Ausgabe von IPsec zusammen mit einem Progressbar.
   2.2. Anzeige was war beim Connectionaufbau fehlerhaft (wenn Fehler)
3. Zertifikat Erstellung rsp. bessere Eingabemöglichkeiten der Ids.
4. Mehrsprachigkeit / Lokalisierung
5. Host Eingabefeld mit „Auto-erkennung“ was für eine Art Host eingegeben wurde. Hier besteht zurzeit noch das Problem, dass eine falsch eingegebene IP als DNSname interpretiert wird.
6. Umstellung der PD und des UI's von Pointern auf SmartPointer. Die DH arbeitet bereits mit SmartPointern

14 Persönliche Erfahrungen

14.1 Thomas Boksberger

14.1.1 C++

Mein Ineresse an dieser Semesterarbeit wurde nicht zuletzt dadurch hervorgerufen, dass ich einmal etwas andseres als Java/Swing sehen sehen wollte. Ich wollte mir beweisen, dass ich in der Lage bin eine kleine Applikation in C++ zu erstellen. So habe ich mich zusammen mit Marco dieser Herausforderung gestellt und diese Semesterarbeit ausgewählt.

Im Rückblick kann ich sagen, ich habe es mir bewiesen. Ich kann C++ programmieren. Zumindest kann ich das jetzt. Am Anfang hatte ich Schwierigkeiten ohne Nachschlagen in der Referenz, eine einfache Liste zu erstellen und mit Inhalten zu füllen. Erst recht konnte ich nicht ohne weiteres darüber iterieren. Am Ende der Semesterarbeit, fiel es mir leicht diesen Code ohne grosses Nachdenken aufzusetzen.

Womit ich aber bis heute Schwierigkeiten habe, ist dass es in C++ keinen Garbage Collector gibt. Immer muss ich im Auge behalten, was im Deconstruktor alles gemacht werden muss und was auf keinen Fall gemacht werden darf. Damit habe ich mich noch immer nicht ganz angefreundet. Obwohl dieses Vorgehen nun einigermassen klar geworden ist.

14.1.2 Gtk

Gtk mit seiner Objekt orientierten Bibliothek (GTKmm) zusammen mit dem Programm Glade habe ich als sehr angenehm empfunden. Es ist ein übersichtliches Programmieren, wenn die Definition des Aussehens nicht im Code eingebettet werden muss. Natürlich hat auch diese Variante Nachteile. Ich meine aber, dass die Vorteile überwiegen (Kap. 6.6).

Den Einstig in die Programmierung mit GTKmm habe ich schnell gefunden. Ich habe gleich am Anfang eine kleine Referenzimplementierung eines Fensters mit zwei Knöpfen und einer Tabelle erstellt. Zum grössten Teil habe ich diesen Code aus Bespielen kopiert. Auf diese Weise habe ich sehr schnell gelernt einfache GUIs zu erstellen.

Das eine oder anderen Feature, das wir einbauen wollten, bereitete mir aber dennoch einige Probleme. So habe ich mindestens eine Woche an einer editierbaren Dropdownbox programmiert. Schlussendlich habe ich eine Lösung gefunden. Der Code war aber derart schlecht und unübersichtlich, dass wir uns entschlossen haben, das Feature nicht einzubinden. Im Nachhinein hat sich herausgestellt, dass wir es sowieso nicht gebraucht hätten und wir es aus dem GUI wieder hätten entfernen müssen.

14.1.3 Planung

Über die Planung haben wir uns am Anfang nicht so viel Gedanken gemacht. Denn eigentlich war das Vorgehen klar. Zuerst mussten wir die Datenhaltung programmieren, um dann die Problemdomain zu implementieren. Zuletzt mussten wir dann das Userinterface darüber legen. An diese Vorgehensweise haben wir uns auch gehalten. Selbstverständlich hatten wir hin und wieder Änderungen in den unterliegenden Schichten vorzunehmen. Nach meinen Erfahrungen ist dieses Vorgehen aber durchaus üblich.

Mit dieser Arbeitsweise hatten wir aber nie eine Kontrolle, ob wir uns im Zeitplan befinden. Nach einigen Wochen hatte Marco eine Planungsskizze erstellt. Darauf war festgehalten, wann wir die Meilensteine erreicht haben sollten. Diese Skizze benutzten wir aber in der Folge kaum, sodass man beinahe von Glück sprechen könnte, dass wir das Projekt erfolgreich beenden konnten. Wenn ich versuche mich zurück zu erinnern, war aber immer klar, wo wir uns im Zeitplan befanden. Für uns war es immer offensichtlich, ob wir uns beeilen mussten, oder ob wir genügend Zeit hatten.

Es war angenehm so zu arbeiten. Da wir das Projekt von Grund auf selber gestalten konnten, hatten wir immer den vollen Überblick. Ich denke, dass es für uns ein böses Erwachen gegeben hätte, wenn es sich um die Weiterführung eines bestehenden Projektes oder ein neues Teilprojekt, gehandelt hätte.

14.1.4 Betreuung

Die Betreuung unseres Teams war gut. A. Steffen und M. Willi haben uns kompetente Auskunft gegeben und uns sehr gut beraten. Man bemerkte sehr wohl, dass die beiden Betreuer sich mit dem Projekt strongSwan schon lang auseinander gesetzt haben.

In der Zeit in der Herr Steffen nicht da war und wir ausschliesslich durch Herrn Willi betreut wurden, stellten wir fest, dass die Zielrichtung des Projektes sich änderte. Nach der Rückkehr von Herrn Steffen mussten wir erkennen, dass wir uns zu sehr darauf konzentriert hatten, unser Konzept auf die Funktionsweise des Charons anzupassen, anstatt ein möglichst einfaches UI zu gestalten. In unserer Arbeit hat uns das jedoch nicht weit zurückgeworfen, obwohl der Weg etwas verwirrend war.

14.1.5 strongSwan

Es war nicht immer leicht, aus der bestehenden Dokumentation alle Informationen zu erhalten, die wir für unser Projekt benötigten. So war es umständlich herauszufinden, welche Werte für welches Keyword erlaubt sind. Wir hatten auch eine Definition gefunden, wie die Konfigurationsdatei aufgebaut sein müsste. Die angegebenen Beispiel-Konfigurationen hielten sich aber nicht an alle diese Definition.

Diese Umstände haben uns bei der Entwicklung der DH einige Mühe bereitet. Dennoch waren es keineswegs unüberwindbare Probleme. Sie haben aber unsere Arbeit umständlicher gestaltet. In den meisten Fällen haben wir hilfreiche Unterstützung durch unsere Betreuer erhalten.

14.1.6 Unser Code

Womit ich persönlich nie gerechnet hätte, waren die 12'203 Zeilen Code. In einem anderen Projekt erarbeiteten wir ähnlich viel Code, aber da waren wir sechs Programmierer. Der hauptsächliche Unterschied lag wahrscheinlich in der Programmiersprache. Bisher habe ich Applikationen fast ausschliesslich in Java erstellt. In C++ muss für das gleich Resultat einfach mehr Code erstellt werden. Dennoch hätte ich mir nicht vorgestellt, dass der Unterschied so gross sein würde.

Die Grösse des Codes bereitete uns aber nicht wirklich grosse Schwierigkeiten. Zu Beginn bemerkten wir gar nicht viel Code wir schrieben. Erst am Ende des Projektes ist uns dies wirklich aufgefallen.

14.1.7 Fazit

Mein Wissen über C++ das ich mir im Verlaufe meines Studiums angeeignet habe war bislang sehr theoretisch. Es war eine gewinnbrinende Herausforderung dieses Wissen in einem praktischen Projekt zu vertiefen. Auch das Wissen darum, dass dieses Projekt einmal wirklich anderen Menschen nützen würde, gab mir weiteren Ansporn, mich in diese Arbeit zu vertiefen.

Leider war unsere Planung etwas zu sehr vom Laissez-faire geprägt. Wir haben uns dadurch um die Erfahrung gebracht, in einem vordefinierten Zeitplan zu arbeiten. Wir konnten nicht kontrollieren, wie wir zeitlich liegen, oder wie schwierig es sein kann einen Zeitplan zu erstellen und sich daran zu halten. Auf den Erfolg unseres Projektes hatte das keinen Einfluss, obwohl uns diese Erfahrung für unsere Zukunft sicher dienlich gewesen wäre.

14.2 Marco Gruber

14.2.1 C++ vs. Java

Java habe ich erst kennen gelernt als ich hier mit der Hochschule angefangen habe. Ich war damals eher ein Freund von C++ und konnte mich mit Java schwer anfreunden. Umso mehr freute es mich endlich wieder mal abseits von Java zu Programmieren.

Mit ein wenig Schrecken musste ich feststellen, dass ich mich mit C++ anfangs nicht zurechtfinden konnte. Es ist ein völlig anderes Programmieren als mit Java. Besonders der Fehlende Garbage Collector hat mir anfangs doch gefehlt und dazu geführt, dass wird nach ein Paar Wochen die Pointer der Datenhaltung durch SmartPointer ersetzt haben.

Inzwischen habe ich mich an viele Eigenheiten von C++ wieder gewöhnt und muss dennoch sagen, dass ich in Zukunft Java sicherlich vorziehen würde (allgemein gesehen und nicht auf dieses Projekt). In meinen Augen ist Java einfach weniger fehleranfällig und übersichtlicher (z.B. Definition und Implementation getrennt ist ein wenig mühsam). Auch sind die Debugging Möglichkeiten von Eclipse einfach um Welten angenehmer als diejenigen von GDB (auch mit einem GUI oben drauf). Schlussendlich bin ich der Meinung, dass man mit Java schneller am Ziel ist und weniger Aufwand in die Wartung stecken muss. Zumindest in den meisten Fällen. Deshalb sehe ich in Zukunft eher Sprachen wie Java als C ähnliche Sprachen in der Wirtschaft im Einsatz.

Unter dem Strich war es für mich eine wertvolle Erfahrung wieder einmal abseits von Java etwas zu machen.

14.2.2 Gtk

Mit Gtk hatte ich einen sehr schweren Start. Ich kam ein paar Tage nach Thomas mit Gtk in Kontakt, dabei habe ich mit dem „Algorithm Fenster“ angefangen und wollte die DropDown Auswahl direkt in die Tabellenfelder integrieren. Leider zeigte sich, dass Tabellen wie bei Swing (und ich denke auch vielen anderen GUI Frameworks) eine echte Herausforderung sind und musste mein Vorhaben aufgeben.

Die Nutzung von GTKmm und Glade war in meinen Augen ein sehr wichtiger Entscheid. Dieser hat mir einiges an Arbeit abgenommen bzw. von mir fern gehalten.

Swing hatte mich im letzten Semester nicht überzeugt und ich hoffte eigentlich, dass all die

Probleme mit Gtk verschwinden werden. Gewisse Probleme verschwanden auch. Doch gab es etliche neue Probleme/Eigenheiten die sich zeigten.

Gtk und Swing sind in meinen Augen weit von einem angenehmen GUI Framework entfernt. Eine Flexibilität und Einfachheit wie ich mir sie zurzeit vorstelle, wird wohl noch einige Jahre ein Traum sein.

14.2.3 Planung

Meiner Meinung nach war die Planung selbst genauso wie uns das immer wieder von Herrn Sommerlad (HSR Dozent) eingetrichtert wurde. Anfangs Woche hatten wir jeweils das Meeting, an dem wir die nächsten Ziele besprochen haben. Es wurde jedoch nie ein Termin für diese besprochenen Aufgaben gesetzt. Thomas und ich haben dann meist diese Arbeiten bis zum nächsten Meeting erledigt. Unter dem Strich würde dann eine Woche für eine Iteration im RUP Prozess stehen.

Eine Planung über das ganze Semester existierte erst nach ein Paar Wochen in unseren Köpfen. Anfangs Dezember haben wir uns dann auch mal aufgerafft, diesen auf einem Zettel festzuhalten. Angeschaut haben wir diesen danach nie wieder. Dennoch haben wir uns besonders im zweiten Teil des Semesters immer wieder mal über die restliche Zeit unterhalten.

Eine grosse Lücke ist in meinen Augen, dass wir die gesamte Planung nie richtig niedergeschrieben haben. Alles ging gut, bis wir nun Ende Semester die Dokumentation erstellt haben. Im Nachhinein alles niederzuschreiben braucht viel mehr Zeit, als wenn man jede Woche ein paar Zeilen geschrieben hätte.

Sofern seine Arbeit nicht von anderen abhängt, wäre eine solche Planung wie wir sie hatten völlig in Ordnung. Bei grösseren Teams und voneinander abhängigen Teilen, würde ein solches Vorgehen in jedem Fall zu Verzögerungen führen (sofern man einen engen Zeitplan hat).

14.2.4 Betreuung

Die Betreuung durch A. Steffen und M. Willi empfand ich als sehr angenehm. Besonders am Anfang als wir zwei Grünschnäbel von strongSwan noch keine Ahnung hatten, beantworteten sie geduldig unsere Fragen. Sofern man Probleme hatte, konnte man sogar zwischen den Meetings ohne sich anzumelden in ihrem Büro vorbeischauen und Fragen stellen. Waren sie mal nicht an ihren Arbeitsplätzen, sendete man eine E-Mail welche schnell beantwortet wurde.

Momentan fallen mir gerade mal zwei Negativpunkte auf, wovon nur der erste wirklich der Betreuung gilt.

Es gab Meetings, an denen nicht beide Betreuer anwesend waren. Dies ist eigentlich kein Problem. An solchen Meetings, merkte man aber sehr schnell dass sich die einzelnen Personen das GUI zum Teil unterschiedlich vorstellten. Es konnte also sein, dass in einer Woche das Projekt in diese Richtung zeigte, eine Woche später in eine (jedoch nicht komplett) andere Richtung und eine Woche darauf wieder in die vorangegangene.

Der zweite Negativpunkt war, dass dies unsere erste Semesterarbeit war. In meinen Augen wurden wir von der Schule zu spät bis gar nicht informiert, wie das ganze abläuft. Ich

vermisste frühzeitige Informationen, wie das Auswahlverfahren für die Arbeit funktioniert, Requirements in der Dokumentation, ...

14.2.5 strongSwan

Ein OpenSource Projekte und die Dokumentation dazu ist immer so eine Sache. Es gibt wie überall gute, mässige und schlechte Beispiele. Aber in der OpenSource Ecke gibt es von den schlechten sehr viele. Umso mehr habe ich mich im vornherein über ein solches Projekt gefreut, welches an unserer Hochschule von Studenten massgeblich mitentwickelt wird. Es musste somit eine Vorbildliche Dokumentation vorhanden sein.

Doch bereits zu beginn des Semesters, als wir uns an die Planung und Implementierung der Datenhaltung machten, zeigte sich ein anderes Bild. Die Manpage, welche unter Linux/Unix erste Anlaufstelle bei Fragen ist, war an etlichen Ecken veraltet, unvollständig oder in meinen Augen etwas zu ungenau.

Auch wenn die Dokumentation nicht gerade meinen Wünschen entsprach, bin ich über die Stabilität von strongSwan positiv überrascht. Im produktiven Einsatz war mir klar, dass strongSwan stabil laufen musste. Dass es jedoch auch mit unseren Teils unüblichen Einstellungen auch problemlos lief, überraschte mich doch ein wenig. Jegliche Probleme die ich hatte, liessen sich auf falsche Konfiguration oder Probleme mit den UserModeLinux Instanzen zurückführen.

14.2.6 Fazit

Meine Erste Semesterarbeit und meine beiden Ziele welche in mir für die beiden Semesterarbeiten vorgenommen habe sind bereits erfüllt. Diese waren einerseits ein Projekt nicht mit Java zu machen und nach unserem SoftwareEngineering2 Projekt wollte ich einmal ein anderes GUI Framework als SWING kennenlernen.

Unter dem Strich hat mir die Semesterarbiet Spass gemacht, auch wenn wir hin und wieder lange mit der Fehlersuche beschäftigt waren, welche zum Teil sehr an der Motivation gezerrt haben.

15 Abkürzungen

DH	Daten Haltung
PD	Problem Domain
UI oder GUI	Grafical User Interface
Gtk	Gimp Tool Kit (Library für das GUI)
GTKmm	Objektorientierter Wrapper für Gtk

16 Literaturverzeichnis

C++ Referenz

http://www.cppreference.com/

GTKmm Dokumentation

http://www.gtkmm.org/docs/gtkmm-2.4/docs/tutorial/html/

GTK Referenz

http://www.gtkmm.org/docs/gtkmm-2.4/docs/reference/html/namespaceGtk.html

Glib Referenz

http://www.gtkmm.org/docs/glibmm-2.4/docs/reference/html/namespaceGlib.html

GTK und Threads

http://developer.gnome.org/doc/API/2.4/gdk/gdk-Threads.html

Lokalisierung

http://www.gnu.org/software/gettext/

http://www.gnu.org/software/gettext/manual/html_chapter/gettext_toc.html

http://www.gtkmm.org/docs/gtkmm-2.4/docs/tutorial/html/ch24.html