HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Bachelorarbeit, Abteilung Informatik

strongMan

HSR Hochschule für Technik Rapperswil

Frühjahrssemester 2016
18. Juni 2016

Autoren: Bühler Severin & Kurath Samuel
Betreuer: Prof. Dr. Andreas Steffen

Arbeitsperiode: 22.02.2016 - 18.06.2016

0.1 Abstract

Die VPN Applikation ist weltweit stark verbreitet, nun soll es möglich sein die Konfiguration per graphischen Interface zu vereinfachen.

Weitere Informationen: `https://github.com/strongswan/`

0.2 Management Summary

Ausgangslage

Management per UI.

Vorgehen

Erarbeitung der Lösung

Ergebnisse

BA

Ausblick

Erweitern

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Entscheide

0.3 Aufgabenstellung

Institute for Internet Technologies and Applications

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
FHO Fachhochschule Ostschweiz

Bachelorarbeit 2016

Django-basiertes Management Tool für strongSwan

Studenten: Severin Bühler und Samuel Kurath

Betreuer: Prof. Dr. Andreas Steffen

Ausgabe: Montag, 22. Februar 2016
Abgabe: Freitag, 17. Juni 2016

Ausgangslage

Die strongSwan Open Source VPN Software wurde vor zwei Jahren mit dem neuen Versatile IKE Configuration Interface (VICI) ausgestattet, eine JSON-artige Schnittstelle, welche es erlaubt eine ManagementAnwendung über ein C, Ruby, Python oder Perl Binding an den Charon IKE Daemon anzubinden.

Aufgabenstellung

In dieser Arbeit soll auf der Basis von Django/Python eine grafische Management-Anwendung für strongSwan geschaffen werden, welche es erlaubt, IPsec Verbindungen über ein Web-Interface zu definieren, in einer Datenbank zu speichern und via die VICI-Schnittstelle an den IKE Daemon zu übermitteln. Weiter soll der Stand der aktuellen IPsec Verbindungen und andere statistische Daten abgefragt werden können.

Ziele

- Implementation einer grafischen Management-Oberfläche mit Django für strongSwan zur Konfiguration eines VPN Clients für folgende vier IKEv2 Authentisierungsmethoden:
  1) X.509 Zertifikat und privater RSA/ECDSA Schlüssel
  2) EAP mit Benutzername/Passwort
  3) Zweirunden-Authentisierung mit Methode 1) gefolgt von Methode 2)
  4) EAP-TLS mit X.509 Zertifikat und privatem RSA/ECDSA Schlüssel
- Oberfläche zur Verwaltung von X.509 End Entity und CA Zertifikaten, sowie privater RSA/ECDSA Schlüssel.
- Persistierung der Konfigurationsdaten in einer Datenbank.
- Verschlüsselte Ablage der RSA/ECDSA Authentisierungsschlüssel in der Datenbank.
- Starten und Stoppen von konfigurierten VPN Verbindungen
- Darstellung von Statusinformation über aktive VPN Verbindungen
- **Optional**: Oberfläche zur Konfiguration eines VPN Gateways

Links

[1] RFC 7296 Internet Key Exchange Protocol Version 2 (IKEv2)
https://tools.ietf.org/html/rfc7296

[2] Spezifikation der strongSwan VICI Schnittstelle
https://github.com/strongswan/strongswan/blob/master/src/libcharon/plugins/vici/README.md

[3] VICI Konfigurationsbeispiele (benutzt swanctl.conf Konfigurationsdatei)
https://www.strongswan.org/testing/testresults/swanctl/

[4] strongTNC Policy Manager als beispielhafte Django Anwendung
https://github.com/strongswan/strongTNC

Rapperswil, 22. Februar 2016

Prof. Dr. Andreas Steffen

Kapitel 1

Technischer Bericht

1.1 Einführung

1.1.1 Vision

Aktuell StrongSwan wird standardmässig per Konfigurationsdateien verwaltet, dies zielt stark auf eher versierte Nutzer und Administratoren ab.

Vision Es soll eine Applikation entstehen, mit der dieser komplexe Prozess erleichtert wird. Dabei wir auf ein graphisches Interface gesetzt.

1.2 Stand der Technik

Android App.

1.2.1 Literaturrecherche

Suchquellen

Folgende Quellen wurden uns empfohlen, um Recherchen in diesem Umfeld durchzuführen:

- a
- b
- c

Auswertung

-

Fazit

-

1.3 Evaluation Usermanagement

1.3.1 Auswertung

-

Entscheid 1. Evaluation Usermanagement

-

1.4 Evaluation Zertifikatstypen

1.4.1 Auswertung

-

Entscheid 2. Evaluation Zertifikatstypen

-

1.5 Evaluation Zertifikatsbibliothek

1.5.1 Auswertung

-

Entscheid 3. Evaluation Zertifikatsbibliothek

-

1.6 Umsetzungskonzept

Die Grundidee für die Umsetzung des Management Tools:

1. 1
2. 2

1.7 Resultate

1.7.1 Zielerreichung

-

Ziel der Aufgabenstellung

Ziel	Resultat
Evaluation eines effizienten Algorithmus zur Erkennung von Fussgängerstreifen auf Orthofotos.	Diverse Algorithmen wurden evaluiert, mit dem Deep Learning Ansatz wurde ein klarer Favorit ermittelt. Mehr dazu unter dem Abschnitt **??** auf der Seite **??**.
Automatische Verarbeitung von Orthofotos.	Es wird automatisch auf Bilder von Bing Maps zugegriffen.
Extraktion der Koordinaten von Fussgängerstreifen aus Orthofotos (Kanton Zürich, optional Europa oder mehr).	Zürich konnte von der Applikation verarbeitet werden, weiter wurde die Suche auf die Ostschweiz ausgebaut.
Evaluation des Crowdsourcing-System zur Daten-Validierung und Übertragung in OSM.	Bei der Evaluation des Crowdsourcing-Systems setzte sich MapRoulette durch die Bekanntschaft bei der Community durch. Mehr dazu unter dem Abschnitt **??** auf der Seite **??**.
Erstellung einer Challenge für das Crowdsourcing-System anhand der gesammelten Daten.	Eine Challenge wurde für MapRoulette generiert und publiziert.

Tabelle 1.1: Resultate

Die in der Aufgabenstellung formulierten Ziele konnten alle in einem angemessenem Rahmen erreicht werden.

1.7.2 Persönlicher Bericht

Mit dem Resultat des Projektes sind wir äusserst zufrieden. Es wurde viel neues dazugelernt, sowohl in technischen Bereichen, wie auch in der Teamkommunikation und dem Projektmanagement.

Neu erlernte Technologien

- Python
- Docker
- Latex

1.7.3 Dank

-

Kapitel 2

Software Projektdokumentation

2.1 Vision

Die Vision ist unter dem Abschnitt 1.1.1 auf der Seite 12 zu finden.

2.2 Anforderungsspezifikation

2.2.1 Allgemeine Beschreibung

Im generellen sind zwei Anwendungsszenarien denkbar:

- VPN-Client
- VPN-Gateway

Dabei ist der VPN-Client eine **Muss**-Anforderung und der VPN-Gateway eine **Kann**-Anforderung.

VPN-Client

Die Applikation wird von einem Standard-Nutzer verwendet. Dieser soll VPN Tunnels zu Gateways konfigurieren können und die Tunnels starten und stoppen. Die Konfigurationsmöglichkeiten sind beschränkt, als Richtwert wird der strongSwan Android Client verwendet.

VPN-Gateway

Der Gateway ist auf Systemadministratoren ausgerichtet. Es soll möglich sein per grafischem Interface strongSwan zu konfigurieren und Tunnels einzurichten, welche als Gateway genutzt werden.

2.2.2 Use Case

Aktoren und Stakeholder

Aktor	Tätigkeit
User	• Konfiguriert VPN-Tunnel als Client • Startet und stoppt VPN-Tunnel
Administrator	• Konfiguriert VPN-Tunnel als Gateway • Startet und stoppt VPN-Tunnel

Tabelle 2.1: Aktoren und Stakeholder

hallo welt

Use Case Diagramm

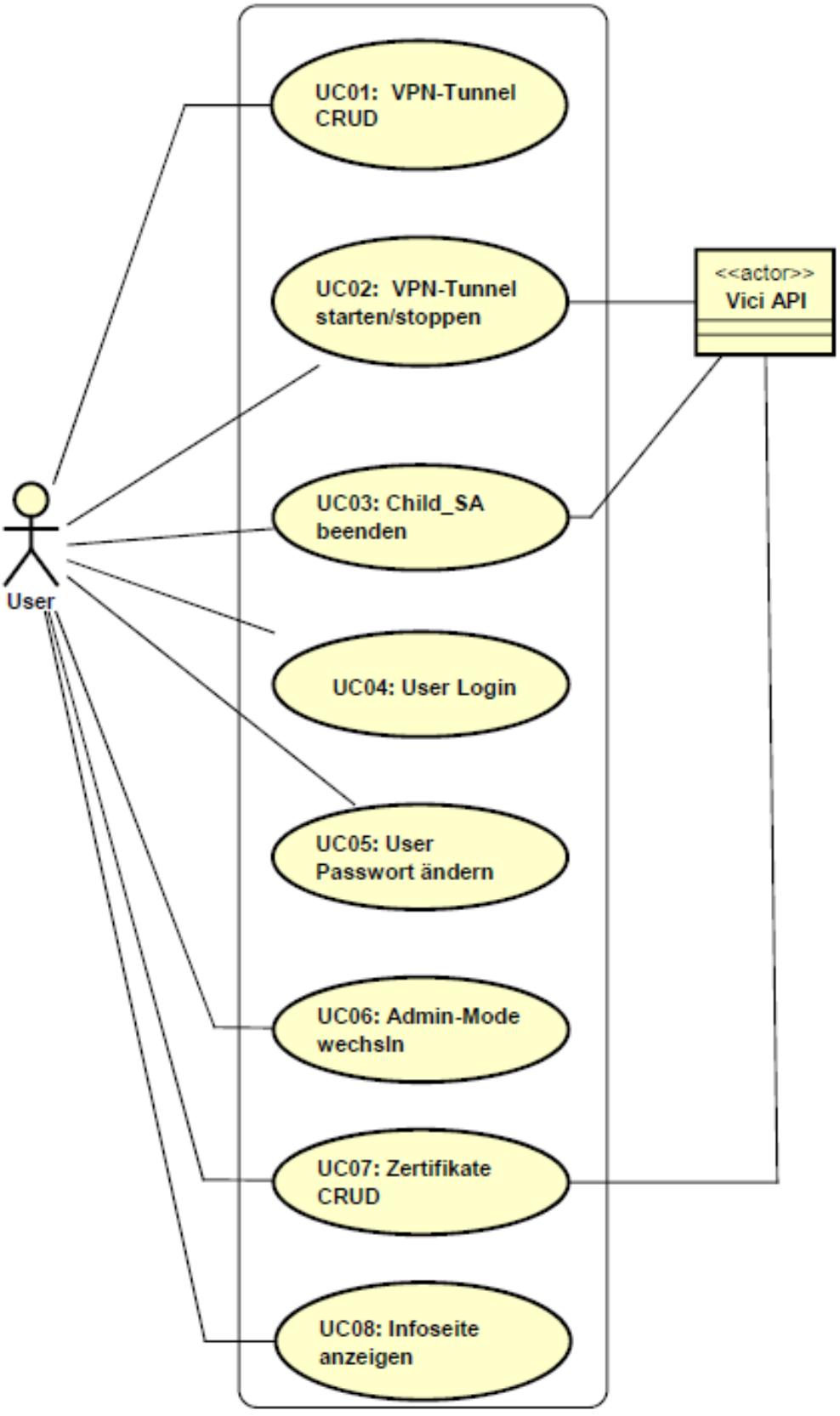

Abbildung 2.1: Use Case Diagramm

Use Cases Brief

Alle hier definierten Use Cases haben auch ein entsprechendes Mockup im Anhang.

UC01: VPN-Tunnel CRUD
Der User kann einen VPN-Tunnel erfassen / konfigurieren. Dabei hat er eine Auswahl von verschiedenen vordefinierten Tunneltypen. Jeder Tunneltyp hat eigene Konfigurationsfelder, die der User ausfüllen muss. Die Tunnel-Übersichtsseite stellt die Hauptseite der Applikation dar. Dort können die Tunnels bearbeitet und gelöscht werden.

UC02: VPN-Tunnel starten/stoppen
Der User kann einen erfassten VPN-Tunnel starten und stoppen. Dabei wird die Konfiguration über die Vici API geladen. Falls ein VPN-Tunnel nicht aufgebaut werden kann, soll eine passende Fehlermeldung angezeigt werden.

UC03: Child_SA beenden
Jeder VPN-Tunnel kann mehrere Child_SA enthalten. Dieser werden in der Hauptseite angezeigt und können vom User beendet werden. Dieser Use Case interagiert mit der VICI Schnittstelle.

UC04: User Login
Der User loggt sich zu Beginn beim Webseiten Aufruf mit einem Passwort ein. Es existiert dabei nur ein User mit Passwort.

UC05: User Passwort ändern
Sobald der User eingeloggt ist, hat er die Möglichkeit, sein Passwort zu ändern. Dabei gibt er sein altes Passwort einmal und sein neues Passwort zweimal ein.

UC06: Admin-Mode wechseln
Das Userinterface unterscheidet zwischen zwei Modis: User- & Admin-Mode. Der Mode kann durch einem Klick auf einen Button gewechselt werden. Der Admin-Mode stellt einige Gateway-spezifische Funktionalitäten zusätzlich zur Verfügung, welche der User zur einfacheren Bedienung nicht sieht.

UC07: Zertifikate CRUD
Dem User wird eine Zertifikatsverwaltung zur Verfügung gestellt. Er kann Zertifikate und Private Key's in den gängigen Formaten uploaden, anschauen, updaten (Passwort ändern) und wieder löschen. Die Dateien können mit einem Passwort verschlüsselt sein. Dieser Use Case interagiert unter Umständen mit der VICI Schnittstelle.

UC08: Infoseite anzeigen
Die Infoseite zeigt dem eingeloggten User verschieden Informationen über das installierte System wie Charon Version, installierte Plugins usw.

Definition Konfigurationsmöglichkeiten UC01

VPN-Client
Folgende Tunneltypen müssen unterstützt werden:

- IKEv2 Zertifikat
- IKEv2 EAP (Benutzername/Passwort)
- IKEv2 Zertifikat + EAP (Benutzername/Passwort)
- IKEv2 EAP-TLS (Zertifikat)

Name	swanclt	vici
Profilname	connections.<conn>	<IKE_SA config name>
Typ	"Verbindungsarten"	"Verbindungsarten"
Gateway	connections.<conn>.remote_addrs	remote-host
EAP Username	connections.<conn>.local<suffix>.eap_id (Ref: secrets.eap<suffix>.id<suffix>)	<IKE>.remote-eap-id
EAP Passwort	secrets.eap<suffix>.secret	<secret>.data
Gateway-Port	connections.<conn>.remote_port	<IKE>.remote-port
CA-Zertifikat	connections.<conn>.remote<suffix>.cacerts	
User-Zertifikat	connections.<conn>.local<suffix>.certs	

Tabelle 2.2: Eingabefelder VPN-Client

Eingabefelder

VPN-Gateway

2.2.3 Sequenzdiagramm

User

Das Sequenzdiagramm beschreibt...

2.2.4 Nichtfunktionale Anforderungen

Funktionalität

Sicherheit

Interoperabilität

Richtigkeit

Zuverlässigkeit

Wiederherstellbarkeit

Fehlertoleranz

Availability

Benutzbarkeit

Robustheit

Effizienz

Supportability

Internationalization

2.2.5 Analyse

Beschreibung Domain Modell

-

2.3 Implementation

-

2.4 Testing

In der Python Standard Library gibt es das Unit Testing Framework **unittest**, das es erlaubt Unittests zu implementieren. Django's Unit tests basieren auf dieser Library und erweitern diese, beispielsweise erbt **django.test.TestCase** von **unittest.TestCase**. Weiter werden auch Integrationtests ermöglicht.

Beispiel Integrationtests

```
from django.test import Client, TestCase
from django.contrib.auth.models import User

class AboutViewsTests(TestCase):

    def setUp(self):
        self.user = User.objects.create(username='testuser')
        self.user.set_password('12345')
        self.user.save()
        self.client = Client()
        self.client.login(username='testuser', password='12345')

    def test_about_get(self):
        response = self.client.get('/about/')
        self.assertEquals(response.status_code, 200)
```

2.4.1 Continuous Integration (CI)

Anforderungen

- Aufbau von IPsec Tunnel zwischen mindesten zwei Rechnern
- automatisierter Ablauf
- geringe Einarbeitungszeit in Technologien
- möglichst kostenfreie Dienste nutzen

Tools

GitHub wird von uns als online Repository verwendet und wird mit dem Versionsverwaltungssystem Git eingesetzt. Weiter bietet es für andere Dienste WebHooks an.

Travis CI wird als eigentlicher Continuous Integration Anbieter verwendet, es werden automatisierte Builds und Tests ermöglicht. Travis CI ist für Projekte, welche auf GitHub gehostet werden ausgelegt.

Docker um Integration Tests zu ermöglichen wird docker-compose eingesetzt, welches gewährleistet mehrere Docker Container zu builden und eine Netzwerk untereinander aufzubauen.

Entscheid 4. Toolstack CI

Um einen IPsec Tunnel aufzubauen nutzen wir zwei Docker Container mit Hilfe von docker-compose. Travis CI unterstützt docker-compose und lässt sich nahtlos mit GitHub kombinieren. Diese Technologien sind uns schon bekannt und werden kostenlos zur Verfügung gestellt.

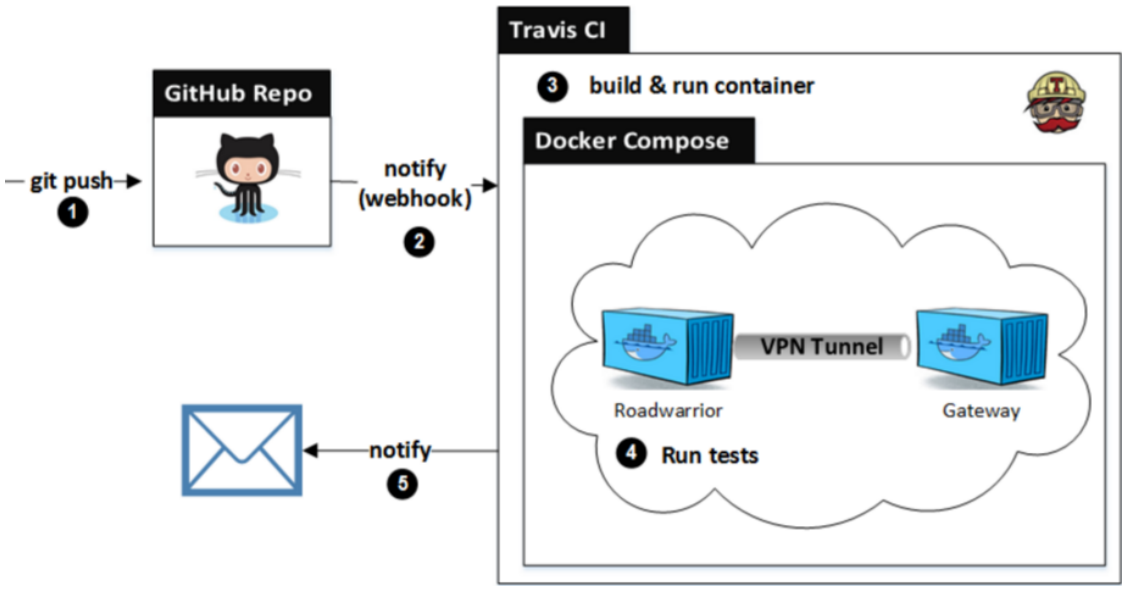

Abbildung 2.2: Schematische Darstellung der Testumgebung

Ablauf

1. git push, lokaler Code wird auf GitHub publiziert
2. Travis CI wird durch WebHook notifiziert
3. Travis CI buildet zwei Docker Container, welche auf der Basis des offiziellen Django Container aufbauen
   (a) StrongSwan mit den nötigen Plugins wird installiert und gestartet
   (b) Auf dem Roadwarrior wird die strongMan Applikation vom GitHub Repository installiert und der aktuelle Branch wird eingecheckt
4. Der Roadwarrior started die Unit- sowie die Integrationstests, dabei werden zwischen den Docker Container diverse Ipsec Tunnels auf- und abgebaut
5. abschliessend wird über Erfolg oder Misserfolge per Email notifiziert

Kapitel 3

Projektmanagement

3.1 Rollen und Verantwortlichkeiten

3.1.1 Prof. Keller Stefan

Abbildung 3.1: Prof. Dr. Andreas Steffen

Professor, Institutsleiter ITA

3.1.2 Bühler Severin

Abbildung 3.2: Severin Bühler

Severin Bühler, Student an der HSR, ist Entwickler des Projektes.

3.1.3 Kurath Samuel

Abbildung 3.3: Samuel Kurath

Samuel Kurath, Student an der HSR, ist Entwickler des Projektes.

3.2 Entwicklungsumgebung und Infrastruktur

3.2.1 IDE (Integrated Development Environment)

Entscheid 5. PyCharm

Beiden Projektmitgliedern ist JetBrains Intellij bekannt und PyCharm ist im Umgang nahe zu identisch. Für Studenten sind die Entwicklungsumgebungen kostenlos verfügbar.

3.2.2 SCM (Source Control Management)

Entscheid 6. GitHub

Der Umgang mit Git ist beiden Projektmitglieder bestens bekannt. GitHub ist ohne Unkosten von überall verfügbar. Das Geometalab der HSR publiziert über diesen Weg diverse Projekte.

3.2.3 CI (Continuous Integration)

Entscheid 7. CircleCI

Das finden eines passenden Continuous Integration Tools stellte sich schwieriger dar, als zu Beginn des Projektes erwartet. Während dem SE2 Projekt haben wir Bekanntschaft mit Travis CI gemacht, welches die vielen Abhängigkeiten unseres Codes nicht abdecken konnte. Mit Circle-CI fanden wir eine Lösung, die auf Docker Hub zugreifen kann, dann den Build des Images durchführt und schlussendlich die Test durchführt.

3.2.4 Projektmanagement Tool

Entscheid 8. Jira

Jira ist den Projektmitgliedern schon aus dem SE2-Projekt bekannt und hat sich sehr bewährt. Das Dashboard ist übersichtlich gestaltet. Es ermöglicht eine Übersicht über die aktuellen Tasks auf einen Blick. Alle Mitglieder haben jederzeit Zugriff auf die Plattform, was die Transparenz erhöht. Weiter bietet Jira diverse Reports um Auswertungen über das Projekt zu fahren.

3.3 Planung

Am Anfang es Projektes haben wir eine grobe Planung zusammengestellt. Dabei haben wir die Phasen und Meilensteine definiert. Während dem Projekt stellten wir immer wieder grössere oder kleinere Abweichungen an der zu Beginn definierten Planung fest. Dieses ist jedoch nicht erstaunlich, da nie absolut korrekt geplant werden kann. Um solche Schwierigkeiten zu handhaben, erstellten wir ein Risikomanagementdokument.

3.3.1 Phasen

1. Inception
    (a) Aufgabenstellung ausarbeiten
2. Elaboration1
    (a) Evaluation der Algorithmen (Bilderkennung)
3. Elaboration2
    (a) Prototyp 1 (In Orthofotos, Out Koordinaten)
    (b) Prototyp 2 MapRoulett
4. Construction1
    (a) Schnittstelle Orthofotos
    (b) Optimierungen durch Strassenverlauf und ähnliches
5. Construction2
    (a) MapRoulette (Tags und Quiz)
    (b) Koordinaten erfassen
6. Transition
    (a) Dokumentation abschliessen
    (b) Challenge auf Maproulette

3.3.2 Meilensteine

1. MS1 Algorithmus für Bilderkennung evaluiert
2. MS2 Prototyp erstellt
3. MS3 Automatisierte Datenverarbeitung
4. MS4 Applikation fertiggestellt
5. MS5 Challenge auf Maproulette

3.3.3 Zeitplanung

Aufwand: 14 Wochen zu 2 * 16 Stunden = **448 Stunden**

Inception	1 Woche
Elaboration1	3 Wochen
Elaboration2	4 Wochen
Construction1	3 Wochen
Construction2	1 Wochen
Transition	2 Wochen

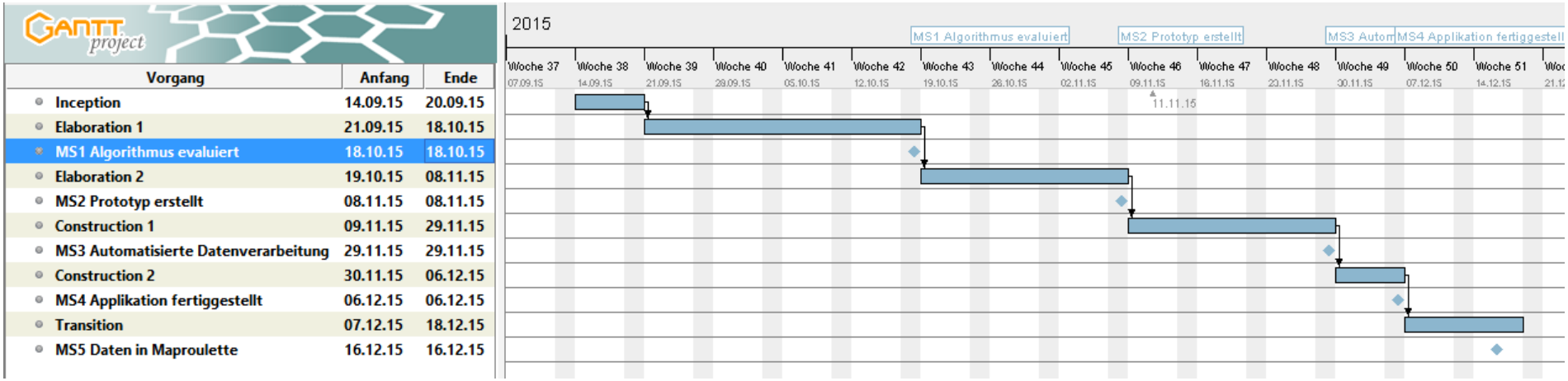

Abbildung 3.4: Gantt Chart

3.4 Risiken

Um den Problemen, die während des Projekts auftreten können entgegenzuwirken, haben wir eine Risiko Analyse durchgeführt. Diese konnte dann bei der Planung eingesetzt werden.

3.4.1 Technische Risiken

Nr	Titel	Beschreibung	maximaler Schaden	Eintrittswahrscheinlichkeit	Gewichteter Schaden	Vorbeugung	Verhalten beim Eintreten
R1	Einarbeitung Python	Python ist den Teammitgliedern teils bekannt, jedoch wurde noch kein grösseres Projekt mit dieser Sprache entwickelt.	40h	10%	4h	Evaluation des Wissensstandes	Informationen bei Studenten einholen, die Python gut kennen
R2	Installation OpenCV	Die Installation von OpenCV mit den Contrib Package ist bekanntermassen ein grosse Hürde	16h	50%	8h	Installation mit Tutorials durchführen	Rücksprache mit Felix Morgner
R3	Detektion	Der Algorithmus, der Fussgängerstreifen erkennt, liefert zu schlechte Resultate und kann nicht gebraucht werden.	40h	50%	20h	Analyse diverser Algorithmen in der Evaluation	Gespräch mit Guido Schuster suchen
R4	Download Orthofoto	Download der Orthofotos von Bing oder ähnlichen Quellen ist nicht möglich	50h	30%	15h	Alternativen im Auge behalten	Auf Bildmaterial der HSR zurückgreifen
R5	Software ist zu langsam	Der Download der Orthofotos oder die Detektion kann einige Zeit in Anspruch nehmen.	60h	70%	42h	Konzept für Parallelisierung erarbeiten	Fläche einschränken, -Grössere und mehrere Maschinen verwenden.

Tabelle 3.1: Risiken - Die technischen Risiken wurden zu Beginn des Projektes, wie in der Tabelle ersichtlich, definiert.

3.4.2 Auswertung

R1 Einarbeitung Python Risiko ist nicht eingetreten, die Entwickler hatte keine Mühe mit Python zu arbeiten.

R2 Installation OpenCV Risiko ist in vollem Umfang eingetreten. Die Installation und Kompilation stellte sich als äusserst Trickreich heraus.

R3 Detektion Die Detektion stellte die Hauptaufgabe unserer Arbeit dar, ist jedoch gleichzeitig eine der Risiko reichsten, da Bilderkennung ein nicht ganz triviales Problem ist. Das Implementieren und Testen der verschieden Algorithmen war sehr zeitaufwändig, was dazu führte, dass auch dieses Risiko eingetroffen ist.

R4 Download Orthofoto Während des Projektes, wechselten wir mehrmals die API für den Download, was sich auch hier auf einen erhöhten Aufwand auswirkte.

R5 Software ist zu langsam Durch den Einsatz von RQ in Kombination mit Redis wurde diesem Risiko Einhalt geboten.

Nr	Titel	Schaden
R1	Einarbeitung Python	0h
R2	Installation OpenCV	20h
R3	Detektion	60h
R4	Download Orthofoto	12h
R5	Software ist zu langsam	0h
Total		92h

Tabelle 3.2: Risikoauswertung

3.5 Soll-Ist-Zeit-Vergleich

Während des ganzen Projektes haben wir in Jira vor jeder Phase die jeweiligen Tasks definiert und den Aufwand dazu geschätzt (Soll). Weiter haben wir dann auch die effektive Zeit zu den Tasks erfasst (Ist).

3.5.1 Inception

Start:	14.09.2015	Ende:	23.09.2015

3.5.2 Elaboration1

Start: 23.09.2015 Ende: 19.10.2015

3.5.3 Elaboration2

Start: 19.10.2015 Ende: 04.11.2015

3.5.4 Construction1

Start: 04.11.2015 Ende: 25.11.2015

3.5.5 Construction2

Start: 04.11.2015 Ende: 11.11.2015

3.5.6 Transition

Start: 11.12.2015 Ende: 18.12.2015

3.5.7 Übersicht

Phase	Soll	Ist	Differenz
Inception	36.00	45.50	9.50
Elaboration1	111.00	150.50	39.50
Elaboration2	98.00	105.75	7.75
Construction1	110.00	122.50	12.50
Construction1	58.00	88.50	30.50
Transition	34.00	52.00	18.00
Total	447.00	564.75	117.75

Tabelle 3.3: Phasen

Schätzen ist wie bekannt, ein relativ schwierige Angelegenheit. So haben wir den Aufwand für die jeweiligen Tasks meist zu tief eingestuft. Stark ins Gewicht fiel die Evaluation und Implementierung des Bilderkennungsalgorithmus.

3.6 Codestatistik

3.6.1 Test Coverage

Test Coverage wurde mit dem Tool **nose** durchgeführt.

Datei	Coverage [%]
src/base/Bbox.py	85
src/base/Node.py	97
src/base/Street.py	92
src/base/Tile.py	98
src/base/TileDrawer.py	24
src/data/MapquestApi.py	100
src/data/MultiLoader.py	94
src/data/StreetLoader.py	100
src/data/TileLoader.py	100
src/detection/BoxWalker.py	100
src/detection/NodeMerger.py	89
src/detection/StreetWalker.py	100
src/detection/deep/Convnet.py	97
src/detection/deep/training/Crosswalk_dataset.py	100
src/detection/deep/training.py	100
src/role/Manager.py	91
src/role/WorkerFunctions.py	70
Durchschnitt	90.5

Tabelle 3.4: Test Coverage

3.6.2 Codezeilen

Die Codezeilen wurden mit Hilfe von **CLOC** [**?**] ausgezählt.

Sprache	Dateien	Zeilen
Python	43	2045

Tabelle 3.5: Codezeilen

3.7 Rational Unified Process (RUP)

Das Projekt Extraction of Crosswalks from Aerial Images führten wir mit dem Vorgehensmodell RUP [**?**] aus. Dies ist eine iterative Art um Softwareprojekt anzugehen. RUP kennt die Phasen:

- Inception, Elaboration, Construction, Transition

Um eine feinere Aufteilung zu haben, unterteilten wir die Elaboration, sowie die Construction in je zwei Hälften.

Fazit

Wir hatten Mühe unserem Projekt ein RUP Stempel auf zu drücken, da wir bei der Evaluation der Algorithmen immer wieder grosse Teile unserer Lösung überdenken und auf den Kopf stellen mussten. Es wurde viel geschriebener Code wieder über den Haufen geworfen. Weiter war es schwierig den Phasen gerecht zu werden, nur mit vielen Überstunden konnte am Ende der Elaboration ein Prototyp präsentiert werden, der ein angemessenes Resultat lieferte. Im Nachhinein hätte sich ein agilieres Vorgehensmodell wie Scrum oder ähnlich vielleicht besser bewährt.

Kapitel 4

Softwaredokumentation

4.1 Installation

4.1.1 Redis

Die Installation wurde auf einem Ubuntu Server durchgeführt, welcher das Paketverwaltungssystem Advanced Packaging Tool (APT) verwendet. Die im Anschluss aufgeführten Befehle werden in einer Shell ausgeführt. Falls Probleme auftreten, bietet Redis ein Quick Start Dokumentation an.

Installaiton

```
# sudo apt-get install redis-server
```

Konfiguration

```
# redis-cli -p 40001
# CONFIG SET requirepass "crosswalks"
# redis-cli AUTH crosswalks
```

Starten

```
# redis-server --port 40001
```

4.1.2 Keras

Keras ist die Bibliothek, mit der das neuronale Netz betrieben wird. Die Installation hier muss nur durchgeführt werden, um das Projekt weiterzuentwickeln.

Info: Der Stand von Keras, der wir während dieser Arbeit verwendet haben, ist auf der CD zu finden.

Installation Theano

```
# apt-get update
# apt-get install -y git libopenblas-dev python-dev python-pip
python-nose python-numpy python-scipy
# pip install git+git://github.com/Theano/Theano.git
```

Installation Keras

```
# apt-get install -y libhdf5-dev python-h5py python-yaml
# pip install --upgrade six
# cd /root
# git clone https://github.com/fchollet/keras.git
# cd keras
# python setup.py install
```

4.1.3 Docker

Die Installation unserer Applikation beinhaltet diverse Abhängigkeiten, welche für die Installation einerseits viel Zeit in Anspruch nehmen und anderseits auch nicht wirklich trivial sind..

Entscheid 9. Docker

Deshalb haben wir ein Docker Image ersellt, das auf Dockerhub [**?**] frei zur Verfügung gestellt wird und somit den DevOps Prozess massiv vereinfach.

Installation

Bei der Installation von Docker ist zu beachten, dass die Anwendung nur auf 64-Bit Maschinen läuft. Weiter wurden die nachfolgenden Befehle auf Ubuntu durchgeführt und variieren deshalb je nach Betriebssystemen.

Vorarbeit

```
# sudo apt-key adv --keyserver hkp://p80.pool.sks
-keyservers.net:80 --recv-
keys 58118E89F3A912897C070ADBF76221572C52609D
# echo 'deb https://apt.dockerproject.org/repo ubuntu-precise main'
>> /etc/apt/sources.list.d/docker.list
# apt-get update
# apt-cache policy docker-engine
# sudo apt-get install linux-image-extra-$(uname -r)
```

Docker installieren

```
# sudo apt-get install docker-engine
```

Starten

```
# sudo service docker start
# sudo docker run hello-world
```

4.2 Benutzerhandbuch

4.2.1 Suche der Fussgängerstreifen

Um die Suche der Fussgängerstreifen durchzuführen, muss eine Redis Datenbank zur Verfügung stehen. Weiter muss auf den Rechnern, die als Jobworker tätig sind, Docker installiert sein. Die Installationen sind in folgenden Abschnitten aufgeführt:

- Redis: Abschnitt 4.1.1 auf der Seite 46
- Docker: Abschnitt 4.1.3 auf der Seite 47

Einführung

Wir haben unsere Applikation in drei Rollen aufgeteilt:

- Manager
  - Unterteilt eine grosse Bounding Box in kleinere Boxen mit einer Höhe und Breite von jeweils 2 Kilometern und stellt dies als Jobs in die Queue.
- Jobworker
  - Arbeite die Jobs der Queue ab.
  - Gefundene Fussgängerstreifen , welche noch nicht in OpenSteetMap erfasst sind, werden als Job Resultat in die Queue gestellt.
- Resultworker
  - Schlussendlich werden die Resultate zusammen getragen und in ein JSON File geschrieben.

Dieser Ablauf ist genauer beschrieben unter dem Abschnitt **??** auf der Seite **??**

Anwendung

Dank Docker kann unsere Applikation innert Minuten gestartet werden.

Docker Pull

```
# docker pull murthy10/osm-crosswalk-detection
```

Manager

```
# docker run murthy10/osm-crosswalk-detection REDIS_IP_ADDR
--role manager left bottom right top
```

Left, Bottom, Right, Top entsprechen den Koordinaten im WGS84 Format.
Ostschweiz: left=8.361002, bottom=47.166994, right=8.977610, top=47.706676

Jobworker

```
# docker run murthy10/osm-crosswalk-detection REDIS_IP_ADDR
--role jobworker
```

Jobworker können auf beliebig vielen Rechnern gestartet werden.

Resultworker

```
# docker run murthy10/osm-crosswalk-detection REDIS_IP_ADDR
--role resultworker
```

Die Resultate werden in der Datei crosswalks.json gespeichert. Diese findet man im Verzeichnis in dem der Resultworker gestartet wurde.

Struktur JSON

Die Struktur der crosswalks.json Datei ist folgendermassen aufgebaut:

```
{
        "crosswalks":
        [
                {"latitude": 47.0, "longitude": 8.3},
                {"latitude": 48.0, "longitude": 8.4}
        ]
}
```

4.2.2 Daten visualisieren

Um das Resultat des Erkennungsalgorithmus zu visualisieren bot sich das Tool CartoDB [**?**] an. Dieses ermöglicht Daten in diversen Formaten hochzuladen und auf einer Karte anzuzeigen.

Vorgehen

1. Daten (crosswalk.json) mit den Spalten latitude und longitude. in CSV Format umwandeln
2. CSV Datei in CartoDB als neues Dataset hochladen.

Struktur CSV

Die Struktur der CSV Datei gliedert sich wie folgt:

```
        latitude,       longitude
        47.0,           8.3
        48.0,           8.4
```

Daten selektieren

CartoDB ermöglicht eine Selektion der Daten. So kann zum Beispiel ein Polygon selektiert werden.

```
SELECT * FROM dataset WHERE ST_WITHIN(
        ST_Transform(the_geom, 4326), ST_SetSRID(
        ST_GeomFromGeoJSON('{ "type": "Polygon",
        "coordinates": [
          [ [8.81429672241211,
             47.22971054221563],
            [8.8385009765625,
             47.22877798599878],
            [8.820991516113281,
             47.2185187846731],
            [8.81429672241211,
             47.22971054221563]
          ]
        ] }'),
   4326))
```

4.2.3 Challenge erstellen

Nach dem die Fussgängerstreifen detektiert wurden und die Datei crosswalks.json erstellt wurde, muss dies noch in ein passendes Format für MapRoulette gebracht werden. Dazu haben wir ein Python Skript geschrieben, welches aus jedem gefundenen Fussgängerstreifen einen Task generiert.

Anwendung

Für eine Challenge benötigt es die Datei challange.json, welches die Challenge beschreibt und ein zweite Datei tasks.json, in dem sich die Tasks befinden.

Tasks generieren

```
# python TaskGenerator.py crosswalks.json
```

Challenge publizieren

```
# curl -u devuser:mylittlesony -vX
  POST http://maproulette.org/api/admin/challenge/
  crosswalk-detection/tasks -d @tasks.json
  --header "Content-Type: application:json"
```

Tasks publizieren

```
# curl -u devuser:mylittlesony -vX
  POST http://maproulette.org/api/admin/challenge/
  crosswalk-detection -d @challenge.json
  --header "Content-Type: application:json"
```

Als Hilfestellung zum Erstellen von MapRoulette Challanges gibt es ein empfehlenswertes Tutorial [**?**].

4.2.4 Keras Training

Für das Training des Neuronalen Netzes steht ein eigenes Docker Image [**?**] auf Docker Hub bereit. Das Image basiert auf einem offiziellen Nvidia Cuda Image und ist fähig mit einer Nvidia Grafikkarte zu arbeiten. Die Grafikkarte wird mit Hilfe des nvidia-docker Projekts geladen. CUDA muss dabei auf dem Host Rechner installiert sein.

Keras Image

Download des nvidia-docker Projekts

```
# git clone https://github.com/NVIDIA/nvidia-docker.git
# cd nvidia-docker
```

Herunterladen des Images

```
# docker pull sebu/keras_cuda
```

Start des Containers und mount der Grafikkarte Nummer 0

```
#GPU=0 ./nvidia-docker run -i -t sebu/keras_cuda /bin/bash
```

Achtung: Die Keras Bibliothek entwickelt sich ständig weiter. Auch die Interfaces der populärsten Klassen können sich ändern und haben sich auch schon während diesem Projekt verändert! In diesem Keras Docker Image ist der Stand von Keras installiert, mit dem wir gearbeitet haben. Keras bietet leider keine Versionierung an. Der von uns verwendete Stand ist auch auf der mitgelieferten CD erhältlich.

Ein Beispiel für das Training eines Neuronalen Netzes kann in examples/ConvnetTrainer.py und in den Keras eigenen Examples eingesehen werden.

Anhang A

Inhalt der CD

Der Inhalt der CD glieder sich folgendermassen:

```
CD
├── Studienarbeit.pdf
├── 0_AUFGABE
│   └── Original Aufgabenstellung
├── 1_CODE
│   └── GitHub Repository
├── 2_DOKUMENTATION
│   ├── Protokolle
│   └── Poster
├── 3_RESULTAT
│   └── Geodaten
└── 4_ANHANG
    └── Keras
```

Anhang B

Eigenständigkeitserklärung

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt sind oder mit dem Betreuer schriftlich vereinbart wurden.
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben.
- dass wir keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt haben.

Ort, Datum:

Rapperswil, 10. Mai 2016

Namen, Unterschriften:

Severin Bühler

Samuel Kurath