HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

INSTITUTE FOR INTERNET
TECHNOLOGIES AND APPLICATIONS

Endpoint Compliance Monitoring based on Software Identification Tags

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrssemester 2014

Autoren:	Danilo Bargen, Christian Fässler, Jonas Furrer
Betreuer:	Prof. Dr. Andreas Steffen
Projektpartner:	ITA, HSR
Experte:	Dr. Ralf Hauser
Gegenleser:	Prof. Hans Rudin
Abgabedatum:	13. Juni 2014

Inhaltsverzeichnis

Teil I

Einleitung

1 Abstract

Das amerikanische National Cybersecurity Center of Excellence[1] schlägt in einem Entwurfsdokument vor, durch eine kontinuierliche Überwachung der installierten Software auf Client-Systemen (Desktop PCs, Laptops, Tablets, Smartphones, etc) die Gefahr von Cyberattacken zu minimieren. Das Software Asset Management soll über Software Identification (SWID) Tags erfolgen, die durch die ISO/IEC 19770-2[1] Norm international standardisiert sind.

Im Rahmen dieser Arbeit wurde eine Client Komponente für Linux Systeme entwickelt, welche SWID Tags gemäss dem aktuellen ISO 19770-2 Draft aus den installierten Softwarepaketen generiert. Implementiert wurde die Unterstützung für drei weit verbreitete Paketmanager. Die Architektur des Generators ist modular aufgebaut, so dass die Unterstützung für weitere Umgebungen einfach hinzugefügt werden kann. Die generierten SWID Tags werden einem sogenannten TNC Policy Manager übermittelt.

Die Trusted Computing Group[2] hat ein offenes Framework für das aktive Monitoring von Endgeräten mit dem Namen «Trusted Network Connect» (TNC) entwickelt. In einer Vorgängerarbeit wurde bereits eine erste Version eines TNC Policy Managers implementiert, welcher nun für die Verwaltung der SWID Tags erweitert wurde.

Über frei definierbare Policies können Clients mit bestimmten Softwarepaketen automatisch aus dem Netzwerk ausgeschlossen oder zu Software Updates gezwungen werden. Dadurch wird ein umfassendes Software Asset Management ermöglicht, welches sich in bestehende Geschäftsprozesse integrieren lässt.

Der strongTNC Policy Manager war bislang über eine gemeinsame Datenbank eng mit dem strongSwan TNC Server gekoppelt. Diese Kopplung wirkt sich negativ auf die Wartbarkeit und die Interoperabilität mit Drittsystemen aus. Aus diesem Grund wurde im Rahmen dieser Arbeit ein Konzept erarbeitet, welches eine serviceorientierte Architektur vorsieht und so eine Trennung ermöglicht. Bestandteil des Konzeptes ist die Definition einer REST Schnittstelle, sowie ein Proof of Concept, welcher diese Schnittstelle für die Verwaltung der SWID Tags bereits erfolgreich implementiert. Da das TNC Framework bisher keine einheitlichen Schnittstellen für die Kommunikation zwischen dem Policy Manager und Umsystemen beinhaltet, wird das erarbeitete Konzept der Trusted Computing Group als Vorschlag unterbreitet.

[1] `http://csrc.nist.gov/nccoe/`
[2] `http://www.trustedcomputinggroup.org/`

2 Management Summary

Ausgangslage

Das amerikanische National Cybersecurity Center of Excellence schlägt in einem Entwurfsdokument vor, durch eine kontinuierliche Überwachung der installierten Software auf Client-Systemen (Desktop PCs, Laptops, Tablets, Smartphones) die Gefahr von Cyberattacken zu minimieren. Das Software Asset Management soll über Software Identification (SWID) Tags erfolgen, welche durch die ISO/IEC 19770-2[1] Norm international standardisiert sind.

Die Trusted Computing Group (TCG) hat ein offenes Framework für das aktive Monitoring von Endgeräten mit dem Namen Trusted Network Connect (TNC) entwickelt. In einer Vorgängerarbeit wurde bereits eine erste Version eines TNC Policy Managers implementiert, welcher nun für das Software Asset Managements mittels SWID Tags erweitert werden soll.

Zusätzlich soll ein Client für Debian und Ubuntu basierte Systeme erstellt werden, welcher SWID Tags aus den Informationen des Paketmanagers generiert.

Vorgehen / Technologien

Der SWID Generator wurde vollständig in Python entwickelt. Es wurde darauf geachtet, dass keine Abhängigkeiten zu externen Bibliotheken bestehen. Dadurch kann sichergestellt werden, dass der Client problemlos zusammen mit dem strongSwan VPN Client verteilt werden kann. Die Analyse des bestehenden strongTNC Policy Managers hat gezeigt, dass eine enge Kopplung mit dem strongSwan TNC Server besteht. Zudem ist aufgefallen, dass der bereits bestehende Code Qualitätsmängel aufweist. Diese beiden Punkte wurden neben der Integration der SWID Tag Verwaltung zum zentralen Bestandteil dieser Arbeit.

Ergebnis

Die Architektur des SWID Generators wurde modular ausgelegt, so dass die Unterstützung für weitere Paketmanager einfach hinzugefügt werden kann. Zum jetzigen Zeitpunkt werden die

drei weitverbreitetsten Paketmanager (DPKG, RPM, Pacman) unterstützt. Um das Deployment zu erleichtern, wurde der Generator in den Python Package Index aufgenommen.

Für die Entkopplung des strongTNC Policy Managers gegenüber den Umsystemen wurde ein API Konzept ausgearbeitet, welches eine serviceorientierte Architektur vorsieht und bereits erfolgreich für die neu integrierten Komponenten umgesetzt wurde. Das Konzept soll nun als Vorschlag zur Umsetzung einer TNC Schnittstelle der Trusted Computing Group unterbreitet werden.

Mittels Continuous Integration, Refactoring und zusätzlichen Integrations- und Unittests konnte die Codequalität von strongTNC messbar verbessert werden.

3 Aufgabenstellung

Institute for Internet Technologies and Applications

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
FHO Fachhochschule Ostschweiz

Bachelorarbeit 2014

Endpoint Compliance Monitoring based on Software Identification Tags

Studenten: Danilo Bargen, Christian Fässler, Jonas Furrer

Betreuer: Prof. Dr. Andreas Steffen

Ausgabe: Montag, 17. Februar 2014

Abgabe: Freitag, 13. Juni 2012

Einführung

Das amerikanische National Cybersecurity Center of Excellence [1] schlägt in einem Entwurfsdokument [2] vor, über eine kontinuierliche Ueberwachung der installierten Software auf Client-Systemen (Desktop PCs, Laptops, Tablets, Smartphones, etc.), die Gefahr von Cyberattacken zu minimieren. Das Software Asset Management soll über Software Identification (SWID) Tags erfolgen, die auf der 2014 Revision der ISO/IEC 19770-2 Norm basieren sollen.

Die Trusted Network Connect (TNC) Funktionalität der strongSwan Open Source VPN Software kann Windows, OS X, Android und Linux Clients nach SWID Tags durchsuchen und dieses Inventar an einen zentralen TNC Server übermitteln [3]. Neu soll ein SWID Generator Tool für Linux Clients erstellt werden, das SWID Tags für alle installierten Linux Pakete generieren kann, die durch einen Paket Manager verwaltet werden.

Die vorgeschlagene Arbeit soll weiter auf dem bestehenden strongTNC Policy Management Tool [4] aufbauen, das in der Skriptsprache Python geschrieben ist und das Django Web-Framework benutzt. Die von den Clients gelieferten SWID Tags sollen zusammen mit Metadaten, die aus den XML-codierten Tags extrahiert werden, in der TNC Datenbank abgespeichert und verwaltet werden. Über eine geeignete Benutzerführung soll visualisiert werden können, welche Version eines Software Paketes in welchem Zeitraum auf welchen Clients installiert war.

Aufgabenstellung

- Einarbeiten in den ISO/IEC 19770-2:2014 SWID Tag Draft Standard.
- Erstellen eines SWID Generators in der Skriptsprache Python, der ISO/IEC 19770-2:2014 SWID Tags für einzelne oder alle Linux Pakete erzeugen kann, die durch den **dpkg** Paket Manager (Debian, Ubuntu, etc.) verwaltet werden.
- Es soll über eine Konfigurationsoption des SWID Generators möglich sein, die Pfadnamen aller Dateien, die durch ein Linux Paket installiert werden in das SWID Tag aufzunehmen.
- **Optional**: Unterstützung weiterer Paket Manager wie z.B. **rpm** (Fedora, RedHat, SuSE) oder **pacman** (Arch Linux).

- Erstellung eines Architektur-Konzepts für die Erfassung und Verwaltung von SWID Tags über eine Erweiterung des bestehenden strongTNC Policy Manager Tools.
- Bidirektionale Verknüpfung der SWID Tags mit den bestehenden strongTNC Objekten *Devices*, *Sessions*, *Packages* und *Files*.
- Implementation der SWID Funktionalität als strongTNC Erweiterung auf der Basis Python/Django.
- Korrektur diverser Bugs in der bestehenden strongTNC Applikation.
- Aufteilung der strongTNC Zugriffsrechte auf einen **read-only** User und einen **admin** User mit erweiterter write/update Berechtigung.

Links

[1] National Cybersecurity Center of Excellence
http://csrc.nist.gov/nccoe/The-Center/Mission/Strategy.html

[2] Continuous Monitoring Building Block - Software Asset Management
http://csrc.nist.gov/nccoe/Building-Blocks/Continuous%20Monitoring%20Building%20Block%20-%20Software%20Asset%20Management.pdf

[3] strongSwan TNC Server Funktionalität
http://www.strongswan.org/tcg/TCC_Orlando_2013.pdf

[4] strongTNC Policy Manager
https://github.com/strongswan/strongTNC

Rapperswil, 17. Februar 2014

Prof. Dr. Andreas Steffen

4 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.

Rapperswil, 13. Juni 2014

Danilo Bargen

Christian Fässler

Jonas Furrer

Teil II

Technischer Bericht

5 Einleitung

5.1 Vorbemerkungen

Bei diesem Dokument handelt es sich um die Bachelorarbeit von Danilo Bargen, Christian Fässler und Jonas Furrer, erstellt an der Hochschule für Technik Rapperswil (HSR) im Studiengang Informatik. Betreut und begleitet wurde die Arbeit durch Prof. Dr. Andreas Steffen und Tobias Brunner, Institut für Internettechnologien und Applikationen (ITA). Die Inhalte dieser Arbeit wurden von den Studenten zu gleichen Teilen erarbeitet.

5.2 Zweck

Der Technische Bericht beschreibt den Aufbau der erarbeiteten Lösung und erläutert Entscheidungen betreffend Design und Vorgehen.

5.3 Einführung

Die Infrastruktur und die Organisation von Netzwerken wird immer komplexer. Neue Technologien und Trends müssen in bestehende Umgebungen integriert werden, beinahe täglich tauchen es neue Herausforderungen auf, die gemeistert werden müssen.
Gerade im Bereich der Netzwerksicherheit ist es wichtig, die aktuellen Entwicklungen nicht aus den Augen zu verlieren. Seit längerem ist «Bring your own device» (BYOD) ein Thema, welches mit Netzwerksicherheit schwer zu vereinbaren ist. Virtuelle Netzwerke mit integriertem «Trusted Network Connect» (TNC) bieten eine gute Grundlage, um zu kontrollieren, welche Geräte Zugang zu einem Netzwerk erhalten und welche nicht. Eine wünschenswerte Ergänzung ist die kontinuierliche Überwachung der installierten Software eines Gerätes. Dies schlägt das National Cybersecurity Center of Excellence (NCCoE) in einem Entwurfsdokument vor. Laut dem NCCoE sind die durch die ISO international standardisierten Software Identification (SWID) Tags für diesen Zweck geeignet.
Nun soll die Unterstützung für SWID Tags in vorhandene TNC Implementationen integriert werden, um dadurch die Sicherheit von Netzwerken zu erhöhen und sich den Herausforderungen der Zukunft besser stellen zu können.

6 Analyse

6.1 Ist-Situation

In der Vorgängerarbeit «Cygnet» von Stefan Rohner und Marco Tanner[2] wurde der Policy Manager «strongTNC» implementiert. Das Produkt und die Codebasis dienen als Grundlage für diese Arbeit.

Im Bereich des SWID Generators gibt es zur Zeit noch keine bestehenden Produkte.

6.1.1 Elemente der strongTNC App

Im Folgenden werden die zentralen Elemente der strongTNC Umgebung beschrieben. Die Namen entsprechen jenen der bestehenden Django-Models. Diese Liste soll einen kurzen Überblick geben, eine ausführliche Beschreibung ist in der Dokumentation «Cygnet»[2] zu finden.

Policy Eine Policy beschreibt eine zu überprüfende Richtlinie. Durch einen Policytyp wird festgelegt, was überprüft werden soll (z. B. erlaubte offene Ports, installierte Software). Weiterhin kann festgelegt werden, welche Aktion bei positivem oder negativen Ausgang der Messung (Überprüfung der Richtlinie) ausgeführt werden soll.

Enforcement Policies werden durch Enforcements erzwungen. Ein Enforcement wird einer Gruppe von Geräten zugeteilt. Spezifiert wird zudem, in welchen zeitlichen Abständen die verknüpfe Policy überprüft wird.

Group Geräte können in Gruppen zusammengefasst werden. Gruppen dienen letztendlich dazu, zu bestimmen, welche Enforcements zur Anwendung kommen. Gruppen lassen sich hierarchisch aufbauen.

Session Wenn sich ein Gerät mit einem strongSwan TNC Netzwerk verbindet, wird durch strongSwan eine Session für das verbundene Gerät erstellt. Sämtliche Messungen werden mit dieser Session verknüpft.

Workitem Für jede Policy, die durch ein Enforcement während einer Session geprüft werden muss, wird ein Workitem erstellt. Diese Workitems werden durch die IMVs (Integrity

Measurement Verifiers) von strongSwan abgearbeitet. Die Resultate werden nach den Messungen in den Workitems abgelegt.

Device Ein Device repräsentiert ein Gerät, welches sich in ein strongSwan TNC Netzwerk verbinden kann. Ein Gerät wird durch die Kombination einer eindeutigen Hardware ID sowie dem Betriebssystem (siehe Product) identifiziert. Ein Device ist Mitglied einer oder mehrerer Gruppen; diese Mitgliedschaft bestimmt, welche Policies auf das Gerät angewendet werden.

Product Ein Product entspricht dem Betriebssystem eines Devices. Einem Product können Gruppen zugeteilt werden. Diese Gruppen stellen die Standardgruppen eines Gerätes mit einem bestimmten Betriebssystem dar.

Package Ein Package beschreibt ein Softwarepaket. Packages werden mit Versionen verknüpft.

Version Eine Version ist immer mit einem Package und einem Product verknüpft. Das heisst, für jedes Betriebssystem existiert eine eigene Version, auch wenn die Versionsbezeichnung dieselbe ist. Versionen können global als «blacklisted» und «security patch» markiert werden. Versionen können Messziele einer Policy sein.

Directory Ein Directory kann Bestandteil einer Policy und somit Grundlage einer Messung sein.

File Files dienen als Grundlage für File-Messungen.

FileHash Bei File-Messungen können Hashes einer gewissen Datei erfasst werden, diese werden zusammen mit dem Product des gemessenen Gerätes einem File zugewiesen.

In Abbildung 6.1 ist eine Übersicht der verwendeten Django-Models zu sehen:

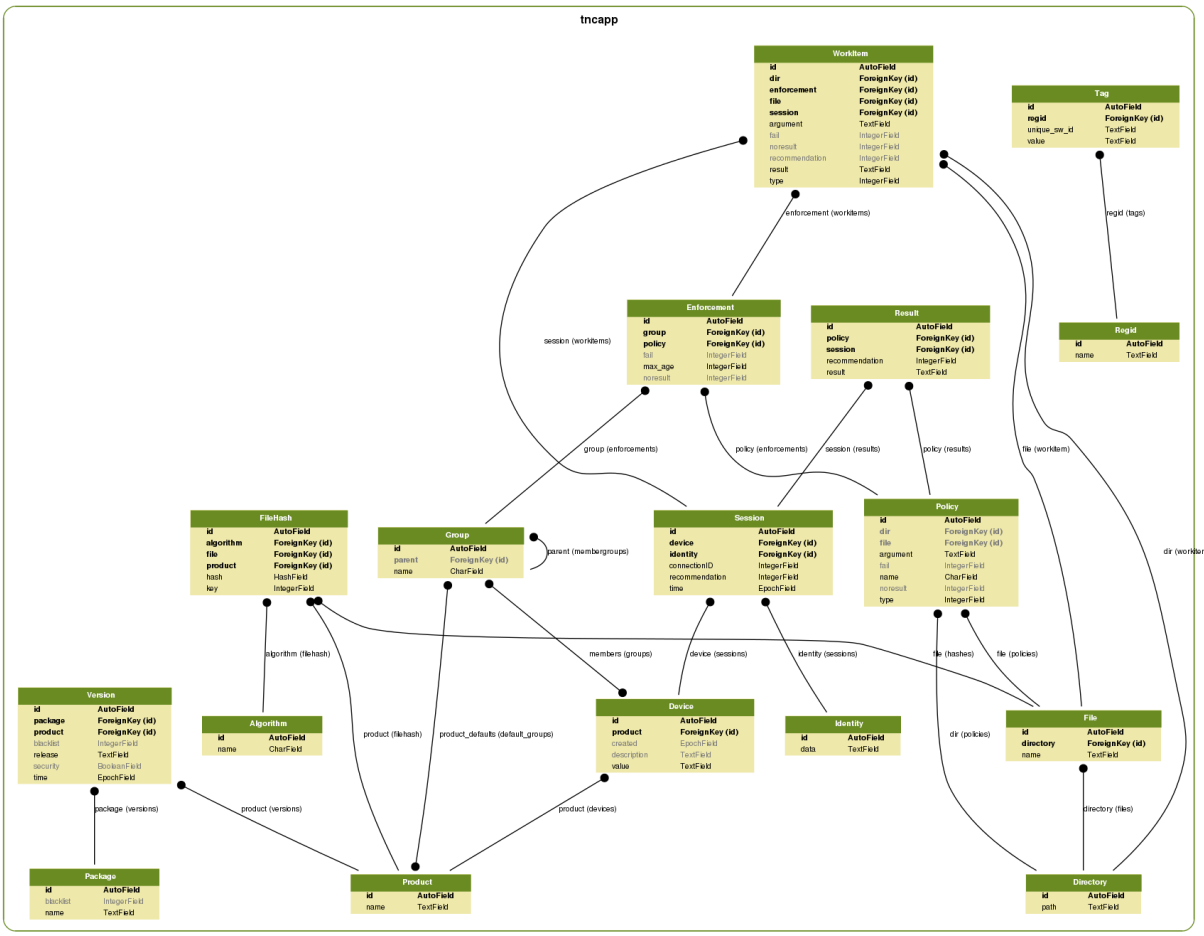

Abbildung 6.1: Ursprünglich vorhandene strongTNC Models

6.1.2 Ablauf einer TNC Messung mit strongSwan und strongTNC

Wenn sich ein Gerät mit einem strongSwan TNC Netzwerk verbindet, wird durch einen Integrity Measurement Verifier (IMV) eine strongTNC Session gestartet. Von diesem wird das Gerät anhand seiner Hardware ID und seines Betriebssystems identifiziert. Falls sich das Gerät zum ersten Mal verbindet, wird es neu erfasst. Anhand der Gruppen, denen das Gerät angehört, werden die relevanten Enforcements gesammelt und daraus Workitems generiert. Diese Vorgänge werden derzeit alle durch den IMV mittels direktem Zugriff auf die Datenbank durchgeführt. Verschiedene IMVs führen anhand der Workitems entsprechende Messungen durch. Die Ergebnisse der Messungen werden durch die IMVs in die Workitems der strongTNC Datenbank

geschrieben. Anhand der abgearbeiteten Workitems fällt der TNC Server eine abschliessende Entscheidung darüber, ob das Gerät in das Netzwerk zugelassen wird oder nicht. Ein IMV schliesst die Session in der strongTNC Datenbank ab und die Resultate der Workitems werden archiviert.

Den Ablauf einer Messung kann man dem Sequenzdiagramm in Abbildung 6.2 entnehmen.

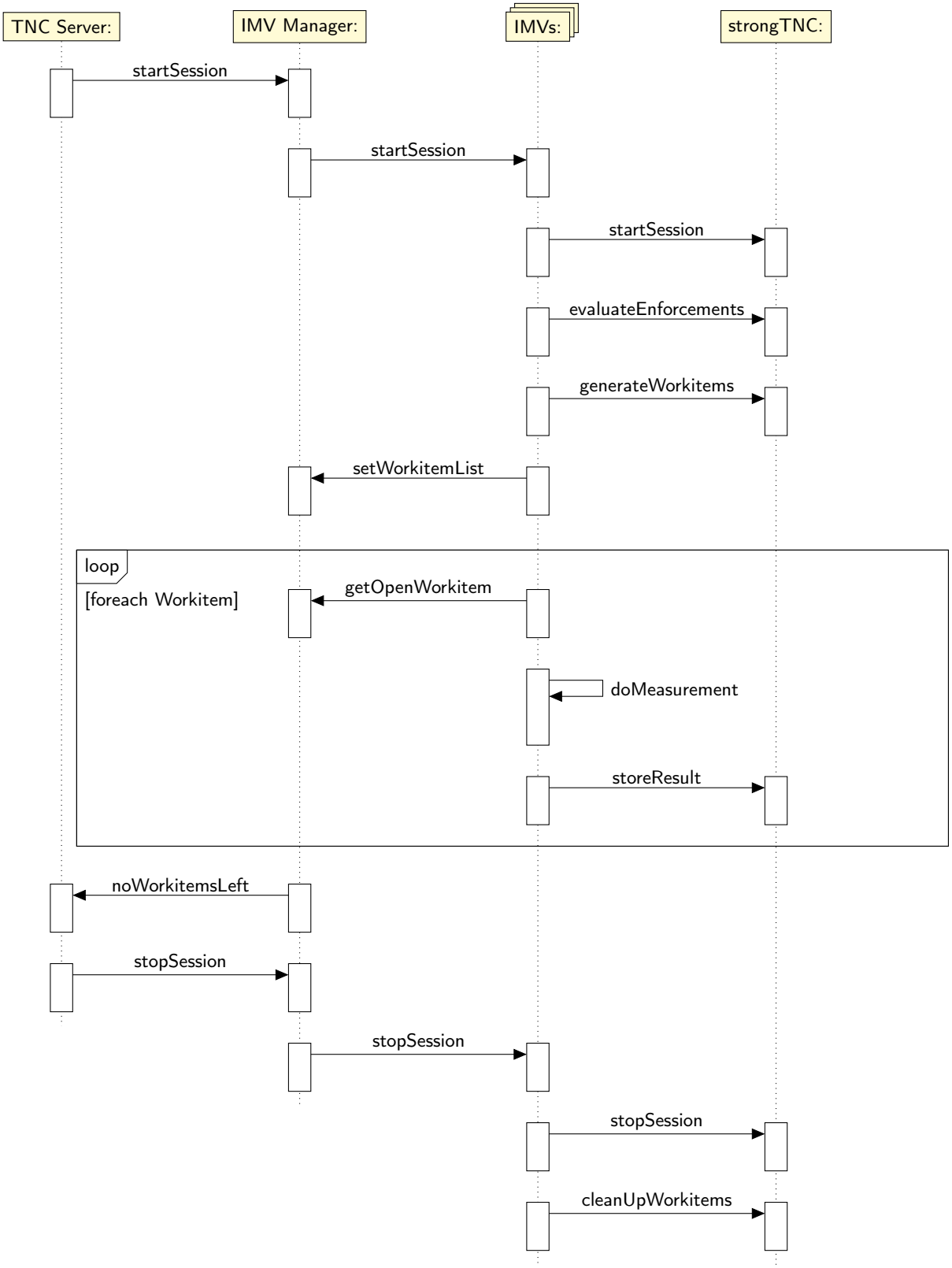

Abbildung 6.2: Sequenzdiagramm, strongSwan TNC Messung

6.1.3 Stand der strongTNC Webapplikation

strongTNC ist eine Implementierung eines TNC Policy Managers. Umgesetzt wurde diese als Webapp mit dem Python basierten *Django Web Framework* in Version 1.5[1]. Die Applikation bietet Unterstützung für das Verwalten von Geräten, Policies und Enforcements und erlaubt die Ausführung von Enforcements mittels hierarchischer Gruppen zu steuern. Stammdaten wie Dateien oder Softwarepakete können eingesehen und teilweise bearbeitet werden. Auch Messresultate werden passend aufbereitet.

Für SWID Tags besteht derzeit keine Unterstützung. Es wurden zwar bereits Views für SWID Tags und Regids erstellt, diese haben jedoch ausser der Auflistung der Datenbankinhalte keine weitere Funktionalität und dienen bisher als Vorschau für zukünftige Features.

Im Frontend der Webapplikation wurden Bootstrap 2.3[2] sowie jQuery 1.9[3] eingesetzt. Grundsätzlich werden die Views als «Master-Detail Ansichten» dargestellt. Als Schnittstelle zur strongSwan TNC Server Infrastruktur dient eine gemeinsam genutzte SQLite[4] Datenbank.

Der Code ist in einer einzelnen Django-App organisiert, es gibt keine thematische Aufteilung in mehrere Apps.

Das Schema der Datenbank ist unabhängig von den Models, diese bilden das aktuelle Schema ab. Um die Datenbank zu initialisieren wird ein separat bereitgestelltes SQL Schema verwendet.

Laut dem Technischen Bericht der Vorgängerarbeit «Cygnet»[2] sollen die Richtlinien des «Style Guide for Python Code»[3], genannt PEP8, eingehalten werden. Ausserdem existieren elf Unit-Tests, welche den bestehenden Python Code zu 21% abdecken.

6.1.3.1 Einschätzung der Ist-Situation

SWID Integration Es existiert noch keine Möglichkeit zur Erfassung und Verwaltung von SWID Tags. Die bestehenden Views können gegebenenfalls als Vorlage für eine solche Integration verwendet werden.

Gemeinsame Datenbank Durch die gemeinsame Datenbank wird die Interoperabilität mit anderen Systemen eingeschränkt und die Wartbarkeit sowie die Ausbaufähigkeit beteiligter Komponenten erschwert. In diesem Fall wird ausserdem SQLite als Datenbank verwendet. SQLite ist nur bei lesendem Zugriff mehrbenutzerfähig, bei schreibendem

[1] `https://www.djangoproject.com/`
[2] `http://getbootstrap.com/2.3.2/`
[3] `http://jquery.com/1.9`
[4] `http://www.sqlite.org/`

Zugriff wird die Datenbank exklusiv gesperrt. Da eine gemeinsam genutzte Datenbank Mehrbenutzerbetrieb impliziert, besteht Handlungsbedarf.

Benutzerschnittstelle Die Benutzeroberfläche wurde noch nicht für den Umgang mit grossen Datenmengen optimiert.Beispielsweise kann das Inventar der Dateinamen in einem Linux System durchaus 40000 Objekte umfassen. Wird die Policy View aufgerufen, werden alle 40000 Einträge geladen.
Diesen Datenmengen wird zur Zeit noch nicht Rechnung getragen. Tendenziell ist der Einsatz einer Endpoint Compliance Lösung in mittelgrossen und grossen Unternehmungen zu erwarten, dementsprechend ist auch die Menge der zu verwaltenden Daten als hoch einzuschätzen.

Django-Apps Django Webapplikationen lassen sich in Module, sogenannte «Apps», aufteilen. Diese Möglichkeit wird allerdings bisher nicht genutzt. Stattdessen befindet sich der gesamte Code in einer einzelnen App namens `tncapp`. Eine solche Unterteilung würde die Wartbarkeit erhöhen.

Datenbank-Schema Bei der Umsetzung von Webapplikationen mit Django ist es gängig, dass das Datenbankschema aus den Models generiert wird. Dadurch muss man sich als Entwickler nicht um Schema-Detailfragen kümmern, sondern kann sich auf die abstrakten Models konzentrieren.
Ein weiterer Vorteil ist, dass das genutzte DBMS beliebig austauschbar ist. Da die Datenbank bei strongTNC jedoch als Schnittstelle dient und nicht ausschliesslich von Django verwendet wird, kann dieses Feature derzeit nicht genutzt werden. Stattdessen bilden die Model-Definitionen den jeweiligen Stand der Datenbank (Abbildung 9.1) ab. Weil die Datenbank nicht aus den Models generiert wird, ist es möglich, dass die Models und das effektive Schema bei Änderungen nicht mehr synchron sind und manuell angepasst werden müssen. Eine «Autorität» für das Schema sowie die Nutzung eines Schema-Migrations-Frameworks würde die Situation stark verbessern.

Codequalität Obwohl sich die Vorgängerarbeit gemäss eigener Aussagen auf die Einhaltung der PEP8-Coderichtlinien[3] beruft [2, S. 79], wurden diese Richtlinien nicht konsequent eingehalten.
Alle Views wurden als Funktionen definiert, obwohl sie häufig über gemeinsame Funktionalität verfügen. Auch die zyklomatische Komplexität der Views ist oft hoch. Ein klassenbasierter Ansatz, eventuell auch in Kombination mit den generischen Views von Django, könnte die Wartbarkeit verbessern.
Die Validierung von übermittelten Formulardaten wurde manuell mittels Javascript gemacht, anstatt das vorhandene Formular-Framework von Django zu verwenden. Auch dies verursacht mehrfach vorhandenen Code und verletzt unnötigerweise das DRY-Prinzip[4, S. 26–27].

Die Durchführung statischer Code-Analyse zeigt Mängel auf, wie beispielsweise unerreichbarer Code, ungenutzte Variablen, mehrfach definierte Funktionen oder verdeckte Python-Standardfunktionen.
Die Templates sind inkonsistent eingerückt, man findet sich im Markup schwer zurecht. Zudem sind viele Elemente – wie z. B. das Suchfeld – mehrfach vorhanden und sollten in ein eigenes Include-Template extrahiert werden.

6.2 Soll-Situation

6.2.1 Aufgabe

Nebst den geforderten Pflichtteilen, der Implementation eines SWID Generators für Linux Systeme und der Integration der SWID Erweiterung in strongTNC, schlagen wir nachfolgende Anpassungen vor, die wir bei vorhandener Kapazität noch umsetzen möchten.

6.2.2 Entkopplung der Datenbank

Wie bereits erwähnt entstehen durch die gemeinsame Nutzung einer SQLite Datenbank einige Probleme. Nachfolgend möchten wir kurz auf die daraus resultierenden Nachteile eingehen.

Mehrbenutzerfähigkeit Die Daten einer SQLite Datenbank können nur von einem Prozess gleichzeitig geändert werden. Da es sich bei der aktuellen Situation aber bereits um eine Multiprozessumgebung handelt ist dieser Einsatz nicht zweckmässig, und kann zu unerwünschten Locks führen.

Verteilung der Komponenten Die Policy Decision Komponente (TNC Server aus dem strongSwan Projekt) kann derzeit nicht getrennt von der strongTNC Webserver Komponente betrieben werden. Dies kann in einem Enterprise Deployment jedoch durchaus erwünscht sein, beispielsweise wenn die strongTNC App auf einem Microsoft Webserver betrieben werden soll, während der Policiy Decision Point auf einem Linux System läuft. Zusätzlich kann es aus sicherheitstechnischen Gründen angezeigt sein, den VPN Gateway, den Decision Point und den strongTNC Webserver in getrennten Netzwerksegmenten zu betreiben.

Anbindung an Drittsysteme Die Anbindung an Umsysteme gestaltet sich als kaum realisierbar. Das TNC Framework sieht aber bereits weitere Komponenten, wie beispielsweise IF-MAP Devices vor, die integriert werden könnten. Auch eine Integration in bestehende Geschäftsprozesse (zum Beispiel CMDB) ist nur schwer realisierbar.

Flexiblere Arbeit mit Django Wenn die strongTNC Anwendung ihre Datenbank nicht mit anderen Systemen teilen muss, können alle Datenbank-Abstraktionsfeatures von Django voll ausgenutzt werden. Im Moment muss man sich aktiv um das Datenbankschema kümmern. Eine Entkopplung hätte zur Folge, dass auch «Migrations»[5] verwendet werden können, welche es erlauben Änderungen an den Models automatisiert in die Datenbank einfliessen zu lassen.

[5] `https://docs.djangoproject.com/en/1.7/topics/migrations/`

Aus diesen Gründen möchten wir ein Konzept entwerfen, um die gemeinsame Nutzung einer SQLite Datenbank aufzuheben.

6.2.3 Codequaliät

Um die Entwicklung und mögliche Weiterentwicklung dieses Projektes einfacher zu gestalten, soll die Codequalität verbessert werden. Folgendes sind die Hauptansatzpunkte:

Coderichtlinien Die PEP8 Coderichtlinien sollen im Verlaufe dieses Projektes und auch in Zukunft strikt eingehalten werden.

Django Best Practice Die Möglichkeiten des Django Webframeworks sollen besser genutzt werden, in dem aktuelle Konzepte wie Class Based Views, separierte Apps oder das Form Framework zum Einsatz kommen.

Testing Um die Qualität und Stabilität längerfristig zu gewährleisten, soll die Testabdeckung erhöht werden.

Continuous Integration Um stets über den Stand der Software Informiert zu sein, soll der Code mit Hilfe von Continuous Integration regelmässig und bei jeder Änderung gebuilded und getestet werden.

6.3 Abgrenzung

Folgende Punkte werden von uns als gegeben betrachtet oder sind nicht Bestandteil unserer Arbeit:

Datenbankschema Das Schema wird nicht grundlegend geändert, sondern übernommen wie es ist. Da verschiedene Komponenten auf die Datenbank zugreifen, kann eine Änderung am Schema im schlechtesten Fall eine Anpassung in allen Komponenten mit Datenbankzugriff zur Folge haben.

Benutzerschnittstelle Das Frontend der Webapp wird nicht grundlegend verändert, es werden Korrekturen, Ergänzungen und Anpassungen vorgenommen, jedoch ohne dabei das bestehende Layout grundsätzlich zu ändern.

strongSwan Anpassungen und Ergänzungen seitens strongSwan sind nicht Bestandteil dieser Arbeit. Änderungen in strongTNC, welche Einfluss auf strongSwan haben, werden mit den verantwortlichen strongSwan Entwicklern besprochen.

6.4 ISO Standard 19770-2

Der ISO Standard 19770-2[1] ist der zweite Teil einer Gruppe von ISO Standards zu den Prozessen und Technologien des Software Asset Management. Dieser Teil des Standards beschreibt den Aufbau und die Verwendung von Software Identification Tags und befindet sich zur Zeit noch im Entwurfsstadium («Draft»).

6.4.1 Bestandteile eines SWID Tags

Nachfolgend eine Zusammenstellung der Bestandteile eines SWID Tags, wie sie in diesem Projekt verwendet werden. Der Standard beschreibt noch weitere Elemente und Anwendungsmöglichkeiten, welche in dieser Arbeit allerdings nicht verwendet werden. Die jeweiligen Listen der Attribute sind nicht vollständig, sondern eine Auswahl, wie sie für diese Arbeit relevant sind. [1, S. 19]

SoftwareIdentity Repräsentation des Wurzelelementes eines SWID Tags. Dieses Element hat Attribute, welche die Anwendung des Tags und dessen Identität, innerhalb seiner Entität, beschreiben.

- **delta** Das `delta` Attribut beschreibt, ob es sich um ein Patch oder eine Modifikation der Software handelt. Dieses Attribut wird von uns nicht verwendet, da bei Linux Systemen üblicherweise keine Patches verteilt werden, sondern gepatchte Softwarepakete. Der Standardwert ist `false`.
- **name** Dieses Pflichtfeld enthält den Namen der Software so wie man üblicherweise darauf verweisen würde.
- **uniqueId** Die `uniqueId` ist eine Kennung, die eine Software innerhalb des Namespaces des Tag Creators eindeutig identifizieren kann. Vorschläge für den Aufbau einer `uniqueId` sind `publisher + product + version` oder eine GUID. Die `uniqueId` ist optional.
- **version** In diesem Attribut wird die Version der Software festgelegt. Das Attribut ist optional und hat einen Standardwert von `0.0`.

Entity Beschreibt die Organisation, die für diesen SWID Tag verantwortlich ist. In einem Tag können theoretisch beliebig viele Entity Elemente enthalten sein. Eine Entity mit der Rolle `tagcreator` (siehe Attribut `role`) ist obligatorisch. Jeder Tag muss mindestens eine Entity mit der Rolle `tagcreator` enthalten. Das `Entity` Element ist ein Kind des `SoftwareIdentity` Elements.

name Name der Organisation, die eine bestimmte Rolle in diesem Tag für sich beansprucht. Dieses Attribut ist ein Pflichtfeld.

regid Die `regid` ist die «Unique registration ID» einer Organisation. Die `regid` ist grundsätzlich wie folgt aufgebaut: `regid.YYYY-MM.<reverse domain name>`, die vierstellige Jahreszahl und der zweistellige Monat ist das Registrierungsdatum der Domain. Für den Aufbau einer `regid` gibt der Standard noch weitere Richtlinien vor, es ist auch definiert wie eine `regid` für Organisationen ohne registrierte Domain auzusehen hat. Dieses Attribut ist optional und wird mit dem Standardwert `invalid.unavailable` befüllt.

role Die Rolle beschreibt, in welcher Beziehung die Organisation zu diesem SWID Tag steht. Das `role` Attribut ist ein Aufzählungstyp mit den möglichen Werten `publisher`, `tagcreator`, `licensor` und ist obligatorisch.

Payload Das `Payload` Element enthält die zu erwartenden Bestandteile dieser Software, wenn sie installiert ist, diese Information ist optional. Das `Payload` Element ist ein Kind des `SoftwareIdentity` Elements und hat keine Attribute.

File `File` Elemente sind Kinder des `Payload` Elements. Sie enthalten Informationen zu den Dateien, die einer Software angehören.

name Name der Datei, ohne Verzeichnis.

location Verzeichnis in dem die Datei zu finden ist, dieses Attribut ist optional.

6.4.2 Wichtige Punkte

- Ein Tag darf nur durch die Organisation modifiziert werden, die ihn initial erstellt hat. Diese Organisation hat mindestens die Rolle des «Tag Creator». Dieser Tag heisst «Primary Tag». [1, S. 6]
- Wenn ein Tag ergänzt werden soll, muss dies mit Hilfe eines «Supplemental Tags» geschehen, da der «Primary Tag» nicht modifiziert werden darf. [1, S. 7]
- Ein SWID Tag wird grundsätzlich als XML Datei abgelegt, diese Datei muss sich im Installationsverzeichnis der repräsentierten Software befinden. Der Tag kann zusätzlich über andere Kanäle wie URIs oder zentrale Verwaltungen zugänglich gemacht werden. [1, S. 10,15]
- Als eindeutige Kennung für einen SWID Tag dient die Kombination von `uniqueId` und `tag_creator_regid`. Diese Kennung wird «software_id» genannt. Es liegt in der Verantwortung des «Tag Creators» dafür zu sorgen, dass seine Tags eindeutig sind. [1, S. 10,16]

- Es ist nicht obligatorisch die Echtheit von SWID Tags zu garantieren; falls eine Echtheitsvalidierung gewünscht oder nötig ist, kann dies durch den Einsatz von der «XML signature syntax»[5] implementiert werden. Echtheitsprüfung ist nicht Bestandteil dieser Arbeit. [1, S. 14]
- Die minimale Information, die ein valider SWID Tag enthalten muss, ist `Entity.role` und `SoftwareIdentity.name`. Bei der Rolle muss es sich um die Rolle des «Tag Creators» handeln. Das empfohlene Minimum besteht aus folgenden Informationen:
  - `SoftwareIdentity`
    - `name`
    - `tagVersion`
    - `uniqueId`
    - `version`
    - `versionScheme`
  - `Entity`
    - `name`
    - `regid`
    - `role`

  [1, S. 16]

6.4.3 Probleme

Bei der Analyse und der Implementation des ISO Standard 19770-2 sind einige Punkte aufgetaucht in denen sich der Standard widerspricht, etwas unklar ist oder Schwierigkeiten bei der Implementation verursacht.

6.4.3.1 Keine garantierte Identität

Die eindeutige Kennung eines SWID Tags setzt sich aus der `regid` des «Tag Creators» und der `uniqueId` zusammen, da jedoch diese beiden Attribute als optional definiert werden, garantiert der Standard keine eindeutige Identität für jeden SWID Tag.

> *(. . .) so it is up to each tag creator to ensure each of their tags is unique.*
>
> *ISO 19770-2 [1, S. 16]*

Da der «Tag Creator» für die Eindeutigkeit seiner Tags verantwortlich ist, muss man davon ausgehen, dass ein erhaltener Tag eindeutig identifizierbar ist. Das Problem bei Tags mit wenig

Informationen liegt allerdings beim Tag Konsumenten:

> *A conforming consumer shall not reject any conforming SWID tag document.*

ISO 19770-2 [1, S. 4]

Vom Tag Ersteller wird lediglich erwartet, dass die produzierten Tags standardkonform sind:

> *A conforming producer shall be able to produce SWID tag documents conforming with this part of ISO/IEC 19770.*

ISO 19770-2 [1, S. 4]

Diese Anforderung ist bereits erfüllt, wenn der Name der Software und der Organisation, welche den Tag erstellt hat, vorhanden sind.

In dieser Arbeit ist dieses Problem dadurch entschärft, dass der SWID Generator die einzige Tag creator Instanz ist und damit eindeutige Tags kreiert. Sobald jedoch mehrere Tag creators vorhanden sind, und die Eindeutigkeit nicht mehr zentral geregelt ist, können Probleme entstehen. Des Weiteren muss davon ausgegangen werden, dass es im Interesse des Tag Erstellers liegt, dass seine SWID Tags nicht Ursache eines Konfliktes werden.

6.4.3.2 SWID Tag XML Dateien

> *SWID tag data is stored in an XML file and shall be located on a devices file system in the same file directory as the application they represent.*

ISO 19770-2 [1, S. 10]

Diese Anforderung ist bei Linux Systemen nicht direkt umsetzbar, da Softwarepakete unter Linux oft nicht nur in einem Verzeichnis installiert werden, sondern Dateien in verschiedene Verzeichnisse kopieren.

In dieser Arbeit wird diese Anforderung nicht beachtet. Der SWID Generator kreiert Tags dynamisch aus den Informationen des Linux Paketmanagers und liefert diese direkt auf die Standardausgabe der Konsole. Die SWID Tags werden nicht in einer Datei gespeichert.

6.4.3.3 Doppelpunkte in der UniqueId

Der SWID Generator setzt die `uniqueId` aus dem Betriebsystemnamen, der Prozessorarchitektur, dem Paketnamen sowie der Versionsnummer zusammen: `<OS>_<Arch>_<Package>_<Version>`. Versionen unter der Linux Distribution «Ubuntu» enthalten oft Sonderzeichen wie Doppelpunkte oder Plus-Zeichen. Anhand der folgenden Aussage aus dem Standard kann man darauf schliessen, dass diese Zeichen in einer `uniqueId` nicht erlaubt sind.

> *(. . .) may be either a GUID, or any reference unique for the tag_creator_regid. The unique_id shall follow the restrictions for URI character use as specified in IETF RFC 3986, section 2, characters.*

ISO 19770-2 [1, S. 13]

Die `uniqueId` unterliegt den Restriktionen einer URI. Für diese gilt: ein Doppelpunkt ist kein unerlaubtes Zeichen jedoch ein reserviertes:

```
reserved = gen-delims / sub-delims
gen-delims = ":" / "/" / "?"/ "#" / "[" / "]" / "@"
sub-delims = "!" / "\$" / "&" / "'" / "(" / ")" / "*" /"+" / "," / ";" / "="
unreserved = ALPHA / DIGIT / "-" / "." / "_ " / "~"
```

IETF RFC 3986[6]

Die `uniqueId` kann als `href` Attribut des `Link` Tags, in Form einer URI mit einem `swid:` Prefix verwendet werden. Daraus folgt, dass die `uniqueId` zum «Authority» Teil einer URI gehört und somit keine Doppelpunkte enthalten darf.

In dieser Arbeit werden daher Zeichen, welche in URIs als reserviert gelten, in der `uniqueId` durch eine Tilde ($\sim$) ersetzt.

7 Vorgehen

Für die Durchführung dieses Projekts haben wir uns für eine agile, iterative Vorgehensweise entschieden. Für ein Team von drei Personen und einer Projektdauer von rund 17 Wochen haben wir uns für ein Subset des Scrum Vorgehensmodelles entschieden, dabei waren die drei Säulen von Scrum von zentraler Bedeutung:

Transparenz Durch die Verwendung von Github zur Sourcecodeverwaltung und auch für das Issuetracking, ist der aktuelle Fortschritt und allfällige Hindernisse im Verlauf des Projektes stets sichtbar, öffentlich und nachvollziehbar festgehalten.

Überprüfung Das Resultat dieser Arbeit wird zeitnah nach ihrem Abschluss an einer internationalen IT Security Konferenz präsentiert. Um zu diesem Zeitpunkt einen stabilen Stand zu erreichen, haben wir uns entschieden, regelmässig den aktuellen Stand unserer Arbeiten mittels Github Pull Requests in das offizielle strongTNC Repository zu integrieren. Dadurch wird sichergestellt, dass die Funktionalität von den Betreuern stets überprüft und getestet werden kann und die Feedback Roundtrips minmal gehalten werden.

Anpassung Da das Produkt bereits im Einsatz ist, werden bei der Bedienung Probleme und neue Anforderungen entdeckt. Dadurch, dass die Anforderungen an das Produkt nicht ein für alle Mal festgelegt, sondern laufend neu bewertet und angepasst werden, entsteht ein Produkt, welches den Ansprüchen der Benutzer besser entspricht.

7.1 Arbeitsweise und Hilfsmittel

Wir haben uns für folgende Artefakte[7] entschieden:

Definition of Done Wir haben eine Checkliste von Aktivitäten definiert, welche festhält, wie einzelne Aufgaben einheitlich gelöst werden. Damit wird einerseits das Schätzen des Aufwandes bei der Iterationsplanung erleichtert und andererseits festgelegt, wann ein Task als erledigt betrachtet werden kann. Auch Qualitätsmassnahmen, wie beispielsweise das Testen und Durchführen von Reviews sind darin enthalten (Abschnitt 15.4).

Scrumboard Sämtliche Tasks werden zur verbesserten Visualisierung auf einem klassischen Scrumboard mit Post-It Zetteln gepflegt, zusätzlich werden alle Task auch im Github Issue Tracker erfasst, um die Nachvollziehbarkeit zu gewährleisten.

Backlog Sämtliche geplanten Features und offenen Bugs werden in einem Backlog gehalten, so ist der Umfang der offenen Arbeit klar ersichtlich.

Iteration Backlog Ebenfalls wird für jede Iteration ein Backlog geführt.

Planning Meeting Zu Beginn jeder Iteration wird ein Planning Meeting durchgeführt, indem die Tasks für die kommende Iteration mit Hilfe von «Planning Poker» eingeplant und geschätzt werden.

Retrospektive Vor jedem Planning Meeting wird eine kurze Retrospektive zur vergangenen Iteration durchgeführt. Die Arbeitszeiten und die Schätzungen, werden durchgesehen und es wird beurteilt, wie genau die Planung der letzten Iteration war und was für die kommende Iteration verbessert werden muss.

Bei der Einarbeitung in die vorhandene strongTNC Umgebung haben wir festgestellt, dass die Architektur und die Codequalität einige Mängel aufweist. Deshalb haben wir uns entschieden, diesem Aspekt genauer zu betrachten und haben entsprechende Massnahmen definiert. Diese sollen die Qualität dieser Arbeit, als auch jene von möglichen Nachfolgearbeiten oder Weiterentwicklungen durch die Community sicherstellen.

Continous Integration Zur fortlaufenden Integration des entwickelten Codes wird Travis CI[1] eingesetzt. Dadurch wird sichergestellt, dass sich stets lauffähiger Code im Master Branch des Repositories befindet oder fehlerhafter Code schnell entdeckt wird. Als zusätzliches Hilfsmittel wurde auf der Webseite des Github Repositories ein Statusindikator integriert, welcher optisch über den aktuellen Buildstand informiert.

Statische Code-Analyse Durch die Integration von Landscape.io[2], einem online Service, wel-

[1] `http://www.travis-ci.org`
[2] `http://landscape.io`

cher statische Code-Analysen durchführt und die Codequalität bewertet, kann sichergestellt werden, dass Code-Smells, welche nicht durch Unit Tests oder während der Entwicklung erkannt werden, trotzdem bemerkt werden. Um immer über den Stand des Codes informiert zu sein, wurde auch für Landscape.io auf Github ein optischer Statusindikator integriert.

Coding-Styleguide Es wurde ein Coding-Styleguide (Abschnitt 15.1) definiert, welcher zusammengefasst folgende Punkte definiert:

- Anwendung und Format von Docstrings zur Inline-Dokumentation von Code
- Code-Style, PEP8
- Future Imports für Vorwärtskompatibilität zu Python 3
- PEP8 Style Checking Konfiguration
- Pytest Testing Framework Konfiguration

Git Guidelines Zum einheitlichen Umgang mit Git wurden Guidelines verfasst (Abschnitt 15.2), welche folgende Punkte beschreiben

- Aussehen von Commit Messages (Sprache, Format, Referenzen)
- Wann und wie «History Rewriting» vorgenommen werden soll
- Wie Pull Requests erstellt werden sollen
- Wie Merging in den Master abläuft

8 SWID Generator

8.1 Requirements

8.1.1 Zweck

Der SWID Generator soll ein Programm für Linux-Systeme sein, welches aus den Informationen aus Paket Management Systemen SWID Tags generiert.

8.1.2 Nichtfunktionale Anforderungen

- Als Implementationssprache wird Python verwendet.
- Es sollen möglichst wenig Abhängigkeiten zu Drittkomponenten wie Libraries oder Frameworks entstehen.
- Die Software soll einfach zu installieren sein, beispielsweise durch Upload in den Python Package Index oder durch Bereitstellen von `.deb`- und `.rpm`-Paketen.
- Als Quelle der Paketinformationen sollen Paketmanager wie DPKG und RPM verwendet werden.

8.1.3 Use Cases

Nachfolgend sind die identifizierten Use Cases des SWID-Generators aufgeführt. Dabei werden die folgenden drei Systemkomponenten berücksichtigt, diese kommunizieren untereinander über die Standardeingabe und -ausgabe.

Abbildung 8.1: SWID Generator Systemkomponenten

8.1.3.1 UC01: Erkennung des Paketmanagers

Akteur	strongSwan IMC
Story	Der Akteur weiss nicht, welcher Paketmanager auf dem Zielsystem verwendet wird. Der SWID Generator soll den verwendeten Paketmanager automatisch erkennen.
Standard Szenario	Der SWID Generator erkennt den System-Paketmanager automatisch.
Alternatives Szenario	Mittels optionalem Parameter kann der zu verwendende Paketmanager definiert werden, die automatische Erkennung wird in diesem Fall nicht durchgeführt.

8.1.3.2 UC02: Software-ID Generierung

Akteur	strongSwan IMC
Story	Der Akteur will die Software-IDs aller installierten Pakete generieren. Das Format dieser IDs folgt dem ISO Draft 19770-2[1] und besteht aus der Regid des Tag Creators sowie der Unique-ID des Tags.
Standard Szenario	Alle Software-IDs werden generiert und auf der Standardausgabe durch Newlines (`\n`) getrennt ausgegeben. Es wird eine Software-ID pro Zeile dargestellt.
Alternatives Szenario	Die Trennzeichen zur Ausgabe der Software-IDs können mittels optionalem Parameter definiert werden.

8.1.3.3 UC03: SWID Tag Generierung

Akteur	strongSwan IMC
Story	Der Akteur will SWID Tags aller installierten Pakete generieren. Das Format dieser XML Dokumente folgt dem ISO Draft 19770-2[1]. Für jedes installierte Paket wird ein eigenes XML Dokument generiert.
Standard Szenario	Alle Tags werden generiert und auf der Standardausgabe durch Newlines (`\n`) getrennt ausgegeben. Es wird ein Tag pro Zeile dargestellt. Die Attribute des Tag Creators werden mit vordefinierten Werten befüllt. Es ist ein definiertes Set Elemente und Attribute enthalten.
Alternatives Szenario 1	Die Attribute des Tag Creators können mittels optionalen Parametern spezifiziert werden.
Alternatives Szenario 2	Die Trennzeichen zur Ausgabe der Tags können mittels optionalem Parameter spezifiziert werden.
Alternatives Szenario 3	Zu Debug-Zwecken können die Tags mittels optionalem Parameter in einer eingerückten und einfacher lesbaren Form ausgegeben werden («pretty printing»).
Alternatives Szenario 4	Mittels optionalem Parameter können die XML Tags mit einem Payload-Element versehen werden, welches für jedes Paket die darin enthaltenen Dateien auflistet.

8.1.3.4 UC04: Targeted Request

Akteur	strongSwan IMC
Story	Der Akteur möchte nur den SWID Tag eines bestimmten Paketes erhalten.
Standard Szenario	Mittels Parameter kann dem Generator ein Filterwert mitgegeben werden, um einen bestimmten Tag herauszufiltern. Als Filterwert gibt es zwei Varianten: Entweder den Package Name oder die Software-ID.
Alternatives Szenario 1	Die Attribute des Tag Creators können mittels optionalen Parametern spezifiziert werden.
Alternatives Szenario 2	Zu Debug-Zwecken können die Tags mittels optionalem Parameter in einer eingerückten und einfach lesbaren Form ausgegeben werden («pretty printing»).
Alternatives Szenario 3	Mittels optionalem Parameter können die XML Tags mit einem Payload-Element versehen werden, welches für jedes Paket die darin enthaltenen Dateien auflistet.

8.2 Paketmanager

Im Rahmen dieser Arbeit wurde die Unterstützung für drei der am weitesten verbreiteten Paketverwaltungssysteme (DPKG, RPM und Pacman) implementiert. Damit werden acht der zehn führenden Linux- und BSD-Distributionen gemäss «DistroWatch.com»[8] abgedeckt.

Ein Paketmanager verwaltet unter anderem alle verfügbaren und installierten Softwarepakete inklusive Meta-Informationen wie Paketname, Version oder Installationsstatus.

8.2.1 DPKG

DPKG[1] (Abkürzung für *Debian Package*) ist die Basis der Paketverwaltung in Debian und in verwandten Distributionen wie Ubuntu.

Liste installierter Pakete abfragen

```
dpkg-query --show --showformat='${Package}\t${Version}\t${Status}\n'
```

Listing 8.1: DPKG Abfrage installierter Pakete

Dateien zu Paket abfragen

```
dpkg-query --listfiles <package-name>
```

Listing 8.2: DPKG Abfrage der Paketdateien

Bemerkungen

Entfernte DPKG Pakete können einen sogenannten `RC` Status haben. Dieser liegt vor, wenn ein Paket deinstalliert wurde, aber die Konfigurationsdateien auf dem System belassen wurden. Die `--show` Option liefert auch diese Pakete. Der Status ist in der Ausgabe als `"deinstall ok config-files"` ersichtlich, diese Pakete mussen nachträglich herausgefiltert werden.

[1] `https://alioth.debian.org/projects/dpkg`

8.2.2 RPM

RPM[2] (Abkürzung für *Red Hat Package Manager*) ist der Standard-Paketmanager für Red Hat Linux Systeme sowie verwandte Distributionen wie openSUSE, CentOS, Mandriva und Fedora.

Liste installierter Pakete abfragen

```
rpm -qa --queryformat %{name}\t%{version}-%{release}
```

Listing 8.3: RPM Abfrage installierter Pakete

Dateien zu Paket abfragen

```
rpm -ql <package-name>
```

Listing 8.4: RPM Abfrage der Paketdateien

8.2.3 Pacman

Pacman[3] ist der Paketmanager unter Arch Linux und Derivaten wie Manjaro Linux oder Chakra.

Liste installierter Pakete abfragen

```
pacman -Q --color never
```

Listing 8.5: Pacman Abfrage installierter Pakete

Dateien zu Paket abfragen

```
pacman -Ql <package-name>
```

Listing 8.6: Pacman Abfrage der Paketdateien

[2] https://rpm.org
[3] https://www.archlinux.org/pacman/

8.3 Architektur

8.3.1 Übersicht

Der SWID Generator wurde komplett in Python geschrieben und nutzt keine externen Bibliotheken. Es wird sowohl Python 2 wie auch Python 3 unterstützt.

Der Quellcode ist modular aufgebaut, gewisse Komponenten wurden in separate Subpackages aufgeteilt, nämlich `enironments` und `generators` (Abbildung 8.2):

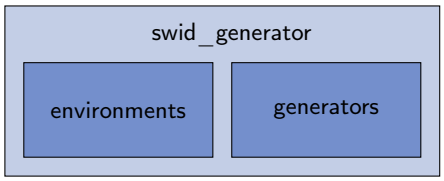

Abbildung 8.2: Aufteilung des SWID Generators in Python Packages

Das Hauptpackage – `swid_generator` – enthält folgende Module:

- `main.py`: Der Einstiegspunkt in das Programm
- `argparser.py`: Klassen und Funktionen um Kommandozeilenargumente zu parsen
- `print_functions.py`: Funktionen um Tag- und den Output der Software-ID-Generatoren auf die Standardausgabe auszugeben
- `package_info.py`: DTO-Klassen für Dateien und Softwarepakete
- `exceptions.py`: Eigene Fehlerklassen
- `settings.py`: Globale Variablen, wie z. B. `DEFAULT_ENTITY_NAME`
- `meta.py`: Metainformationen zum Generator, wie z. B. der Programmname, die Lizenz oder die Version

Die zwei Subpackages sind für Input respektive Output zuständig. Im `environments` Package sind die Environment-Klassen für die verschiedenen Paketmanager enthalten. Die Klassen greifen auf die jeweiligen Paketdatenbanken zu und geben diese Informationen an das Hauptprogramm zurück. Die Environment-Registry befindet sich ebenfalls in diesem Package. Mehr dazu im Abschnitt 8.3.3.1. Das `generators` Package enthält die nötigen Funktionen, um aus den durch das entsprechende Environment zurückgegebenen Paketinformationen, die dazugehörigen Software-IDs oder SWID Tags zu generieren. Mehr dazu im Abschnitt 8.3.3.3.

Als Einstiegspunkt in den SWID Generator dient das `main.py` Modul. Um das Programm direkt aus dem Quellcode-Ordner heraus zu starten, sollte Python mit der `-m` Option und dem vollständigen Modulpfad gestartet werden:

```
python -m swid_generator.main
```

Listing 8.7: Aufruf der Main Funktion des SWID Generators

Wenn das Programm über `pip` oder `setup.py` installiert wird (Abschnitt 8.6), kann es auch direkt über den Programmnamen aufgerufen werden:

```
swid_generator
```

Listing 8.8: Aufruf des installierten SWID Generators

8.3.2 Ablauf

8.3.2.1 Initialisierung

Wenn der SWID Generator gestartet wird, beginnt die Code-Ausführung im `main.py` Modul. Zuerst müssen die Environment-Klassen registriert werden. Danach werden alle Kommandozeilen-Argumente verarbeitet. Mit den nun vorhandenen Informationen muss entschieden werden, welches Environment effektiv verwendet werden soll – entweder durch den Benutzer definiert oder durch die automatische Paketmanager-Erkennung.

Der Ablauf ist wie folgt:

1. Die Environment Registry (Abschnitt 8.3.3.2) wird initialisiert.
2. Alle verfügbaren Environment Klassen (Abschnitt 8.3.3.1) werden der Registry übergeben.
3. Der Argument Parser wird mit einer Referenz auf die Registry instanziert.
4. Die Parser Instanz parst mithilfe des `argparse` Moduls aus der Python Stanardbibliothek die Kommandozeilen-Argument und generiert daraus ein «Options»-Objekt.
5. Dieses Objekt wird nun der Environment Registry übergeben. Diese kann mit den Informationen darin die geeignete Environment Klasse finden und an das Hauptprogramm zurückgeben. Das Environment kann per Kommandozeilen-Argument vorgegeben werden, andernfalls wird es automatisch detektiert.

In der folgenden Abbildung 8.3 ist dieser Ablauf als UML Sequenzdiagramm dargestellt:

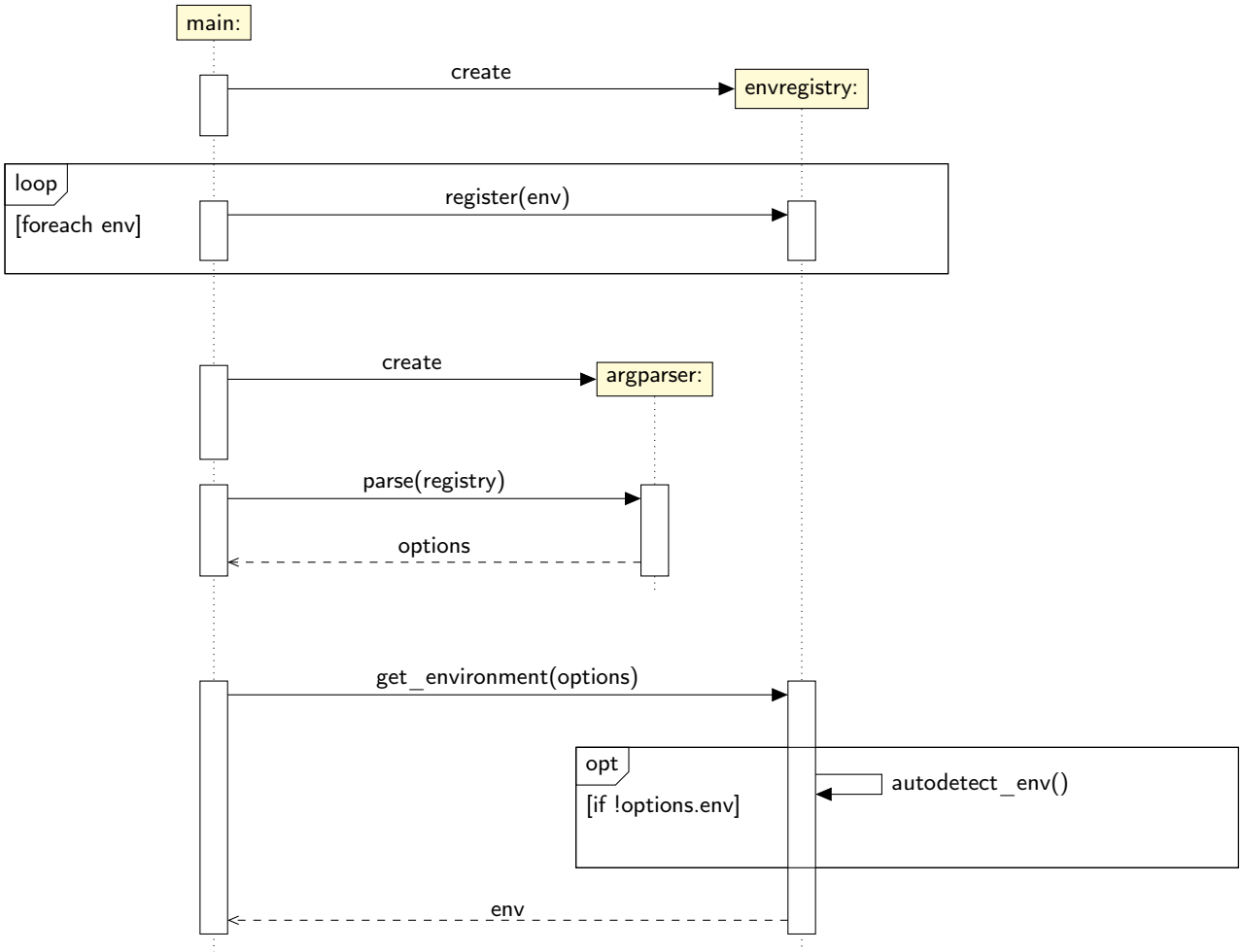

Abbildung 8.3: Sequenzdiagramm, Initialisierung

8.3.2.2 Generierung

Nachdem nun, wie in Abschnitt 8.3.2.1 beschrieben, alle nötigen Informationen gesammelt wurden, kann die Generierung der Daten aus den Paketmanager-Informationen erfolgen. Ob SWID Tags oder Software-IDs generiert werden, ist abhängig vom ersten Kommandozeilenargument, welches entweder `swid` oder `software-id` sein kann.

1. Zuerst werden mit Hilfe des SWID Tag Generators (Abschnitt 8.3.3.3) die benötigten Informationen aus der Paketmanager-Datenbank ausgelesen und daraus SWID Tags

generiert.

2. Die zurückgegebenen Tags werden mit Hilfe der entsprechenden Print-Funktionen (Abschnitt 8.3.3.4) auf die Standardausgabe ausgegeben.
3. Falls im ganzen Programmablauf keine Fehler aufgetreten sind, wird das Programm nun mit dem Status Code 0 (=keine Fehler) beendet.

Dieser Ablauf ist im folgenden Sequenzdiagramm (Abbildung 8.4) visualisiert:

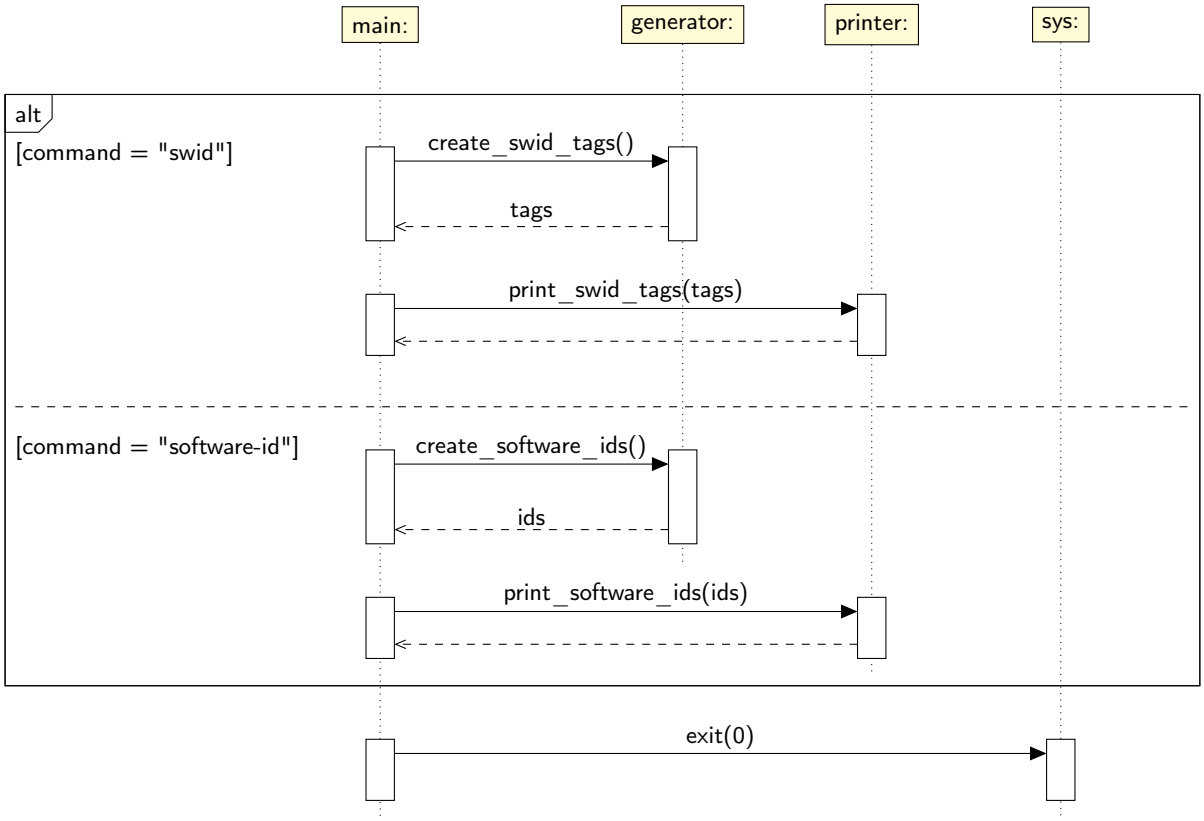

Abbildung 8.4: Sequenzdiagramm, Generierung

8.3.3 Implementationsdetails

8.3.3.1 Environments

Die Environments sind das Kernstück des SWID Generators, sie greifen auf den System-Paketmanager zu und bereiten die Liste der installierten Pakete für die weitere Verarbeitung

auf.

Wie bereits im Abschnitt 8.2 erwähnt, bietet der SWID Generator zum aktuellen Zeitpunkt Unterstützung für drei Paketmanager an. Damit der SWID Generator in Zukunft auch auf bisher noch nicht unterstützten Systemen wie Slackware Linux, FreeBSD oder Windows einsatzfähig ist, wurde Wert darauf gelegt, dass sich das Hinzufügen neuer Environments einfach gestaltet.

Um dieses Ziel zu erreichen, ist eine einheitliche Schnittstelle für die Environment-Klassen essentiell. Wie es in Python üblich ist, wird dafür ein Ansatz mit impliziten Contracts anstelle von explizit ausprogrammierten Interfaces verwendet[9]. Dieser Architektur-Ansatz ist auch als *Duck Typing* bekannt.

Damit eine Klasse ein gültiges Environment darstellt, benötigt sie fünf Methoden:

- `get_package_list()`: Gibt eine Liste von `PackageInfo` Instanzen zurück, welche die installierten Softwarepakete auf dem aktuellen System repräsentiert.
- `get_files_for_package(package_name)`: Akzeptiert einen Paketnamen und gibt eine Liste von `FileInfo` Instanzen zurück, welche die zu einem Softwarepaket gehörigen Dateien repräsentiert.
- `is_installed()`: Prüft, ob der repräsentierte Paketmanager auf dem aktuellen System installiert ist oder nicht. Dies kann beispielseweise getan werden, indem geprüft wird, ob zentrale Executables (z. B. `dpkg-query` bei DPKG) vorhanden sind.
- `get_architecture()`: Gibt den System-Architekturtyp (z. B. `x86_64` oder `i386`) zurück.
- `get_os_string()`: Gibt den Bezeichner der aktuellen Betriebssystem-Distribution zurück (z. B. `debian_7.4` oder `fedora_19`).

Da Environments meist gemeinsam benötigte Funktionalität aufweisen – beispielsweise die Überprüfung, ob eine vom Paketmanager gemeldete Datei auch wirklich auf dem System vorhanden ist – wurden diese in eine Basisklasse namens `CommonEnvironment` ausgegliedert. Diese kann wahlweise von einem Environment erweitert werden.

Die Organisation der aktuell vorhandenen Environment-Klassen ist in der nachstehenden Grafik 8.5 ersichtlich:

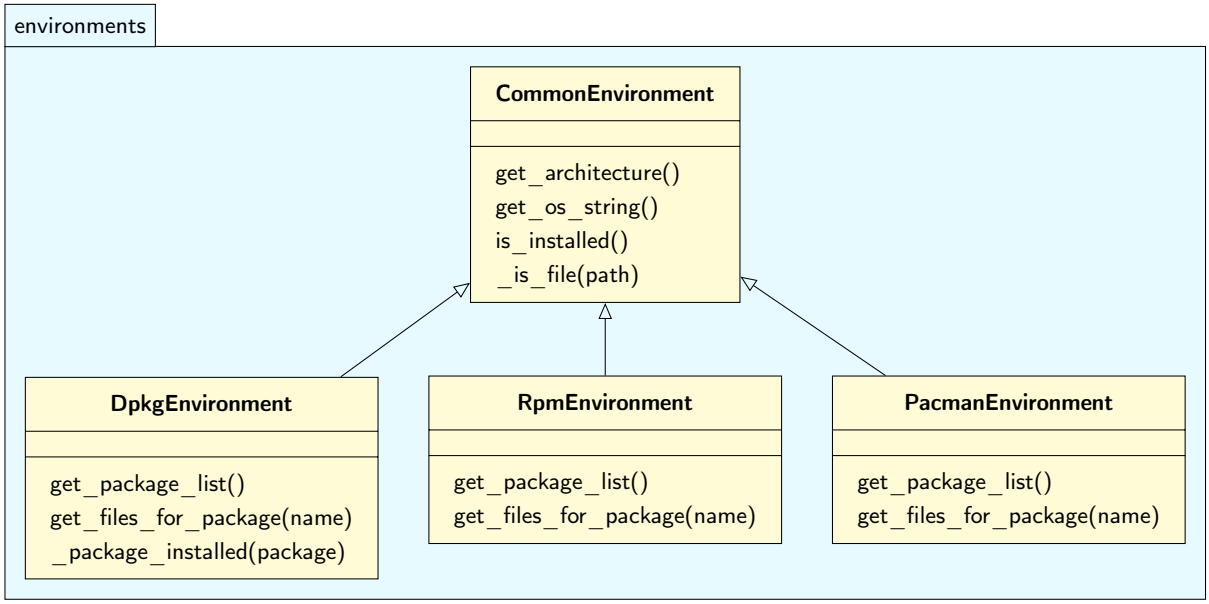

Abbildung 8.5: Die Struktur Environment-Klassen

8.3.3.2 Registry

Nachdem eine Environment-Klasse definiert wurde (Abschnitt 8.3.3.1), muss diese im Hauptprogramm registriert und verwaltet werden. Dies ist die Aufgabe der `EnvironmentRegistry`-Klasse.

Das Kernstück dieser Klasse ist die `register()`-Funktion. Sie benötigt als Argumente eine Kurzbezeichnung für das Environment sowie eine Referenz auf die Environment-Klasse:

```
registry = EnvironmentRegistry()
registry.register('rpm', RpmEnvironment)
registry.register('dpkg', DpkgEnvironment)
registry.register('pacman', PacmanEnvironment)
```

Listing 8.9: Registrieren von Environments

Die Klassen werden als Instanzen intern in einem Dictionary gespeichert. Um die Liste der verfügbaren Environments abzufragen, kann die `get_environment_strings()` Methode verwendet werden:

```
>>> registry.get_environment_strings()
['auto', 'dpkg', 'pacman', 'rpm']
```

Listing 8.10: Verfügbare Environments abfragen

Als Resultat werden die Kurzbezeichnungen der registrierten Environments alphabetisch sortiert aufgelistet. Zusätzlich wird der String `'auto'` zurückgeliefert, welcher für die automatische Environment Erkennung steht. Diese Liste wird vom Argument Parser verwendet, um die Liste der gültigen Werte für die `--env` Option zu finden.

Die Methode `get_environment()` der `EnvironmentRegistry`-Klasse nimmt eine Environment-Kurzbezeichnung entgegen und führt folgende Schritte aus:

1. Wenn die Kurzbezeichnung `'auto'` ist, wird versucht, das installierte Environment automatisch zu finden, indem über die Liste der registrierten Environments iteriert wird und für jedes Environment geprüft wird, ob der passende Paketmanager auf dem System installiert ist. Das erste Environment, bei dem dies zutrifft, wird zurückgegeben. Falls dies für keines der Environments der Fall sein sollte, wird eine `AutodetectionError` Exception ausgelöst. Weiter zu Schritt 3.
2. Wenn die Kurzbeschreibung nicht `'auto'` ist, wird die passende Environment Instanz aus dem internen Dictionary geholt. Wenn keine entsprechende Instanz gefunden wird, wird ein `KeyError` ausgelöst. Weiter zu Schritt 3.
3. Zuletzt wird geprüft, ob das zurückgegebene Environment installiert ist. Wenn nicht, wird eine `EnvironmentNotInstalledError` Exception ausgelöst. Ansonsten wird die Environment-Instanz zurückgegeben.

8.3.3.3 Generatoren

Die Generatoren im `generators` Package sind dafür zuständig, aus den durch das entsprechende Environment zurückgegebenen Paketinformationen die dazugehörigen Software-IDs oder SWID Tags zu generieren.

Diese Funktionen sind als Python Generator-Funktionen[10, S. 94-95] implementiert und geben die einzelnen Resultat-Strings mit `yield` anstelle von `return` zurück. Dadurch kann über alle SWID Tags oder über alle Software-IDs iteriert werden, ohne dass alle Strings gleichzeitig in den Hauptspeicher geladen werden müssen. Jedes Mal wenn auf der Generator-Instanz `next()` aufgerufen wird, wird nur der nächste String berechnet. Dieses Prinzip ist auch als «Lazy Evaluation»[11, S. 85–93] bekannt.

8.3.3.4 Print-Funktionen

Die Print-Funktionen sind im Modul `print_functions.py` enthalten. Ihre Aufgabe ist es, den Output der Generatoren korrekt auf die Standardausgabe zu schreiben.

Aktuell sind zwei Print-Funktionen definiert, eine für die SWID Tags und eine für die Software-IDs. Beide nehmen einen Generator entgegen und geben den Inhalt Zeile für Zeile aus, getrennt durch ein vordefiniertes Trennzeichen.

Die Art, wie die Inhalte ausgegeben werden, ist jedoch unterschiedlich. Während die XML Tags normal oder in einer «prettified» Version ausgegeben werden können, werden die Software-IDs direkt ausgegeben, ohne sie zu manipulieren. Aus diesem Grund haben wir für die Implementierung das Strategy Pattern[12] verwendet.

Es gibt eine generische Funktion namens `iterate`, welche als Argumente den Generator, eine Action-Funktion (die «Strategie»), einen Separator und einen End-String annimmt. Die Funktionsimplementation sieht folgendermassen aus:

```
def iterate(generator, action_func, separator, end):
    item = next(generator)
    while item:
        action_func(item)
        try:
            item = next(generator)
            safe_print(separator, end='')
        except StopIteration:
            safe_print(end, end='')
            break
```

Listing 8.11: Implementation der Iterate-Funktion

Die einfachste Form eines Aufrufes wird für die Software-IDs verwendet, wo die Action-Funktion keine weitere Logik enthält als die Ausgabe des Strings (siehe Listing 8.12). Die SWID-Tag Version sieht ähnlich aus, allerdings wird dort zusätzlich das «Prettifying» der XML Tags durchgeführt.

```
def action(swid):
    safe_print(swid, end='')
iterate(software_ids, action, separator, end='\n')
```

Listing 8.12: Implementation der Software-ID Print-Funktion

Dadurch, dass die Generatoren jeweils nur ein einzelnes Element zurückgeben, kann dieses sofort auf die Standardausgabe geschrieben werden und dann vom Garbage Collector gelöscht werden. Dies spart einerseits Arbeitsspeicher, andererseits kann es die Wartezeit eines aufrufenden Programmes verkürzen, da es mit der Verarbeitung bereits beginnen kann, bevor alle Daten generiert und ausgegeben worden sind.

8.3.3.5 Python 3 Kompatibilität

Obwohl Python 3.0 bereits im Jahr 2008 veröffentlicht wurde, hat es sich bisher noch nicht durchgesetzt. Da mit dieser Version die Rückwärtskompatibilität gebrochen wird, wehrten sich viele Entwickler gegen einen Wechsel von der etablierten Version 2 auf Python 3. Einerseits war es anfänglich schwierig, Python 2 und 3 mit einer einzigen Codebasis zu unterstützen, andererseits nutzten viele Programme Dritt-Bibliotheken welche noch nicht auf Python 3 portiert waren.

Mit Python 3.3 wurde der Unicode-Präfix für Strings wieder eingeführt[13]. Dadurch wurde die Unterstützung beider Sprachversionen mit einer einzigen Codebasis bedeutend einfacher. Auch wurden mehrere Portierungs-Ratgeber publiziert, welche die Übergangsphase vereinfachen[14, 15]. Die Adoptionsrate stieg dadurch stark an, so dass inzwischen 162 der 200 meistgenutzten Python Pakete mit Python 3 kompatibel sind[16].

Bei der Entwicklung wurden vor allem die Hinweise aus dem Ratgeber «Quick Tips on Making Your Code Python 3 Ready»[17] verwendet. Eines der wichtigsten Elemente daraus ist der folgende «Future-Header»:

```
from __future__ import print_function, division, absolute_import, unicode_literals
```

Listing 8.13: Python 3 Compat Future-Header

Durch die konsequente Nutzung dieser Import-Zeile in jedem Python-Modul konnten die Inkompatibilitäten stark reduziert werden; die Portierung auf Python 3.3 und 3.4 war anschliessend einfach umzusetzen.

Um die Ausgabe von Unicode- wie auch Byte-Strings unter beiden Versionen zu vereinfachen, haben wir zusätzlich eine `safe_print` Funktion definiert (Listing 8.14).

Zur Sicherstellung der Kompatibilität wurde die Test Suite zudem so konfiguriert, dass die Tests auf allen unterstützten Python-Versionen ausgeführt werden (Abschnitt 8.5).

```
def safe_print(data, end='\n'):
    # Python 3
    if hasattr(sys.stdout, 'buffer'):
        if isinstance(data, bytes):
            sys.stdout.buffer.write(data)
        else:
            sys.stdout.write(data)
        if isinstance(end, bytes):
            sys.stdout.buffer.write(end)
        else:
            sys.stdout.write(end)
        sys.stdout.flush()
    # Python 2
    else:
        print(data, end=end)
```

Listing 8.14: Safe-Print Funktion

8.4 Ergebnisse

8.4.1 Generieren von Software-IDs

Um Software-IDs zu generieren, kann der SWID Generator folgendermassen aufgerufen werden:

```
usage: swid_generator software-id [-h] [--env {auto,dpkg,pacman,rpm}]
                                  [--doc-separator DOCUMENT_SEPARATOR]
                                  [--regid REGID]
```

Listing 8.15: Generierung von Software-IDs

Nachfolgend ein Beispielaufruf auf einem Debian-System:

```
$ swid_generator software-id --regid=regid.1998-05.ch.hsr
regid.1998-05.ch.hsr_debian_7.4-x86_64-acpi-1.6-1
regid.1998-05.ch.hsr_debian_7.4-x86_64-acpi-support-base-0.140-5
...
regid.1998-05.ch.hsr_debian_7.4-x86_64-zlib1g-1:1.2.7.dfsg-13
regid.1998-05.ch.hsr_debian_7.4-x86_64-zlib1g-dev-1:1.2.7.dfsg-13
```

Listing 8.16: Software-ID Auszug eines Debian-Systems

Die vollständige Dokumentation der Optionen kann auf der Kommandozeile mit dem `--help` Parameter ausgegeben werden.

8.4.2 Generieren von SWID Tags

Die Syntax um SWID Tags zu generieren sieht folgendermassen aus:

```
usage: swid_generator swid [-h] [--env {auto,dpkg,pacman,rpm}]
                           [--doc-separator DOCUMENT_SEPARATOR]
                           [--regid REGID] [--entity-name ENTITY_NAME]
                           [--full] [--pretty]
                           [--software-id SOFTWARE-ID | --package PACKAGE]
```

Listing 8.17: Generieren von SWID Tags

Hier ein Beispiel eines Targeted Requests für das `cowsay`-Paket mit «Pretty-Printing»:

```
$ swid_generator swid --package cowsay --pretty
<?xml version="1.0" encoding="utf-8"?>
<SoftwareIdentity name="cowsay" uniqueId="debian_7.4-x86_64-cowsay-3.03+dfsg1-4"
      version="3.03+dfsg1-4" versionScheme="alphanumeric"
      xmlns="http://standards.iso.org/iso/19770/-2/2014/schema.xsd">
  <Entity name="strongSwan" regid="regid.2004-03.org.strongswan" role="tagcreator"/>
</SoftwareIdentity>
```

Listing 8.18: Beispiel eines Targeted Requests

8.4.3 Prüfen der Rückgabewerte

Nach einem Aufruf des SWID Generators sollte immer der Exit Code geprüft werden. In einem Bash-Terminal kann der Rückgabewert mit `echo $?` angezeigt werden.

Folgende Rückgabewerte sind möglich:

0 Ausführung war erfolgreich.

1 Ein Targeted Request hat keine Resultate ergeben.

2 Kommandozeilen-Argumente sind ungültig.

3 Der System-Paketmanager konnte nicht erkannt oder gefunden werden.

8.4.4 Performance

Durch die in den Abschnitten 8.3.3.3 und 8.3.3.4 beschriebenen Optimierungen konnte der Speicherverbrauch und die gepufferte Ausgabe verbessert werden. Nachfolgend ein paar Geschwindigkeitsmessungen, die auf verschiedenen Systemen durchgeführt wurden. Getestet wurden dabei die folgenden Befehle:

- ids: `swid_generator software-id > /dev/null`
- tags: `swid_generator swid > /dev/null`
- tags-full: `swid_generator swid --full > /dev/null`
- tags-full-pretty: `swid_generator swid --pretty --full > /dev/null`

Die Befehle wurden jeweils mehrmals ausgeführt, um mögliche Caching-Effekte auszugleichen. Nachfolgend die getesteten Systeme:

- dpkg-952: DPKG, 952 installierte Pakete
- pacman-1223: Pacman, 1223 installierte Pakete
- rpm-437: RPM, 437 installierte Pakete

Die Ergebnisse:

	dpkg-952	**pacman-1223**	**rpm-437**
ids	0.143s	0.054s	0.511s
tags	0.227s	0.102s	0.534
tags-full	34.256s	13.822s	6.213s
tags-full-pretty	54.039s	31.202s	8.920s

Da auf den Systemen nicht gleich viele und gleich grosse Pakete installiert sind, sind diese Resultate schwer vergleichbar. Die zu erwartende Laufzeit ist jedoch abschätzbar.

Klar ersichtlich ist, dass der grösste Zeitaufwand beim Generieren von SWID Tags mit der `--full` Option auftritt. Dies ist erklärbar, da hierbei jede Datei in jedem installierten Paket auf dem System vom SWID Generator auf Existenz geprüft wird. An diesem Punkt gibt es möglicherweise noch Optimierungspotential.

Das Pretty Printing ist ebenfalls ein zeitaufwändiger Vorgang, da die generierten Tags hierfür ein zweites Mal geparst und in einer «eingerückten» Variante ausgegeben werden. Der Mehraufwand gegenüber der Variante ohne `--pretty` betrug in den drei gemessenen Beispielfällen zwischen 43% und 125%. Da diese Option nur für Debugging-Zwecke gebraucht werden sollte, ist dies in der Praxis kein nenneswertes Problem und auch kaum ein sinnvolles Ziel für Optimierungen.

8.5 Qualitätsmanagement

Wie in Abschnitt 9.3.1.2 genauer erläutert, haben wir zum Schreiben und Ausführen der Testsuite Pytest verwendet. Im Gegensatz zu strongTNC sollte der swidGenerator mehrere Python-Versionen unterstützen, da die Deployment-Ziele (TNC Clients) eine höhere Betriebssystem-Diversität aufweisen.

Um dies zu gewährleisten, wurde Tox[4] verwendet. Tox ist ein Tool, welches gemäss einer Konfigurationsdatei automatisch verschiedene «Virtualenvs» erstellt. Jedes Virtualenv enthält eine andere Python-Version mit den entsprechenden Abhängigkeiten. Die Testsuite wird anschliessend in jedem Virtualenv separat ausgeführt.

```
$ tox -e py27,py33,py34,pypy
...
================== 26 passed, 17 skipped in 1.23 seconds ==================
_________________________________ summary _________________________________
  py27: commands succeeded
  py33: commands succeeded
  py34: commands succeeded
  pypy: commands succeeded
  congratulations :)
```

Wie in Abschnitt 9.3.1.3 beschrieben, wird auch für den SWID Generator Travis CI eingesetzt. Mit der entsprechenden Konfiguration wird jede von Tox unterstützte Version in einem separaten Job getestet. Die Resultate werden dann als Build Matrix dargestellt.

Job	Duration	Finished	ENV
304.2	12 sec	6 days ago	TOXENV=py27
304.3	22 sec	6 days ago	TOXENV=py33
304.4	16 sec	6 days ago	TOXENV=py34
304.5	26 sec	6 days ago	TOXENV=pypy

Abbildung 8.6: Tox Resultate

Analog zu strongTNC werden beim SWID Generator die Coding-Guidelines durch die Test Suite forciert. Verletzungen der Guidelines werden als Fehler betrachtet.

[4] http://tox.readthedocs.org/

8.6 Packaging / Deployment

Für eine vereinfachte Installation des SWID Generators auf den Zielsystemen wurde ein Setuptools[5] basiertes `setup.py` Script erstellt. Dieses liest die benötigten Metadaten aus dem `swid_generator/meta.py` Modul. Die Datei `MANIFEST.in` steuert, welche Dateien beim Paketieren eingebunden werden.

Um den SWID Generator mit diesem Script in das globale Python-Verzeichnis zu installieren, reicht folgender Befehl:

```
$ sudo python setup.py install
```

Listing 8.19: Installieren des SWID Generators

Auch ein Archiv mit der Source-Distribution des Projektes kann über die `setup.py` Datei erzeugt werden:

```
\begin{bashcode}
$ python setup.py sdist
```

Listing 8.20: Erstellen einer SWID Generators Source Distribution

Um die Installation noch weiter zu vereinfachen, kann der SWID Generator auch in den Python Package Index[6] hochgeladen werden. Dafür muss das Projekt dort registriert, eine Source- sowie eine Binärdistribution erstellt und diese schlussendlich hochgeladen werden. All diese Punkte können mit folgenden Befehlen erledigt werden:

```
$ pip install wheel
$ python setup.py register
$ python setup.py sdist upload
$ python setup.py bdist_wheel upload
```

Listing 8.21: Registrieren des SWID Generators auf PyPI

[5] `https://pythonhosted.org/setuptools/`
[6] `https://pypi.python.org/`

Da wir den SWID Generator bereits auf PyPI publiziert haben[7], kann dieser bereits über pip installiert werden.

```
$ sudo pip install swid_generator
```

Listing 8.22: Installieren des SWID Generators mit pip

[7] https://pypi.python.org/pypi/swid_generator/

9 strongTNC

9.1 Requirements

9.1.1 Nichtfunktionale Anforderungen

Da es sich bei diesem Projekt um eine aufbauende Arbeit handelt, werden keine neuen nichtfunktionalen Anforderungen definiert. Es sollen die Anforderungen der Vorgängerarbeit[2] eingehalten werden.

9.1.2 Use Cases

Nachfolgend werden die identifizierten Use Cases beschrieben. Als «System under Decision» wird die strongTNC Django Webapp als Ganzes betrachtet.

9.1.2.1 UC01: CMDB, Softwareinventar eines Gerätes

Akteur	strongTNC Benutzer
Story	Der Akteur will feststellen, welche Software in welcher Version zu einem bestimmten Zeitpunkt auf einem Gerät installiert war, beispielsweise weil in einer Version eine kritische Sicherheitslücke enthalten ist.
Standard Szenario	Der Akteur betrachtet das SWID Tag Inventar des gewünschten Gerätes und kann dort zu jedem Mess-Zeitpunkt die installierte Software abrufen.
Alternatives Szenario	Der Akteur wählt den gewünschten SWID Tag in der SWID Liste und die Geräte, welche die Software installiert hatten, werden aufgelistet.

9.1.2.2 UC02: Detaillierte SWID Tag File Information

Akteur	strongTNC Benutzer
Story	Der Akteur will feststellen, welche Dateien zu einem bestimmten Softwarepaket gehören.
Standard Szenario	Der Akteur findet den SWID Tag, der zum gesuchten Softwarepaket gehört in der SWID Tag Ansicht. Die Dateien, die mit dem SWID Tag gemessen wurden, sind in der Detailansicht aufgelistet.
Alternatives Szenario	Der Akteur findet das Softwarepaket in der Paketübersicht. Dort kann auf einen verknüpfen SWID Tag zugegriffen werden. So gelangt der Akteur auf die SWID Tag Ansicht und findet dort die Liste der gemessenen Dateien.

9.1.2.3 UC03: Verfügbare Entities auflisten

Akteur	strongTNC Benutzer
Story	Der Akteur will alle im System erfassten Entites auflisten und herausfinden, welche SWID Tags zu der jeweiligen Entity gehören.
Standard Szenario	Der Akteur kann sich in der Entity Ansicht alle erfassten Entities auflisten lassen. Wenn eine Entity angewählt wird, werden alle SWID Tags, die der Entity zugeordnet sind, aufgelistet.

9.1.2.4 UC04: Verfügbare SWID Tags auflisten

Akteur	strongTNC Benutzer
Story	Der Akteur will alle im System erfassten SWID Tags auflisten. Dabei ist er an folgenden Informationen eines Tags interessiert: • Software-ID • Unique-ID • Entity • RAW XML String • Im SWID Tag enthaltene Files • Verknüpfte Pakete • Devices welche diesen SWID Tag installiert haben (Abschnitt 9.1.2.1)
Standard Szenario	Der Akteur kann sich in der SWID Tag Ansicht alle erfassten SWID Tags auflisten lassen. Wenn ein SWID Tag angewählt wird, sind alle gewünschten Details sichtbar.

9.1.2.5 UC05: Erfassen eines neuen SWID Tags

Akteur	strongSwan TNC Server oder Administrator
Story	Der Akteur will einen neuen SWID Tag im System erfassen. Der Akteur liefert ein XML Dokument gemäss SWID Spezifikation[1] an das System.
Standard Szenario	Der Akteur liefert ein XML Dokument an das System und erhält eine Erfolgsmeldung die bestätigt, dass der Tag gespeichert wurde.
Alternatives Szenario 1	Eine Entity welche im gelieferten XML enthalten ist existiert bereits im System (identifiziert durch die Regid) besitzt aber einen anderen Namen. Der Name der bestehenden Entity wird entsprechend aktualisiert.
Alternatives Szenario 2	Der Tag welcher erfasst werden sollte existiert bereits im System, identifiziert durch die Software-ID. **2a** Die Erfassung eines neuen SWID Tags erfolgt programmatisch über die TNC Server Komponente. Ein Tag mit derselben Unique-ID existiert bereits und wird nicht ersetzt. **2b** Die Erfassung eines neuen SWID Tags erfolgt manuell durch den Administrator. Ein Tag mit derselben Unique-ID existiert bereits und wird mit den neuen Daten ersetzt.
Alternatives Szenario 3	Es wurde ein invalides XML Dokument geliefert. Das System informiert den Akteur über den Fehler und erstellt keinen neuen Tag.
Häufigkeit des Auftretens	Bei Systemeinführung wird diese Operation häufig durchgeführt. Idealerweise sieht das Deployment vor, dass für jeden Produktetyp ein Referenzgerät vorhanden ist. Mit diesem sollen initial alle Tags generiert und erfasst werden und danach in regelmässigen Abständen aktualisiert werden. Dadurch ist die Anzahl der nötigen Erfassungen durch Endgeräte geringer.
Datenvariation	Es soll möglich sein, eine Liste mit mehreren XML Dokumenten auf einmal ans System auszuliefern (Batch-Create). Das Erfassen soll in diesem Fall so erfolgen, dass entweder alle übermittelten SWID Tags oder keine gespeichert werden. Dadurch kann die Anzahl nötiger Systemaufrufe erheblich reduziert werden.

9.1.2.6 UC06: Durchführen einer SWID Messung für ein Gerät

Akteur	strongSwan TNC Server
Story	Der Akteur will alle auf einem Device vorhandenen SWID Tags mit einer Session verknüpfen.
Standard Szenario	Der Akteur sendet eine Liste der Software-IDs an das System, die er mit der aktuellen Session verknüpfen möchte. Der Server assoziiert die entsprechenden Tags mit der Session und bestätigt den erfolgreichen Vorgang.
Alternatives Szenario	Es existieren nicht für alle gelieferten Software-IDs Tags im System. Das System assoziiert keine Tags mit der aktuellen Session und gibt eine Liste derjenigen Software-IDs zurück, für welche keine Tags vorhanden sind. Der Akteur erfasst daraufhin alle fehlenden Tags im System. Sobald alle Tags im System erfasst sind tritt das Standard Szenario ein.

9.1.2.7 UC07: Auflisten aller Tags zu einem File

Akteur	strongTNC Benutzer
Story	Der Akteur will wissen, in welchen Tags ein bestimmtes File enthalten ist.
Standard Szenario	Der Akteur kann sich in der File Ansicht alle erfassten Files auflisten lassen. Wenn ein File angewählt wird, sind alle zugeordneten SWID Tags aufgelistet.

9.2 SWID Erweiterung

9.2.1 Datenmodell

Die Erweiterung des bestehenden Datenmodells ist in Abbildung 9.1 zu sehen. Die hinzugefügten Tabellen sind durch den grünen Hintergrund hervorgehoben. Bei Tabellen mit grauem Rahmen handelt es sich um Zwischentabellen, welche eine n:m Beziehung auflösen.
Beim Modellieren der zusätzlichen Tabellen wurde darauf geachtet, dass keine der bestehenden Tabellen angepasst werden mussten. Die Beziehungen werden daher alle von der Seite der Erweiterung hergestellt.

swid_entity Diese Tabelle beinhaltet alle erfassten Entities. Diese werden in einer eigenen Tabelle gehalten und sind über die swid_entityroles mit den SWID Tags verknüpft.

swid_entityroles Ein SWID Tag kann mehrere Entities enthalten. Entities nehmen in einem Tag eine oder eine Kombination von Rollen ein. Es existieren die Rollen Tag Creator, Licensor und Publisher. Pro SWID Tag darf es nur einen Tag Creator geben. Diese Einschränkung wird jedoch erst in den Django Models durch das ORM angewendet.

swid_tags_files In dieser Tabelle wird die Beziehung zwischen einem File und einem SWID Tag erfasst. Ein SWID Tag kann mehrere Files beinhalten (UC04: Verfügbare SWID Tags auflisten).

swid_tags_sessions Hier werden die gemessenen Tags mit passenden Sessions verknüpft (UC06: Durchführen einer SWID Messung für ein Gerät).

swid_tags Die SWID Tags werden in dieser Tabelle gespeichert.

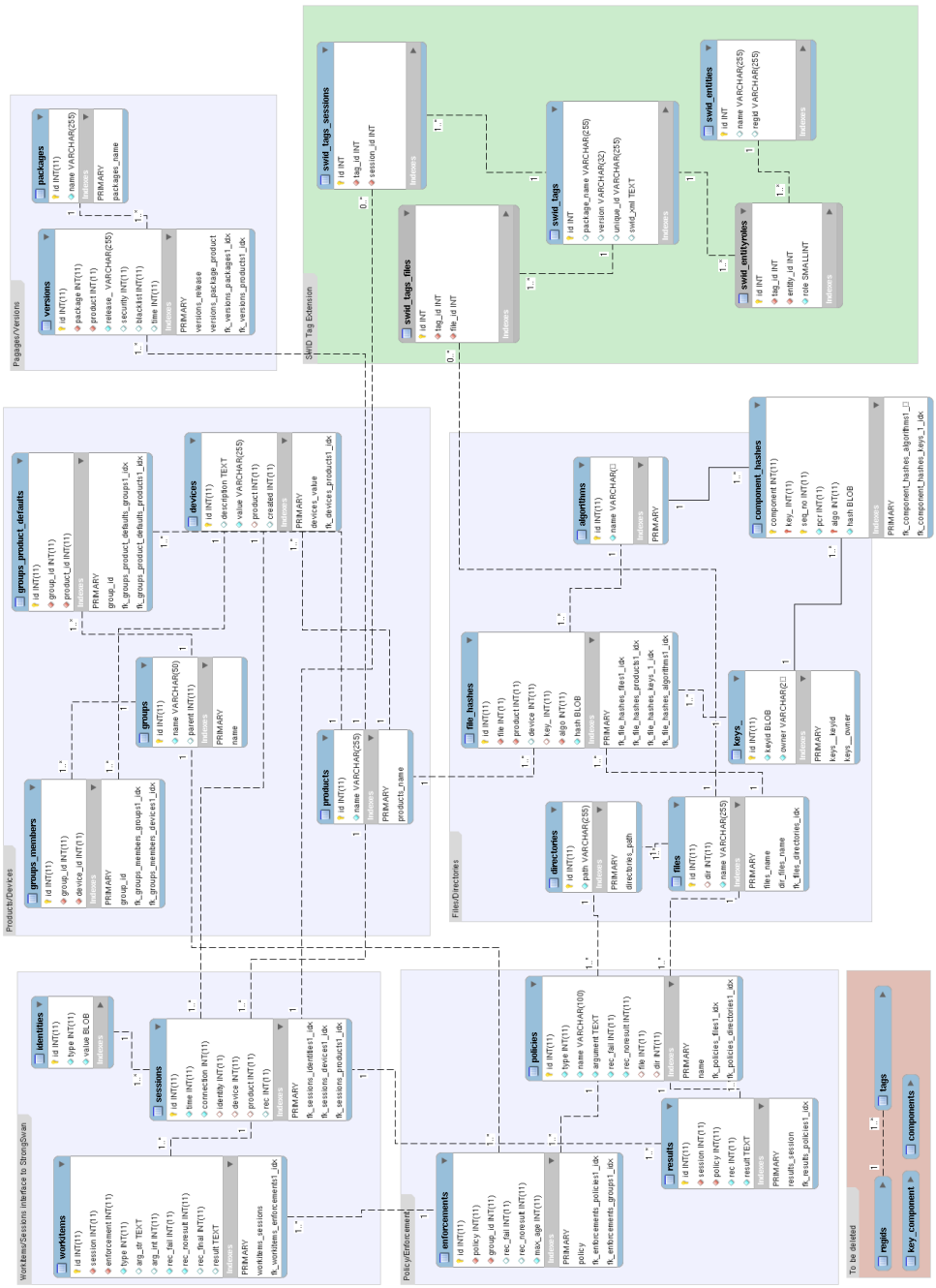

Abbildung 9.1: Bestehendes Datenmodel inklusive SWID Erweiterung (grün)

9.2.2 Architektur SWID Messung

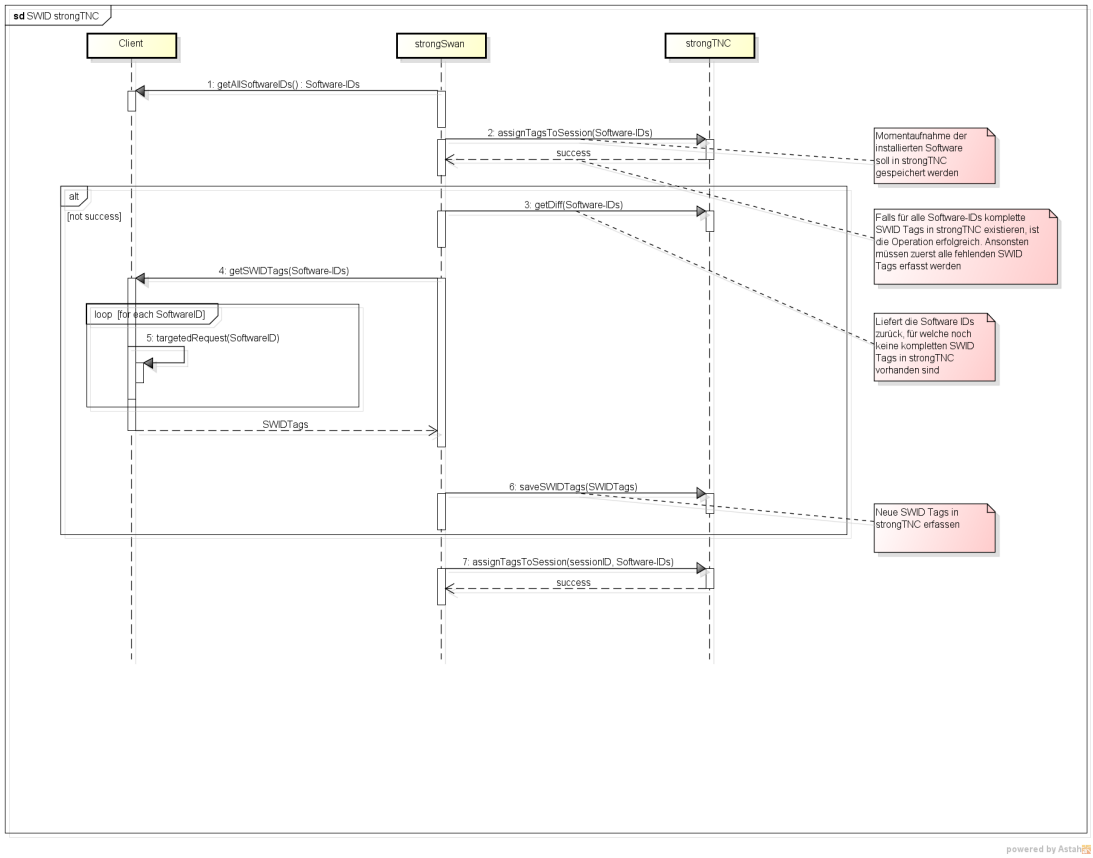

Abbildung 9.2: Ablauf einer SWID Tag Messung

In Abbildung 9.2 ist der Ablauf der SWID Tag Messung (UC06: Durchführen einer SWID Messung für ein Gerät) ersichtlich. Eine Messung läuft wie folgt ab:

1. Auf dem Client werden die Software-IDs aller installierten Pakete generiert.
2. Es wird versucht die Liste der generierten Software-IDs mit einer Session zu verknüpfen.

   2a Existiert zu allen Software-IDs ein entsprechender SWID Tag, ist die Messung abgeschlossen und die SWID Tags sind mit der Session verknüpft. Es sind keine weiteren Schritte mehr nötig.

   2b Existiert für mindestens eine Software-ID kein entsprechender SWID Tag, wird die Messung abgebrochen und eine Liste jener Software-IDs, für welche kein SWID Tag existiert, zurückgeliefert. Weiter zu Punkt 3.

3. Bevor die Messung durchgeführt werden kann, müssen alle fehlenden SWID Tags erfasst werden. Die einzelnen XML Dokumente der SWID Tags werden vom Client anhand sogenannter «Targeted Requests» (Abschnitt 8.1.3.4) erstellt. Das heisst, der Generator kreiert gezielt die Tags für die gegebenen Software-IDs und nicht für alle installierten Pakete.
4. Der Client erfasst alle neu generierten SWID Tags in strongTNC.
5. Der Client startet erneut eine Messung und übermittelt alle Software-IDs. Nun sind für alle Software-IDs die entsprechenden SWID Tags vorhanden und die Messung wird erfolgreich abgeschlossen. Ansonsten zurück zu Schritt 2.

Dieses Vorgehen erzwingt eine Aktualisierung der strongTNC Datenbank falls diese nicht alle SWID Tags enthält, bevor eine Messung erfolgreich abgeschlossen werden kann. Dadurch wird erreicht, dass auf Serverseite kein Zustand gewahrt werden muss. Dieses Vorgehen entspricht den Prinzipien von HTTP, eine Messung kann jederzeit durchgeführt werden, ohne dass ein Zustand oder Kontextinformationen vorhanden sein müssen. Das heisst, wenn eine Messung nicht vollständig und korrekt durchgeführt werden kann, werden keine Daten verändert.

9.2.3 Technische Umsetzung

Der Zugriff auf die beschriebene Systemoperation erfolgt durch die, in dieser Arbeit ausgearbeiteten, REST Schnittstelle. Das komplette Konzept ist in Anhang (Kapitel 10) zu finden.

9.2.3.1 REST Kommunikation

Im Folgenden soll der Ablauf einer Messung anhand eines konkreten Beispiels illustriert werden. In diesem Beispiel erfolgt eine Messung für drei Tags mit folgenden Software-IDs:

```
regid.2004-03.org.strongswan_Ubuntu_12.04-i686-logrotate-3.7.8-6ubuntu5
regid.2004-03.org.strongswan_Ubuntu_12.04-i686-lsb-base-4.0-0ubuntu20
regid.2004-03.org.strongswan_Ubuntu_12.04-i686-strongswan-4.5.2-1.5+deb7u3
```

Listing 9.23: Software-IDs

Der HTTP Endpunkt ist wie folgt definiert:

URI Path `/sessions/{id}/swid-measurement/`
Archetype Controller
Methods POST
Content-Type `application/json; charset=utf-8`
Request Parameter
`softwareId` Software-IDs als JSON-Liste.
Response Statuscodes
200 OK SWID Tags der übermittelten Software-IDs sind eingetragen und wurden für die übermittelte Session eingetragen
404 Not Found Session mit der spezifizierten ID wurde nicht gefunden.
412 Precondition Failed Es existieren nicht alle SWID Tags für die übertragenen Software-IDs. Als Payload werden die fehlenden Software-IDs übertragen.
JSON Format Response

```
{"data" :
    [
        "regid.2004-03.org.strongswan_Ubuntu_12.04-i686-logrotate-3.7.8-6ubuntu5",
        "regid.2004-03.org.strongswan_Ubuntu_12.04-i686-lsb-base-4.0-0ubuntu20",
        "regid.2004-03.org.strongswan_Ubuntu_12.04-i686-strongswan-4.5.2-1.5+deb7u3"
    ]
}
```

Listing 9.24: SWID Measurement Endpunkt

Übermitteln von Software-IDs

Die Daten werden als JSON-Liste an den entsprechenden Endpunkt gesendet. Der HTTP Request sieht folgendermassen aus:

```
POST /api/sessions/2/swid-measurement/ HTTP/1.1
Authorization: Basic cm9vdDpyb290
Host: strongtnc
Accept: application/json
Content-Type: application/json; charset=utf-8
Content-Length: 232

{
    "data":
    [
        "regid.2004-03.org.strongswan_Ubuntu_12.04-i686-logrotate-3.7.8-6ubuntu5",
        "regid.2004-03.org.strongswan_Ubuntu_12.04-i686-lsb-base-4.0-0ubuntu20",
        "regid.2004-03.org.strongswan_Ubuntu_12.04-i686-strongswan-4.5.2-1.5+deb7u3"
    ]
}
```

Listing 9.25: HTTP Request einer SWID Messung

Der entsprechende cURL[1] Befehl:

```
curl -i -X POST https://strongtnc/api/sessions/2/swid-measurement/ \
                -u username:password \
                -H "Accept: application/json" \
                -H "Content-Type: application/json; charset=utf-8" \
                -d "{ \"data\" : $DATA }"
```

Listing 9.26: cURL Beispiel für SWID Measurement

Antwort - Status 412 Precondition Failed
Sind noch nicht alle Tags in der strongTNC Datenbank vorhanden, werden die Software-IDs der fehlenden Tags zurückgeliefert. Der HTTP Status Code ist `412 Precondition Failed`. Es werden zu diesem Zeitpunkt noch keine Tags mit der Session verknüpft. Eine entsprechende Antwort kann wie folgt aussehen:

```
HTTP/1.0 412 PRECONDITION FAILED
Date: Wed, 14 May 2014 15:21:45 GMT
Server: WSGIServer/0.1 Python/2.7
Vary: Accept, Cookie
Content-Type: application/json
Allow: POST, OPTIONS

["regid.2004-03.org.strongswan_Ubuntu_12.04-i686-strongswan-4.5.2-1.5+deb7u3"]
```

Listing 9.27: HTTP Response mit Status Code `412 PRECONDITION FAILED`

Antwort - Status 200 OK
Sind für alle Software-IDs bereits Tags in der Datenbank vorhanden, würde die Antwort wie folgt aussehen:

[1] `http://curl.haxx.se/`

```
HTTP/1.0 200 OK
Date: Wed, 14 May 2014 15:21:45 GMT
Server: WSGIServer/0.1 Python/2.7
Vary: Accept, Cookie
Content-Type: application/json
Allow: POST, OPTIONS

[]
```

Listing 9.29: HTTP Response einer erfolgreichen SWID Messung

Nacherfassen von SWID Tags

War die Messung nicht erfolgreich (`HTTP 412`), müssen für die zurückgelieferten Software-IDs zuerst komplette SWID Tags erfasst werden. Die Daten für diesen Request bestehen aus einer JSON-Liste von XML Strings, dadurch können auch mehrere Tags auf einmal erfasst werden. Empfehlenswert ist allerdings eine Aufteilung in mehrere Gruppen, da die Requests sonst lange dauern können.

Der Request sieht folgendermassen aus:

```
POST /api/swid/add-tags/ HTTP/1.1
Authorization: Basic cm9vdDpyb290
Host: strongtnc
Accept: application/json
Content-Type: application/json; charset="utf-8"
Content-Length: 239

{ "data" :
    [
        "<?xml version=\"1.0\" encoding=\"utf-8\"?><SoftwareIdentity name=\"fortune-mod\"...",
        "<?xml version=\"1.0\" encoding=\"UTF-8\"?><SoftwareIdentity name=\"strongswan\"..."
    ]
}
```

Listing 9.30: SWID Tags erfassen via REST API

cURL Befehl:

```
curl -i -X POST https://strongtnc/api/swid/add-tags/ \
                -u username:password \
                -H "Accept: application/json" \
                -H "Content-Type: application/json; charset=utf-8" \
                -d "{ \"data\" : $DATA }"
```

Eine ausführliche Spezifikation zur Kommunikation mit diesem REST Endpunkt ist im Anhang Abschnitt 15.7 zu finden. Darin sind unter Anderem noch Informationen zu Fehlersituationen aufgeführt.

9.2.4 SWID Views

Für die SWID Tag Erweiterungen wurden drei neue Views erstellt. Mit diesen Views können sämtliche ermittelten Usecases (Abschnitt 9.1.2) abgedeckt werden.
Die Views entsprechen visuell dem Stil der bereits vorhanden Templates. Allerdings wurde für die Implementation nicht der funktionsbasierte, sondern der klassenbasierte Ansatz gewählt. Diese Views dienen somit auch als Vorlage, beziehungsweise Machbarkeitsnachweis für die Umsetzung mit «Class Based Views».

9.2.4.1 Regid View

Der Zweck der Regid View ist das Auflisten der im System vorhandenen Entities mit Detailinformationen (UC03: Verfügbare Entities auflisten). Beim Wählen einer Entity werden die referenzierten Tags aufgelistet, diese sind mit der Tag View verknüpft.

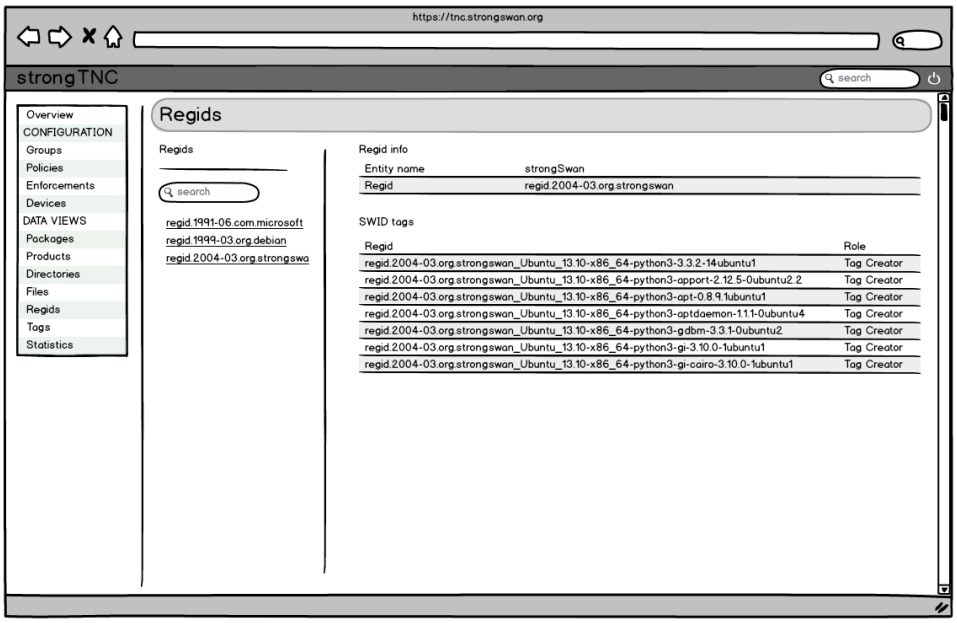

Abbildung 9.3: Mockup Regid View

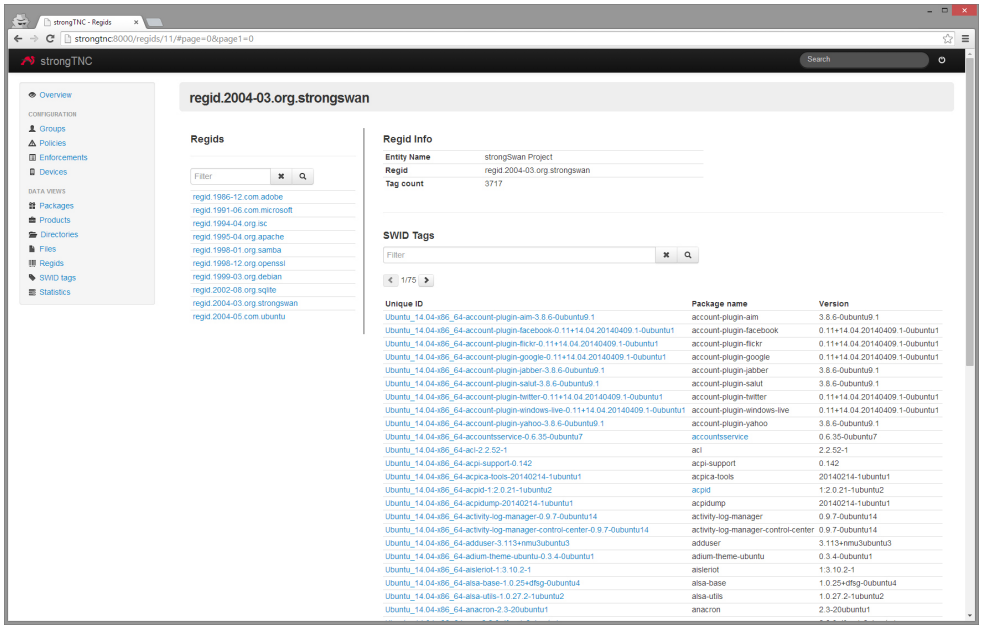

Abbildung 9.4: Regid View mit Detailansicht

9.2.4.2 SWID Tag View

Der Zweck der SWID Tag View ist das Auflisten im System vorhandener SWID Tags (UC04: Verfügbare SWID Tags auflisten). Beim Auswählen eines SWID Tags werden Detailinformationen angezeigt. So ist es möglich herauszufinden auf welchen Geräten ein Tag registriert ist und welche Dateien zum jeweiligen Tag gehören.

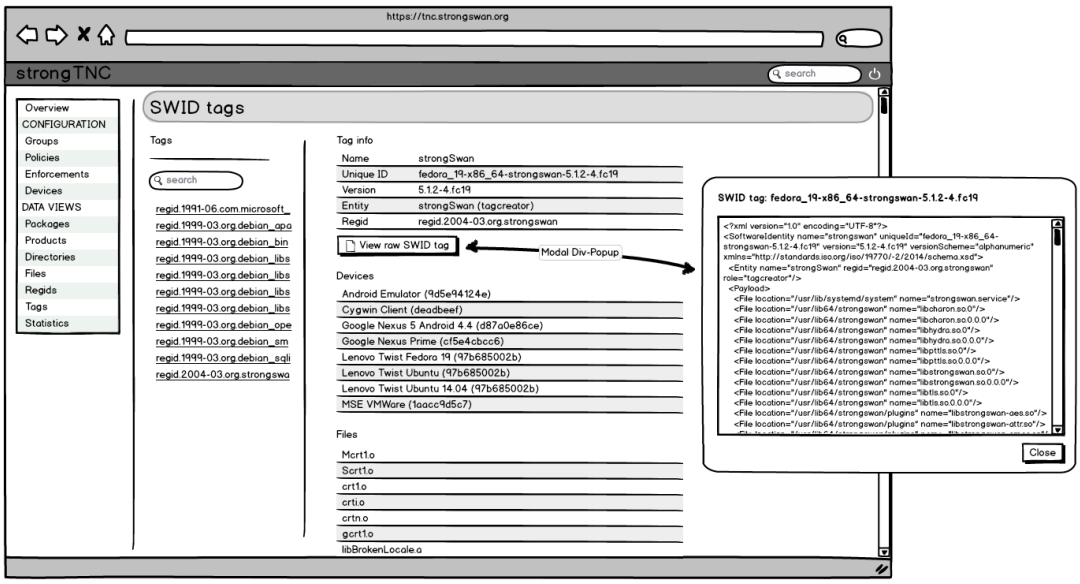

Abbildung 9.5: Mockup SWID Tag View

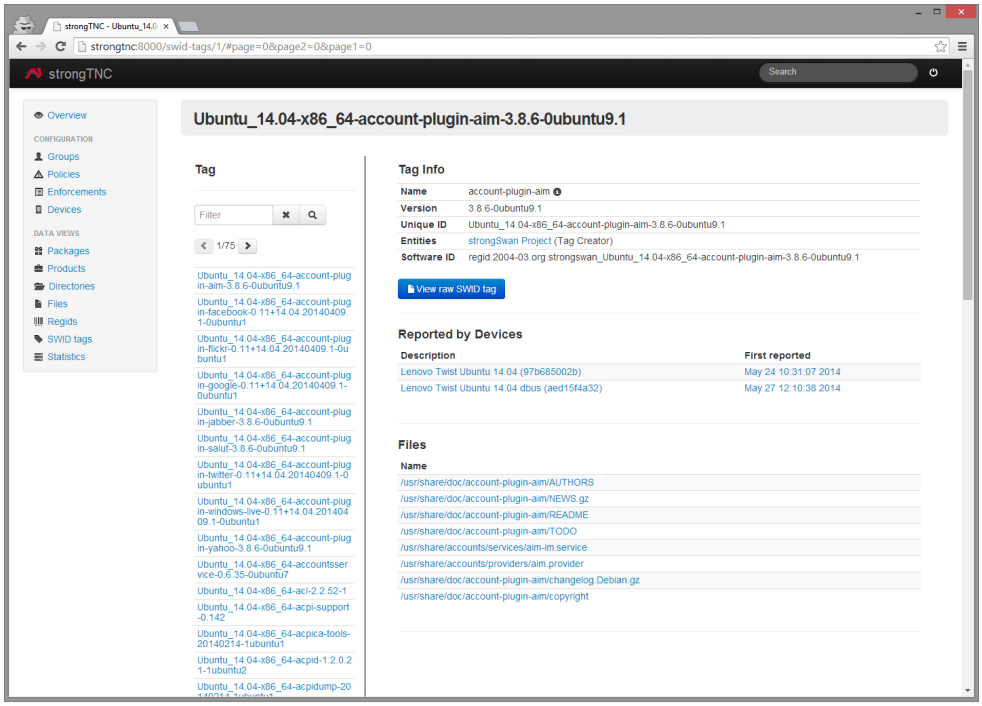

Abbildung 9.6: SWID Tag View mit Details

9.2.4.3 SWID Inventory View

Mit Hilfe der SWID Inventory View kann festgestellt werden, welche Software zu Beginn einer ausgewählten Session auf einem Gerät installiert war (UC01: CMDB, Softwareinventar eines Gerätes).
Mittels Datumsauswahl kann eingeschränkt werden welcher Bereich von Sessions geladen werden soll. Die Tags der geladenen Sessions werden dynamisch nachgeladen.

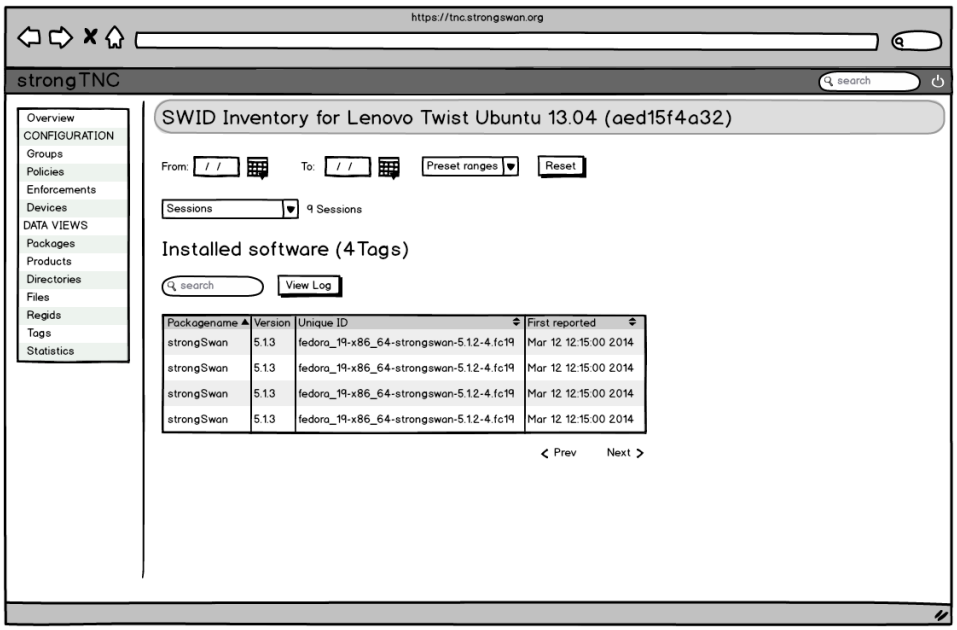

Abbildung 9.7: Mockup SWID Inventory

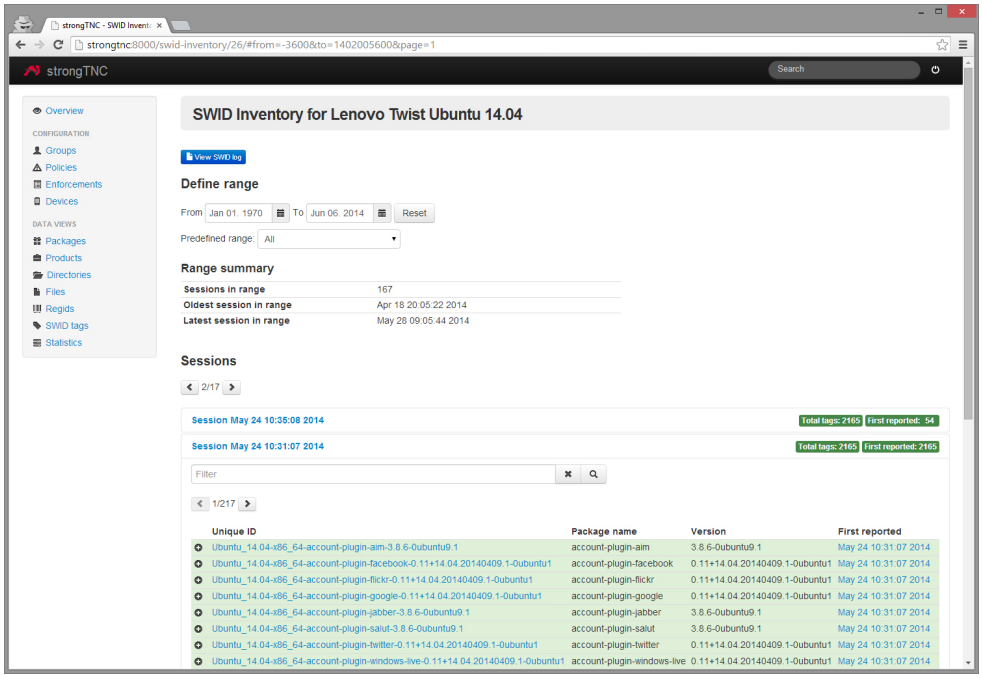

Abbildung 9.8: SWID Inventory View

9.2.4.4 SWID Log View

Die Log View zeigt den Verlauf der Installationen und Deinstallationen von Software Paketen für ein ausgewähltes Gerät (UC01: CMDB, Softwareinventar eines Gerätes).
Da bei jeder SWID Messung jeweils alle vorhanden SWID Tags übermittelt werden, lässt sich für jedes Paket sagen, wann es installiert, beziehungsweise entfernt wurde. Um diese Information zu erhalten, müssen zwei zeitlich aufeinanderfolgende Sessions verglichen werden, die Differenz der gemessenen Tags entspricht dem Installationsverlauf.

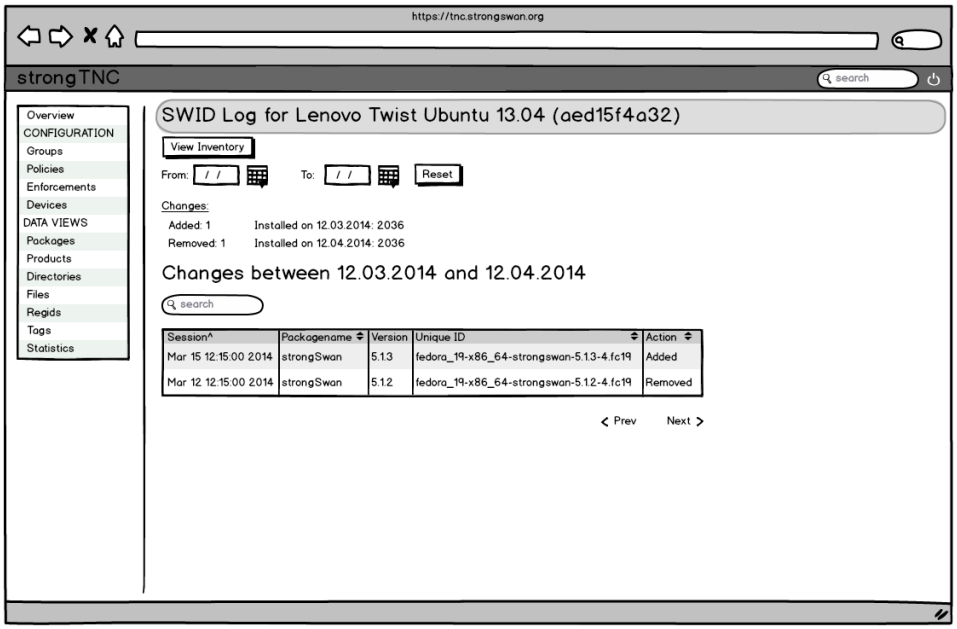

Abbildung 9.9: SWID Log View Mockup

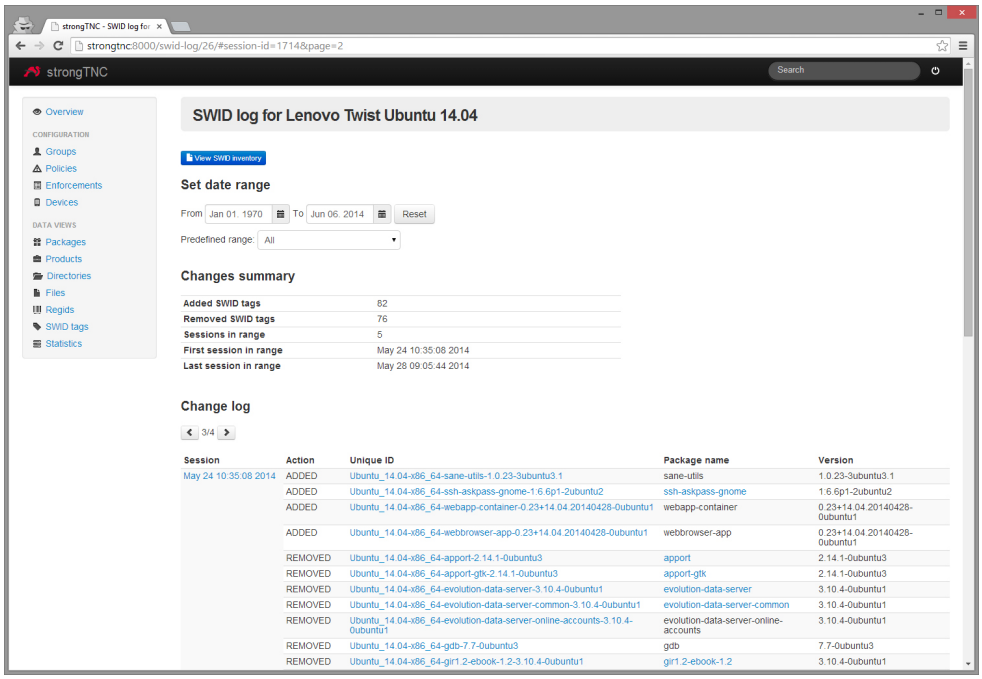

Abbildung 9.10: SWID Log View mit Details für eine ausgewählte Session

9.3 Erweiterungen

9.3.1 Qualität

9.3.1.1 Aufteilung in mehrere Django Apps

Bereits in der Analyse (Abschnitt 6.1.3) fiel auf, dass das bestehende strongTNC Projekt aus einer einzigen Django App besteht. Innerhalb der App waren alle Models in einer einzigen Datei enthalten (`models.py`), während die Views in mehrere thematisch gruppierte Dateien aufgeteilt wurden (`device_views.py`, `product_views.py`, etc).
Diese Aufteilung der Views deutet darauf hin, dass die `tncapp` App für mehrere thematisch unterschiedliche Bereiche zuständig ist, welche in der Codebasis separiert werden könnten und sollten. Die Aufteilung in mehrere Python-Dateien ist jedoch in Django Projekten der falsche Weg, um dies zu erreichen. Stattdessen bietet Django für solche Fälle das Konzept einer App an.

Eine App ist ein Verzeichnis innerhalb eines Django Projektes, welches seine eigenen Models, Views und Templates enthält. Die Apps werden in der Projektweiten `settings.py` registriert. Ein einzelnes Projekt lässt sich so in mehrere kleine MVC Container aufteilen.

Zusammenhängende Codeteile sollen wenn möglich in separate Apps unterteilt werden. Dadurch werden die Apps «pluggable» und können aktiviert- und deaktiviert werden kann. Die Separierung von Code in mehrere Apps führt zu tiefer Kopplung[18, S. 232–237] und dadurch zu bessert wartbaren Code.

Aus den oben genannten Gründen haben wir die bestehende monolithische App in folgende zehn moderat bis hoch kohäsiven Apps aufgeteilt:

- `core`: Enthält Elemente, die von fast allen Apps verwendet werden, wie beispielsweise das `Session` Model oder die `WorkItemType` Klasse.
- `front`: Enthält alle Elemente zur Darstellung der Seiten, welche nicht App-spezifisch sind. Beispiele hierfür sind das Base-Template, der `highlight` Template Filter oder der `pagination` Ajax Endpoint.
- `auth`: Enthält Hilfsfunktionen, Mixins und Template Tags für die Authentisierung und Berechtigungsprüfung.
- `api`: Enthält Hilfsfunktionen und Mixins der API.
- `devices`: Enthält Models wie `Product`, `Device` und `Group` sowie dazugehörige Views.
- `filesystem`: Enthält Models und Code mit Bezug auf das Dateisystem (`File`, `Directory`, etc).

- `packages`: Enthält Models und Code zu Softwarepakete und deren verfügbaren Versionen.
- `policies`: Enthält Models und Code mit Bezug zu Policies und deren Durchsetzung mit Hilfe von Enforcements.
- `swid`: Enthält die Implementation der SWID Integration in strongTNC.
- `tpm`: Enthält Models für die TPM Integration in strongTNC. Diese App bietet aktuell noch keine Funktionalität.

9.3.1.2 Verbesserung der Testing-Infrastruktur

Unit- und Integration-Tests sind zentral für die agile Entwicklung und für die Sicherstellung der Wartbarkeit eines Softwareprojektes. Es wurde Zeit und Aufwand in die Konfiguration der Testing-Umgebung investiert, um diese besser und flexibler zu machen.

Die ursprüngliche Codebasis verwendete das `unittest`[2] Modul aus der Python Standardbibliothek. Alle Tests befanden sich in einer einzigen Datei. Obgleich `unittest` eine stabile Lösung ist, fällt bei der Verwendung auf, dass das stark von Java Unittests inspirierte Projekt, relativ umständlich zu nutzen ist. Alle Tests müssen als Methoden in einer Klasse geschrieben werden. Um Assertions zu definieren, müssen ebenfalls Methodenaufrufe wie `assertEqual` oder `assertLessThan` verwendet werden. Nachfolgend ein Minimalbeispiel:

```
import unittest

class TestTheAnswer(unittest.TestCase):
    def testIt(self):
        self.assertEqual(19 + 23, 42)
```

Listing 9.31: Unittest Minimalbeispiel

In Python gibt es jedoch bereits ein `assert` Keyword, welches beliebige Assertions testen kann. Zudem ist es in Python üblich auf Klassen zu verzichten, wenn sie keinen Mehrwert bieten. Im Fall von Tests ist es oft sinnvoll , diese direkt als top-level Funktionen zu definieren und in mehrere Module zu organisieren.

Um die Situation zu verbessern wurde die Test-Infrastruktur von Unittest auf Pytest[3] umgestellt. Das Pytest Projekt beinhaltet Test-Discovery und -Ausführung, viele Hilfsfunktionen und Unterstützung für Plugins.

[2] `https://docs.python.org/2/library/unittest.html`
[3] `http://pytest.org/`

Nachfolgend dasselbe Minimalbeispiel wie in Listing 9.31, mit Pytest:

```
def test_the_answer():
    assert 19 + 23 == 42
```

Listing 9.32: Pytest Minimalbeispiel

Falls mehrere Tests auf dieselben Daten zugreifen, kann wie bei Unittest eine Klasse verwendet werden, es gibt jedoch noch eine weitere Möglichkeit: Fixtures. Eine Fixture ist eine Funktion, deren Rückgabewert über «Dependency Injection» einem Test übergeben wird. Eine selbst definierte Fixture könnte beispielsweise folgendermassen aussehen:

```
import pytest

@pytest.fixture
def database(self):
    conn = MyDatabaseConnection()
    return conn

def test_the_database(database):
    assert database.status() == 'OK'
```

Listing 9.33: Pytest Fixtures

Ein weiteres, von uns häufig genutztes Feature, ist die Parametrisierung von Tests. Ein «klassischer» Unittest könnte folgendermassen aussehen:

```
import pytest

def test_add():
    assert 19 + 23 == 42
    assert 13 + 29 == 42
    assert 42 + 1 == 42
    assert 62 - 23 == 42
```

Listing 9.34: Kombinierte Tests

Der Nachteil eines solchen Tests ist, dass obwohl vier verschiedene unabhängige Bedingungen getestet werden, in einem Fehlerfall der gesamte Test als *failing* angezeigt wird.

Pytest bietet hier als Abhilfe parametrisierte Tests. Dabei werden, mit Hilfe eines Decorators, aus einer einzelnen Funktion zur Laufzeit mehrere Tests generiert.

```
import pytest

@pytest.mark.parametrize(['a', 'b'], [
        (19, 23),
        (13, 29),
        (42, 1),
        (62, -23),
])
def test_add(a, b):
    assert a + b == 42
```

Listing 9.35: Parametrisierte Tests

In diesem Beispiel werden vier unabhängige Tests generiert. Drei davon sind erfolgreich, während der Vierte einen Fehler meldet, da die Bedingung `42 + 1 == 42` nicht erfüllt ist.

Um das Testen mit Django zu vereinfachen, wurde das `pytest-django` Plugin verwendet. Es wurde ein `runtests.py` Script geschrieben, welches die die Pfad-Variablen korrekt setzt, alle Tests ausführt und anschliessend die Test Coverage anzeigt. `pytest-django` liefert Fixtures, wie z. B. `client` um auf einen Django Test Client zugreifen zu können oder `transactional_db`, um transaktionellen Zugriff auf die Datenbank zu erhalten.

9.3.1.3 Continuous Integration

Eine Sicherstellung und langfristige Verbesserung der Softwarequalität ist nicht möglich, wenn diese nicht regelmässig überprüft wird. Um dieses Ziel zu erreichen, wurde in verschiedenen Bereichen auf «Continuous Integration» (CI) gesetzt: Tests, Testabdeckung und statische Analyse.

Ausführung der Tests

Um die Test Suite (Abschnitt 9.3.1.2) automatisiert auszuführen, wurde Travis CI[4] verwendet. Travis ist eine CI-Plattform mit Python-Unterstützung, die für Open Source Projekte kostenfrei ist. Mit wenigen Zeilen in einer Datei namens `.travis.yml` kann die Test-Ausführung konfiguriert werden. Ein Fehlerfall wird auf Github angezeigt und der Autor des entsprechenden Commits per E-Mail benachrichtigt.

[4] `https://travis-ci.org/`

Überwachen der Testabdeckung

Neben dem Testerfolg ist die Testabdeckung ebenfalls eine wichtige Kennzahl. Um diese laufend zu überwachen wurde Coveralls[5] verwendet. Coveralls bietet eine Integration mit Travis CI und zeigt die Entwicklung der Testabdeckung an. Dank einer Integration mit Github wird auch für jeden Pull Request die Veränderung der Testabdeckung gemessen.

Die Testabdeckung für strongTNC ist inzwischen bei 70% und stieg während der gesamten Projektdauer kontinuierlich an:

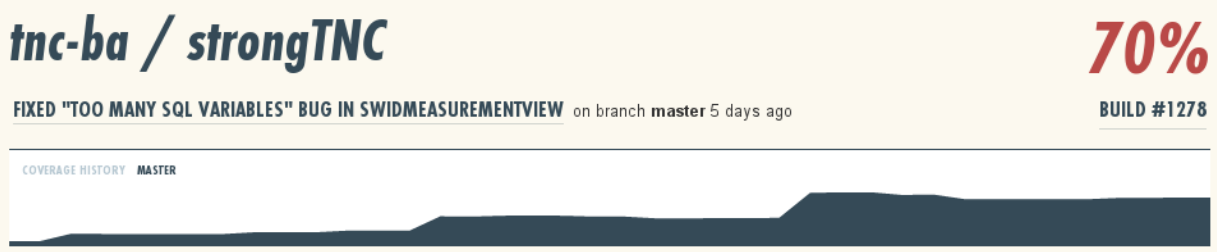

Abbildung 9.11: Entwicklung der Testabdeckung

Messen der Codequalität

Neben Testing-Werkzeugen sind auch statische Codeanalyse-Tools hilfreich zur Verbesserung der Codequalität. Sie erkennen häufige Schwachstellen im Code, wie beispielsweise ungenutzte Variablen oder unerreichbaren Code.

Während der Entwicklung wurde bereits auf Tools wie Pyflakes[6] und Pylint[7] gesetzt. Zusätzlich wurde das noch junge Projekt Landscape[8] eingebunden, dieses überprüft den Code indem verschiedene Checks automatisch nach jedem Commit ausgeführt werden. Den Report kann man über den Webbrowser betrachten und analysieren.

[5] `https://coveralls.io/`
[6] `https://pypi.python.org/pypi/pyflakes`
[7] `http://www.pylint.org/`
[8] `https://landscape.io/`

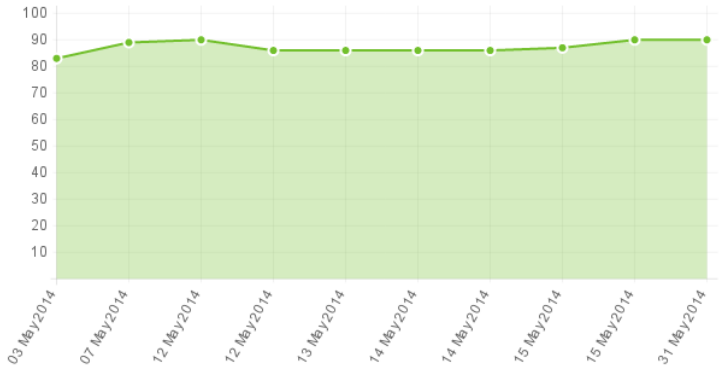

Abbildung 9.12: Coveralls: Entwicklung des Landscape Quality Scores

Auf Landscape liegt das Projekt aktuell bei einem Quality Score von 90%. Zu bemerken ist, dass Landscape einige False Positives liefert. Das liegt daran, dass das Projekt noch jung ist und gewisse Spezialfälle – welche in einer dynamischen Sprache wie Python häufig vorkommen können – nicht erkennt. Zudem sind gewisse Kennzahlen, wie die maximale Zeilenlänge, noch nicht konfigurierbar und werden deshalb als Fehler angezeigt, obwohl im Coding Styleguide (Abschnitt 15.1) andere Regeln vereinbart wurden. Damit sich diese Situation in Zukunft verbessert, wurden Feature Requests für die entsprechenden Anpassungen gestellt[9].

9.3.1.4 Deployment Dokumentation, Vagrant, Ansible

Wenn man strongTNC in einen produktiven Umgebung einsetzt, müssen einige Dinge beachtet werden. Beispielsweise sollte die Applikation nie ohne TLS über ein Netzwerk genutzt werden, da die API Endpoints nur mit Hilfe von Basic Auth geschützt sind.

Für eine einfache Installation haben wir eine detaillierte Deployment Dokumentation erstellt. Das Dokument findet sich im Repository unter `doc/deployment.rst`[10] sowie im Anhang dieser Dokumentation auf Seite 133.

Falls man strongTNC nur testen möchte, wurde eine Vagrant[11]- und eine Ansible[12]-Konfiguration vorbereitet.
Vagrant ist ein Tool um die Erstellung und Konfiguration von Virtuellen Maschinen zu automatisieren, während Ansible ein System zur Automatisierung von Deployments ist. Durch die

[9] `http://git.io/BqNWMw`, `http://git.io/O8MaNA`
[10] `https://github.com/strongswan/strongTNC/blob/master/docs/deployment.rst`
[11] `http://www.vagrantup.com/`
[12] `http://www.ansible.com/`

Kombination der beiden Tools kann mit folgenden zwei Befehlen eine komplett konfigurierte VM erstellt werden.

```
$ cd /path/to/strongtnc/vagrant/
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
...
```

Listing 9.36: Starten einer Vagrant Box

Alle weiteren Informationen befinden sich im Projekt-Repository unter `vagrant/README.rst`[13].

9.3.1.5 Weitere Punkte

Code Standards (PEP8) Damit die Einhaltung der PEP8 Richtlinien garantiert ist, wurde eine entsprechende Prüfung in den Build Prozess integriert. Dadurch gilt ein Build als fehlgeschlagen, falls die Richtlinien nicht vollständig eingehalten wurden.

Dynamisches Auflösen von URLs Gemäss Django Dokumentation soll das Verwenden von internen URLs in Templates und Views vermieden werden. Anstelle sollen URLs nur einmal, in der `urls.py` Datei, konfiguriert und danach anhand des Namens aufgelöst werden. Auf diese Weise können zentral Änderungen an URLs vorgenommen werden, ohne Templates oder Views anpassen zu müssen.

Konsequente Frontend-Validierung Die bestehende Frontend-Validierung wurde an diversen Stellen inkonsequent oder fehlerhaft implementiert. Diese Mängel wurden behoben und damit die Validierung vereinheitlicht.

Upgrade auf Django 1.6 strongTNC wurde von Django 1.5 auf Django 1.6 aktualisiert. Die Codebasis ist so zukunftssicherer, kann neue Features nutzen und beinhaltet alle aktuellen Security-Fixes.

9.3.2 Features

9.3.2.1 Aufteilung der Zugriffsrechte

Um die Informationen in strongTNC zugänglich zu machen, ohne dass Änderungen vorgenommen werden können, soll eine Readonly Rolle eingeführt werden.

[13] `https://github.com/strongswan/strongTNC/blob/master/vagrant/README.rst`

In einem ersten Schritt soll ein Modell mit zwei Rollen umgesetzt werden. Eine Rolle für den readonly Zugriff und eine für den Vollzugriff. Die Rollen sollen nicht personalisiert sein. Die Möglichkeit, das Login zu personalisieren sollte aber nicht grundsätzlich ausgeschlossen werden.

Technische Umsetzung
Das Django Webframework besitzt ein vollständiges Authentication- und Permission-System, damit lassen sich komplexe Permission-Szenarien abbilden. Es besteht die Möglichkeit, nur Teile davon zu verwenden und so ein einfaches Rollen-Modell zu umzusetzen.

Es werden zwei technische Benutzer erstellt. Die Namen der beiden Benutzer sind `admin-user` und `readonly-user`. Dem `admin-user` wird eine `write_access` Berechtigung erteilt. Diese kann in den Views mittels Django-Internem `@permission_required()` Decorator überprüft werden.

Diese Variante der Umsetzung erlaubt es, mit wenig Aufwand dennoch personaliserte Logins einzuführen. Es muss keine eigene Benutzerverwaltung entwickelt werden, da Django ein Admin-Interface zur Verwaltung von Benutzern zur Verfügung stellt.

In den Templates können die Zugriffsrechte über das `perms` Objekt geprüft werden:

```
{% if 'auth.write_access' in perms %}
    <p>Read-write access.</p>
{% endif %}
```

Listing 9.37: Überprüfung der Zugriffsrechte in Templates

Die zusätzliche Implementation eines Template Tags erleichtert die Steuerung von HTML Formularelementen, deren Bedienung nur für schreibberechtigte Benutzer erlaubt sein soll:

```
@register.simple_tag(takes_context=True)
def input_editability(context):
    perms = context.get('perms', [])
    if 'auth.write_access' not in perms:
        return 'disabled="disabled"'
    return ''
```

Listing 9.38: Template Tag zur Zugriffssteuerung von HTML Form Elementen

Die Anwendung sieht wie folgt aus:

```
<input type="text"name="name" value="{{ group }}" {% input_editability %} />
```

Listing 9.39: Anwendung des Template Tags zur Zugriffssteuerung von HTML Form Elementen

Diese Umsetzung folgt einer Variation, des in «Security Patterns»[19] beschriebenen Entwurfsmusters «Full Access With Errors». Alle Funktionen sind stets für den Benutzer sichtbar. Dadurch kann das User Interface einheitlich gestaltet werden und vermeidet auf diese Weise mögliche Verwirrungen. Diejenigen Elemente, die der Benutzer nicht bearbeiten darf, werden deaktiviert.

Um die beiden benötigten Benutzer initial zu erstellen, wurde das Django Management-Kommando `setpassword` angepasst. Das Kommando erstellt direkt die beiden Benutzer und erlaubt es, die Passwörter interaktiv oder per Parameter festzulegen.

Alternativen
Als Alternative zum Django Permission-System könnten auch die standardmässig vorhandenen Flags `user.is_admin` und `user.is_staff` verwendet werden. Dies wäre bedeutend einfacher zu implementieren, jedoch ist die Lösung nicht ausbaufähig. Deshalb wurde das Permission-System bevorzugt.

9.3.2.2 Nachladen von grossen Datenmengen

strongTNC muss mit grossen Datenmengen umgehen können. Wenn Datei-Messungen durchgeführt- oder SWID Tags erfasst werden, erreicht beispielsweise die Zahl der erfassten Dateien schnell den fünfstelligen Bereich. Wenn einzelne dieser Dateien ausgewählt werden sollen, muss verhindert werden, dass immer die gesamte Menge der Daten geladen wird.

Technische Umsetzung
Mittels Ajax Requests werden jeweils nur die Daten nachgeladen, welche benötigt werden. Die Ajax Endpunkte wurden mit Hilfe der «Dajax-ice»[14] Django Erweiterung realisiert. «Dajax-ice» stellt die nötigen Bibliotheken bereit, um Ajax Endpunkte wie gewöhnliche Views zu implementieren. So werden zum Beispiel benötigte HTTP Header automatisch gesetzt und geprüft. Ausserdem generiert «Dajax-ice» Javascript Funktionen, welche clientseitig die benötige

[14] `http://www.dajaxproject.com/`

Infrastruktur bereitstellen, um Ajax Requests auszuführen.

In Listing 9.40 ist die Beispielimplementation eines Ajax Endpoints zu sehen. Um auch bei Ajax-Anfragen eine Authentisierung zu erzwingen, wurde ein eigener Decorator mit dem Namen `@ajax_login_required` erstellt.

```
@dajaxice_register
@ajax_login_required
def directories_autocomplete(request, search_term):
    dirs = Directory.objects.filter(path__icontains=search_term)
    results = {'results': [{'id': d.id, 'directory': d.path} for d in dirs]}
    return json.dumps(results)
```

Listing 9.40: Beispiel eines Ajax Endpunktes

Clientseitig kann der Endpunkt wie folgt mit Javascript angesprochen werden:

```
Dajaxice.apps.filesystem.directories_autocomplete(callback, {search_term: 'system'});
```

Listing 9.41: Absenden eines Ajax Requests

Damit bei der inkrementellen Suche nicht bei jedem Tastenanschlag ein Request abgesetzt wird, wurde ein Verzögerungsmechanismus implementiert, welcher erst einen Request absetzt, wenn für eine bestimmte Zeit keine Eingabe mehr gemacht wurde (angelehnt an Nagle's Algorithmus [20]). Durch die Reduktion der Anzahl HTTP Requests, wird auch die Anzahl der teilweise komplexen und zeitintensiven Datenbankabfragen minimiert.

Langlaufende Operationen / Fehlerbehandlung
Asynchrone Aufrufe bringen zwei wesentliche Probleme mit sich. Zum einen kann es sein, dass Fehler auftreten, die der Benutzer nicht bemerkt. So zum Beispiel Timeouts. Zum Anderen muss für den Benutzer ersichtlich sein, ob Datenübertragungen im Hintergrund anstehen und falls ja, wann diese erfolgreich abgeschlossen sind. Speziell für langlaufende Operationen durch datenbankintensive Abfragen oder grosse Datenmengen, können Antwortzeiten im Sekundenbereich entstehen.
Zu diesem Zweck wurde eine Proxy Komponente [12] implementiert, welche anstelle des eigentlichen Dajax-ice Aufrufes eingesetzt wird. Komponenten, deren Inhalt per Ajax nachladen wird, werden mit einem Statusindikator besetzt. Zusätzlich wurde ein globaler Statusindikator eingeführt, der anzeigt, ob mindestens eine asynchrone Datenübertragung stattfindet.

Treten Fehler bei einer Ajax Datenübertragung auf, wird der Benutzer über den Fehler informiert.

9.3.2.3 Ajax Pagination

Um grosse Datenmengen besser handhaben zu können, wurde die bestehende statische Pagination durch eine dynamische, Ajax basierte Version ausgetauscht. Das Ziel war, jegliche Art von Daten seitenweise auszuliefern, sowie die Datenmenge mit Hilfe von Filtern dynamisch zu reduzieren. Bei der Architektur der Pagingkomponente wurde darauf geachtet, dass diese wiederverwendet werden kann. Der Inhalt, der seitenweise ausgeliefert werden soll, ist von der Pagination entkoppelt und kann mittels einer Art Strategie («Strategy Pattern»[12]) festgelegt werden.

Technische Umsetzung
Die Implementation der Pagination besteht aus den folgenden fünf Teilen:

1. Javascript Frontend Komponente
2. Template Tag, Templates und HTML Markup
3. Generischer Ajax Endpunkt
4. Individuelle Konfiguration pro Pagination Block
5. «List Producer» für die Datenaufbereitung

Abbildung 9.13 zeigt den Grobablauf beim dynamischen Anfordern einer neuen Seite.

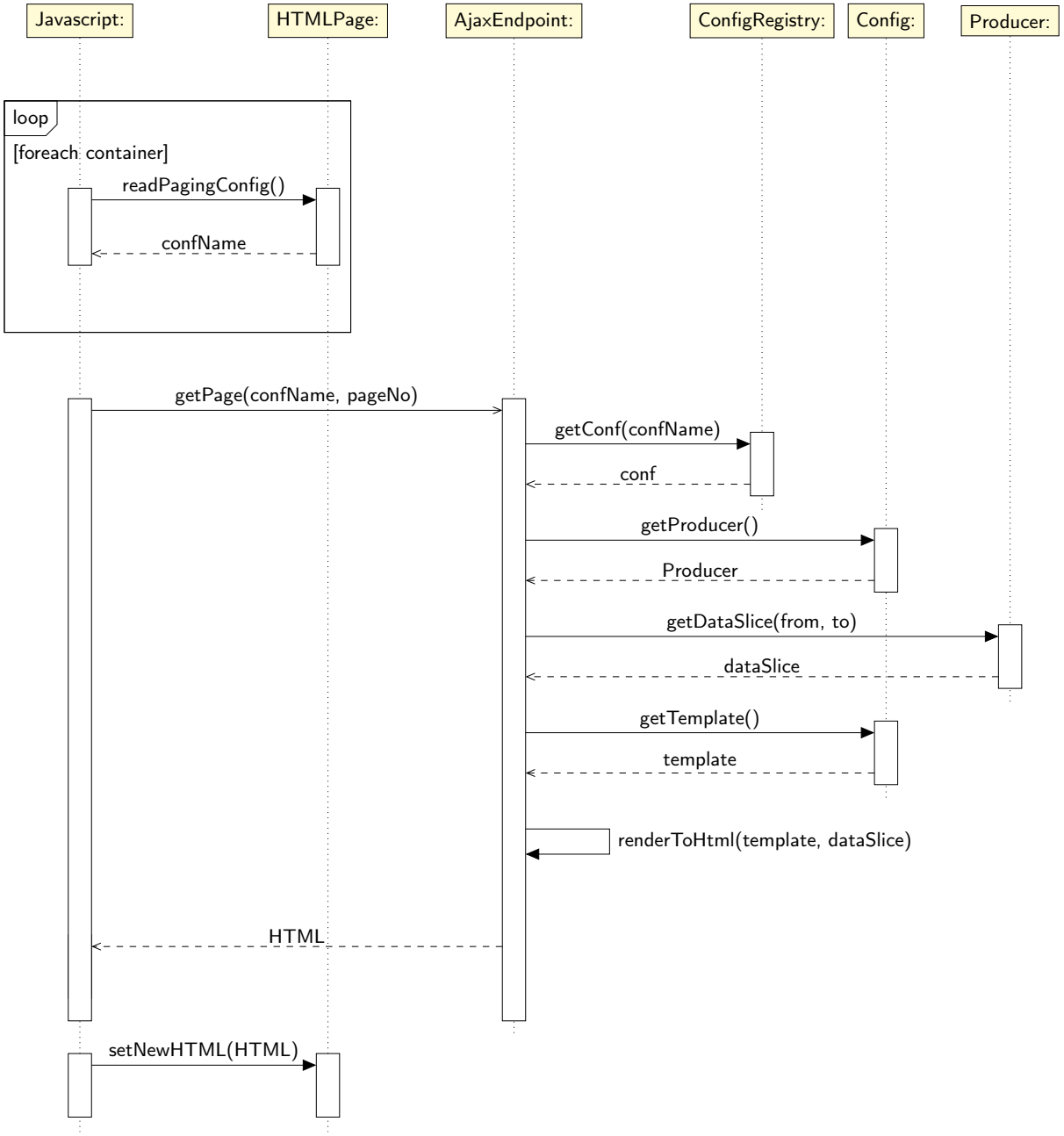

Abbildung 9.13: Sequenzdiagramm, Grobablauf

HTMLPage Auf einer HTML Seite werden Container definiert, welche später mit den Seiteninhalten befüllt werden. Die Container werden mittels, des dafür kreierten, Template Tags (Listing 9.43) generiert.

```
<div class="ajax-paged"
  data-config="regid_list_config"
  data-args="{}"
  data-initial="True"
  data-urlparams="True"
  data-filter="True">
    [...]
    <div class="paged-content"></div>
    [...]
</div>
```

Listing 9.42: Generiertes HTML Markup des Template Tags

```
{% paged_block config_name="regid_list_config" with_filter=True %}
```

Listing 9.43: Verwendung eines «Paged Blocks» mittels Template Tag

Javascript Eine Aufgabe der Javascript Komponente ist das Auslesen aller Paged Block Konfigurationen aus dem DOM der aktuellen HTML Seite. Identifiziert werden diese Blöcke durch die CSS Klasse `ajax-paged`. Es können mehrere Paged Block Komponenten pro Seite definiert werden. Für jeden der gefunden Paging Container wird ein Javascipt Pager-Objekt instanziert, welches für die Verwaltung des Zustandes – zum Beispiel die aktuelle Seite – und die Kommunikation mit dem AjaxEndpoint verantwortlich ist. Nach erfolgreicher Kommunikation mit dem AjaxEndpoint wird der zurückgelieferte HTML Block in den entsprechenden Container geladen.

AjaxEndpoint Der Ajax Endpoint nimmt Requests entgegen und kann anhand des übergebenen Konfigurationsnamens die zugehörige Konfiguration laden. Diese beinhaltet einen Producer und die Referenz auf ein Template. Die Datengenerierung wird an den Producer delegiert, dieser übernimmt auch die Unterteilung der Daten in die Angeforderte Seitengrösse. Die Daten, welche der Producer zurück gibt werden anschliessend als Kontext verwendet, um das Template zu HTML zu rendern. Durch diese Separierung wird eine hohe Wiederverwendbarkeit erreicht. Verschiedene Datenquellen und Templates können kombiniert und wiederverwendet werden.

```
regid_list_paging = {
    'template_name': 'front/paging/default_list',
    'list_producer': regid_producer_factory.list(),
    'stat_producer': regid_producer_factory.stat(),
    'url_name': 'swid:regid_detail',
    'page_size': 50,
}
```

Listing 9.44: Beispiel einer Pagination Config

Producer Die Datenaufbereitung, Datenbankabfragen, Slicing der Daten in Seiten und vorgängiges Filtern sind im Producer definiert. Durch ein gemeinsames Producer Interface (impliziter Contract[9]), können Producer flexibel ausgetauscht und wiederverwendet werden. Eine konkrete Implementation eines Producers ist in Listing 9.45 zu sehen.

```
def swid_files_list_producer(from_idx, to_idx, filter_query, dynamic_params, static_params=None):
 if not dynamic_params:
     return []
 tag_id = dynamic_params['tag_id']
 return Tag.objects.get(pk=tag_id).files.all()[from_idx:to_idx]
```

Listing 9.45: Beispielimplementation eines Producers

Für generische List Producer, welche lediglich eine Liste von Objekten eines bestimmten Models liefern, wurde eine eine «Factory» erstellt:

```
swid_producer_factory = ProducerFactory(Tag, 'unique_id__icontains')
swid_producer_factory.list()
```

Listing 9.46: Producer Factory

9.3.2.4 Models für MySQL Support

Datenbankwechsel
Die enge Kopplung von strongSwan und strongTNC durch eine gemeinsam genutzte Datenbank, hat zur Folge, dass Anpassungen des Datenbankschemas, Änderungen an allen Komponenten bedingen. Des Weiteren ist zu erwarten, dass für grössere Installationen SQLite zu wenig Funktionalität bietet. Optimierungen und Profiling sind nur eingeschränkt durchführbar, zudem wird keine referenzielle Integrität unterstützt. Problematisch ist auch, dass SQLite nicht mehrbenutzerfähig ist.

Generierung des Datenbankschemas
Django unterstützt diverse Datenbankbackends[15], unter anderem MySQL, PostgreSQL, MSSQL und das zur Zeit verwendete SQLite. Wir empfehlen das SQLite Backend möglichst bald durch ein anders Datenbankbackend zu ersetzen. Aus diesem Grund wurden die Django Models so angepasst, dass sie dem aktuellen Datenbankschema entsprechen. Das Django ORM kann daraus datenbankspezifisches SQL-DDL generieren und so die Migration erheblich erleichtern.

9.3.2.5 Konfigurationsdateien

Um beim Deployment von strongTNC den Code und die Konfiguration zu trennen, sollte man die Möglichkeit haben, wichtige Einstellungen wie Datenbank-Connection-Strings, in einer nicht versionierten Datei zu konfigurieren. Drei verbreitete Möglichkeiten sind:

- Konfiguration über Umgebungsvariablen, wie es durch die «12 Factor App» Methodologie empfohlen wird[16].
- Eine `settings_local.py` Datei, welche alle Konfigurationen von der regulären `settings.py` Datei importiert und zu konfigurierende Werte überschreibt.
- Eine `settings.ini` Datei, welche durch die `ConfigParser` Klasse aus der Python Standardbibliothek geparst wird.

Die erste Methode ist gut geeignet, wenn man seine Applikation auf einem Cloud-Hosting Provider wie Heroku[17] deployen will. Für Systemadminstratoren, die sich Konfigurationsdateien gewöhnt sind, ist dies allerdings etwas umständlich.

Die zweite Möglichkeit bietet grosse Flexibilität, weist jedoch auch Gefahren auf, da in einer solchen Datei beliebiger Code ausgeführt werden kann.

[15] `https://docs.djangoproject.com/en/1.6/ref/databases/`
[16] `http://12factor.net/config`
[17] `http://www.heroku.com`

Deshalb wurde der dritte Ansatz gewählt: Eine `settings.ini` Datei, welche lediglich nicht-ausführbare Konfigurationsdaten enthält. Die Datei muss sich entweder im Projektverzeichnis oder im `/etc/strongTNC/` Verzeichnis befinden. Eine Beispielkonfiguration mit dem Namen `settings.sample.ini` wird mit strongTNC mitgeliefert.

9.3.2.6 Weitere Punkte

Django Admin-Interface Django bietet die Möglichkeit, die Datenbank über ein automatisch generiertes Admin-Interface zu verwalten. Auf diese Weise können potentiell fehleranfällige Datenmanipulationen per SQL vermieden werden. Die Models wurden mit den nötigen Metadaten erweitert, damit dieses Feature aktiviert werden konnte.

Erstellen von Files und Versions Bisher war es nicht möglich über die Oberfläche von strongTNC neue Files und neue Versionen zu erstellen. Die Ansichten wurden um diese Funktionalität ergänzt.

Verknüpfung von Daten in Views Die Views wurden durch sinnvolle Verknüpfungen ergänzt. Die Verknüpfungen waren bereits in der Datenbank vorhanden, jedoch in den Views nicht direkt ersichtlich, so zum Beispiel welche Enforcements einer Gruppe zugeordnet sind.

URL Hash Parameter Um Parameter in der URL abzulegen und auszulesen, wurde eine Javascript Hilfsbibliothek erstellt. Die Bibliothek erlaubt es, Parameter in die URL zu schreiben ohne einen Reload auszulösen. Ausserdem kann auf Änderungen eventbasiert («Observer Pattern» [12]) reagiert werden. Änderungen welche mittels Ajax ausgelöst und somit keinen Reload zur Folge haben, können so über die URL und die Browser History zugänglich gemacht werden.

Umstellung auf UTC Die bestehende Konfiguration von strongTNC hatte keine Unterstützung für Zeitzonen. Dies stellt ein Problem dar, wenn Clients aus verschiedenen Zeitzonen mit strongTNC kommunizieren. Mit dieser Anpassung bietet strongTNC vollständige Unterstützung für Zeitzonen und stellt alle Zeiten gemäss der konfigurierten Zone dar.

10 API Konzept

10.1 Trennung

Um strongTNC von strongSwan zu trennen und die gemeinsame Datenbank abzulösen, wurde im Rahmen dieser Arbeit ein API Konzept erarbeitet. Das Konzept orientiert sich dabei am «REST» Architekturstil, welcher von Roy Fielding in seiner Dissertation aus dem Jahr 2000 erstmals definiert wurde[21].

In diesem Abschnitt werden wichtige Punkte des API Konzeptes genauer betrachtet. Das vollständige Konzept ist im Anhang (Abschnitt 15.6 ab Seite 139) zu finden. Detaillierte Beschreibungen zur verwendeten Syntax, sowie die Definition der Archetypes, sind ebenfalls im Konzept zu finden.

10.2 Vorgehen

Das Ziel der REST API ist eine vollständige Ablösung der gemeinsamen Datenbank als Schnittstelle. Um einen Überblick über den benötigten Umfang einer TNC Policy Manager Schnittstelle zu bekommen, wurden die Datenbankzugriffe im IMV Code der strongSwan Implementation betrachtet. Dazu wurden sämtliche SQL abfragen im IMV Quellcode gesammelt und analysiert. Das Ergebnis war eine Liste der Tabellen, auf die schreibend oder lesend zugegriffen wurde. Ebenso konnte ermittelt werden, welche Tabelle über ein SQL `JOIN` Statement verknüpft wurden. Basierend auf diesen Daten konnte ein Set von REST Endpunkten definiert werden, mit dem es möglich ist, sämtliche aktuell bestehenden Datenbankzugriffe abzudecken.

10.3 Documents und Collections

Für alle Elemente der strongTNC Domäne wurden REST Ressourcen von den Archetypes «Document» und «Collection» konzipiert. Oft handelt es sich dabei um vollständige CRUD Ressourcen, wie beispielsweise bei der `Version` Ressource sichtbar ist:

URI Path `/versions/{id}/`
Archetype Document
Methods GET, PUT, PATCH, DELETE
JSON Format Response

```
{
    "id": 5,
    "uri": "https://strongtnc/api/versions/5/",
    "package": "https://strongtnc/api/packages/42/",
    "product": "https://strongtnc/api/products/23/",
    "release": "5.0.2-2.2+squeeze1",
    "securtiy": true,
    "blacklist": false,
    "time": 1402061820
}
```

Listing 10.47: REST Document für Versions

URI Path `/versions/`
Archetype Collection
Filter Query
productName `<str,product-name>`
packageName `<str,package-name>`
Methods GET, POST
Response List of Version documents

Listing 10.48: REST Collection für Versions

Die angebotenen Operationen, beziehungsweise HTTP Methoden, wurden immer so weit wie möglich eingeschränkt, so gibt es diverse Ressourcen die kein `DELETE` anbieten oder nur lesenden Zugriff erlauben. Dadurch soll erreicht werden, dass die Schnittstelle nicht zu offen wird oder durch das Anbieten von nicht benötigten Operationen unnötig komplex erscheint.

Um den `JOIN` Verknüpfungen in einer ressourcenorientierten Weise Rechnung zu tragen, wurden

verschiedene virtuelle Ressourcen geschaffen. Beispielsweise lassen sich die `sessions` eines Gerätes direkt auf der `devices` Ressource abfragen:

URI Path `/device/{id}/sessions/`
Archetype Readonly Collection
Filter Query
timeFrom `<int,timestamp>`
timeTo `<int,timestamp>`
Methods GET
Response List of Session documetns

Listing 10.49: Readonly Collection für Devices

10.4 Controller

Komplexere Abläufe, welche bei der gegenwärtigen Schnittstelle direkt mit mehreren Datenbankzugriffen ablaufen, werden im REST Konzept durch «Controller» abgebildet. Zu diesen Abläufen gehören das Starten und Stoppen einer `Session`. Ebenfalls wurde die bisher noch nicht existierende Funktion der SWID Tag Messung, beschrieben in Abschnitt 9.2, als «Controller» umgesetzt.

10.4.1 Session Steuerung und Ablauf

Das grundsätzliche Konzept der Session Steuerung wurde gegenüber der jetzigen Schnittstelle nicht geändert, es werden weiterhin sogenannte «Workitems» verwendet, um anzuzeigen, welche Messungen durchgeführt werden müssen. Der Ablauf einer Session nach dem gegenwärtigen Verfahren kann der Abbildung 6.2 entnommen werden. Ausführliche Informationen dazu sind in der Vorgängerarbeit «Cygnet»[2] zu finden.

Folgendes ist der Aufbau der Endpunkte für den Sessionstart und das Beenden der Session:

URI Path `/sessions/start/`
Archetype Controller
Methods POST
Request Parameter

`connectionId` strongSwan Connection ID
`clientIdentity` strongSwan Client-Identity
`hardwareId` Die ID, welche das Gerät identifiziert, so zum Beispiel, AIK, Android-ID, DBUS machine-ID, o.ä. Dies entspricht dem `value` Feld in der `device` Tabelle in der Datenbank
`productName` Der Productname ist der Name des OS wie er in der `product` Tabelle der Datenbank steht

JSON Format Response

```
{
    "sessionId": 420,
    "workitems": [
        {
            "id": 5,
            "uri": "https://strongtnc/api/sessions/420/workitems/5/",
            "session": "https://strongtnc/api/sessions/420",
            "type": 15,
            "argument": {
                "swidFlags": [
                    "R"
                ]
            }
        }
    ],
    "uri": "https://strongtnc/api/sessions/420"
}
```

Listing 10.50: Controller zum starten einer Session

Dieser Controller erstellt und startet eine Session, das Device, welches der Session zugeordnet werden soll wird anhand der `hardwareId` und dem `productName` bestimmt. Falls eines der Objekte noch nicht existiert, wird dieses durch den Controller erstellt.

Die ID, die im Response Dokument zurück geliefert wird, dient zur zukünftigen Identifikation der soeben gestarteten Session. Ausserdem wird eine Liste von Workitems zurückgegeben, die für diese Session abgearbeitet werden müssen.

Um aktive und vergangene Sessions abzufragen existiert eine readonly Collection. Die Session Documents sind ebenfalls als readonly definiert. Sessions sollen nicht direkt geändert werden, sondern nur über die entsprechenden Controller. Der Grund dafür ist, dass im Hintergrund noch zusätzliche Operationen vorgenommen werden müssen.

Dasselbe gilt für die Workitems, für diese sind ebenfalls eine readonly Collection mit readonly Documents definiert. Worktitems werden durch den Session-Start Controller anhand der Enforcements eines Devices automatisch erstellt, darum dürfen diese nicht manuell verändert werden.

Die Messresultate müssen für jedes Workitem erfasst werden, dafür steht eine virtuelle Ressource zur Verfügung:

URI Path `/sessions/{id}/workitems/{id}/result/`
Archetype Document
Request Parameter
`recommendation` Resultat/Empfehlung für dieses Workitem.
`comment` Kommentar zum Resultat.
Methods GET, POST
Response Statuscodes
201 Created Resultat wurde erfolgreich gespeichert.
409 Conflict Resultat existiert bereits.
JSON Format Response

```
{
    "recommendation": 0,
    "comment": "received inventory of 2165 SWID tag IDs and 0 SWID tags"
}
```

Listing 10.51: Result Document auf der Workitem Resource

Da eine Session nach dem Abschluss nicht mehr verändert werden soll, sind die Workitems flüchtig. Wenn die dazugehörige Session beendet wird, werden die Workitems entfernt und die eingetragenen Resultate auf die Session übertragen. Auf diese Weise können die Messresultate einer Session dauerhaft nachvollzogen werden. Über den Verweis auf das Enforcement kann die Messung auch dem ursprünglichen Zweck zugeordnet werden:

URI Path `/sessions/{id}/results/{id}/`
Archetype Readonly Document
Methods GET
JSON Format Response

```
{
    "id": 5,
    "uri": "https://strongtnc/api/sessions/420/results/5/",
    "enforcement": "https://strongtnc/api/enforcements/13/",
    "recommendation": 0,
    "comment": "received inventory of 2165 SWID tag IDs and 0 SWID tags"
}
```

Listing 10.52: Readonly Document für Results

URI Path `/sessions/{id}/results/`
Archetype Readonly Collection
Methods GET
Response List of Result documents

Listing 10.53: Readonly Collection für Results

Über folgenden Endpunkt kann eine Session abgeschlossen werden. Im Normalfall wird dies gemacht, wenn alle Workitems abgearbeitet sind, dies ist allerdings keine Voraussetzung, eine Session kann jederzeit abgeschlossen werden. Beim Abschliessen wird die Recommendation mitgegeben.

URI Path `/sessions/{id}/end/`
Archetype Controller
Methods POST
Request Parameter
`recommendation` Endgültiges Resultat/Empfehlung für diese Session.

Listing 10.54: Controller zum beenden einer Session

Wenn eine Session abgeschlossen wird, werden alle Workitems dieser Session abgeräumt und die jeweiligen Resultate auf die Session übertragen.

10.5 Implementation

Die im Rahmen dieser Arbeit konzipierte HTTP Schnittstelle wurde teilweise als Proof of Concept umgesetzt. Da die Entwicklung einer HTTP Schnittstelle für strongTNC nicht Bestandteil der Aufgabenstellung war, wurde aus Zeitgründen nicht der gesamte Umfang implementiert.

Für die Implementation wurde das «Django REST Framework»[1] (DRF) gewählt. Eine Evaluation der vorhandenen REST Frameworks für Python ergab zwei nahezu gleichwertige Kandidaten: Django REST Framework und Django-Tastypie [22]. Wir haben uns aus folgenden Gründen für DRF entschieden:

- DRF nutzt Class Based Views und Mixins, was näher am Django-Paradigma liegt als Tastypie. Zudem erhöht dieser Architekturansatz die Flexibilität und Anpassbarkeit, da eigene Klassen einfach durch Komposition und Spezialisierung erzeugt werden können.
- DRF liefert im Gegensatz zu Tastypie einen API Browser mit, inklusive Unterstützung für POST, PUT und DELETE Requests.
- Unterstützung für Basic Auth, Digest Auth, OAuth1 und OAuth2 wird mitgeliefert, HTTP Signatures sind über ein Plugin nutzbar.
- Ausgezeichnete Dokumentation und hilfsbereiter Support via IRC.
- DRF wird unter anderem von Mozilla und Eventbrite verwendet. Dies zeugt von einer gewissen Stabilität und Reife des Projektes.

10.5.1 Serialisierung

Die Django Models müssen für die Übermittlung in ein passendes Format serialisiert werden. Das von Clients gewünschte Format wird durch den `Accept` Header spezifiziert. DRF implementiert Standard-Serialisierer für verschiedene Formate wie JSON und XML, welche direkt verwendet werden können.

Wie im REST Konzept beschrieben, wäre es hilfreich, den Benutzern der API die Möglichkeit anzubieten, über Filter nur ausgewählte Felder der Datenstrukturen zu erhalten. Das heisst, es sollen nicht immer alle Felder eines Objektes serialisiert werden, sondern nur ein Subset davon. Dies ist vergleichbar mit einer SQL Projektion (`SELECT feld1, feld2, ...`). DRF bietet diese Möglichkeit standardmässig nicht. Um dieses Verhalten zu erreichen wurde eine Komponente entwickelt, welche die Auswahl der Felder durch Query-Parameter in der URL erlaubt. Der Aufruf von `https://strongtnc/api/swid-tags/2/?fields=packageName,version` liefert

[1] `http://www.django-rest-framework.org/`

beispielsweise nur die Felder `packageName` und `version` des abgefragten SWID-Tags. Realisiert wurde die Erweiterung durch ein Mixin mittels «Mixin-based Inheritance»[23].

Serverseitig werden die Model-Serialisierer durch die Klasse `DynamicFieldsMixin` erweitert. Diese extrahiert vor dem Serialisieren die Parameter aus dem Request und übergibt sie dem Serialisierer (Listing 10.55). Die Mixin-Klasse ist vollständig entkoppelt von der Implementierung des Serialisierungs-Mechanismus und so kann für alle Serialisierer verwendet werden.

```
class TagSerializer(DynamicFieldsMixin, serializers.HyperlinkedModelSerializer):
    entities = EntityRoleSerializer(source='entityrole_set', many=True)
```

Listing 10.55: Erweiterung durch `DynamicFieldsMixin` zur Abfrage bestimmter Felder

10.5.2 Fehlerbehandlung

Um für API Benutzer die Fehlerbehandlung zu erleichtern, werden bei auftretenden Fehlern detaillierte Informationen im Body der Response ausgeliefert. Ein möglicher Fehler ist beispielsweise das Übermitteln eines ungültig formatierten JSON Objektes.

```
HTTP/1.0 400 BAD REQUEST
Content-Type: application/json
Vary: Accept
Allow: POST, OPTIONS

{
    "detail": "JSON parse error - Expecting , delimiter: line 1 column 27 (char 27)"
}
```

Listing 10.56: Fehlerinformation beim Übermitteln eines ungültigen JSON Objektes

10.5.3 Verlinkte Ressourcen

Meist enthalten Objekte nicht nur einfache Attribute sondern auch Verweise auf andere Objekte. Zum Beispiel referenziert ein SWID Tag mindestens eine Entity. Bei einer Entity handelt es sich wiederum um ein Objekt mit weiteren Attributen.

Solche entfernten Objekte werden standardmässig nicht serialisiert. Anstelle des serialisierten Objektes wird eine URI ausgeliefert, über welche das entsprechende Objekt abgefragt werden kann. Das API Konzept sieht jedoch vor, einen Parameter einzuführen, welcher es erlaubt, die Serialisierungstiefe zu bestimmen.

Das Django REST Framework bietet standardmässig die Möglichkeit, für einen solchen Parameter (z. B. `depth=2`) mitzugeben. Allerdings funktioniert dies standardmässig nur für Vorwärts-Beziehungen, nicht für Rückwärts-Beziehungen. Wenn beispielsweise eine `Version` einen Fremdschlüssel auf ein `Package` hat, können `Packages` in die Repräsentation einer `Version` eingebettet werden, jedoch nicht umgekehrt.

Es wäre zwar möglich diese Rückwärtsbeziehungen manuell in den Serialisierern zu definieren. Aus zeitlichen Gründen konnte diese Erweiterung jedoch im Rahmen dieser Arbeit nicht implementiert werden.

```
HTTP/1.0 200 OK
Content-Type: application/json
Vary: Accept
Allow: GET, HEAD, OPTIONS

{
    "id": 2,
    "uri": "https://strongtnc:8000/api/swid-tags/2/",
    "packageName": "account-plugin-facebook",
    "version": "0.11+14.04.20140409.1-0ubuntu1",
    "uniqueId": "Ubuntu_14.04-x86_64-account-plugin-facebook-0.11+14.04.20140409.1-0ubuntu1",
    "entities": [
        {
            "entity": "https://strongtnc/api/swid-entities/11/",
            "role": 2
        }
    ],
    "swidXml": "<?xml version='1.0' encoding='UTF-8'?>\n<SoftwareIdentity ..."
}
```

Listing 10.57: Serialisierter SWID-Tag

10.5.4 Browsable API

Das Django REST Framework bietet die Möglichkeit, die API mittels Browser zu durchsuchen und POST/PUT/DELTE Requests abzusetzen. So können Entwickler die Programmier-Schnittstelle testen, ohne dass dafür Code geschrieben werden muss.

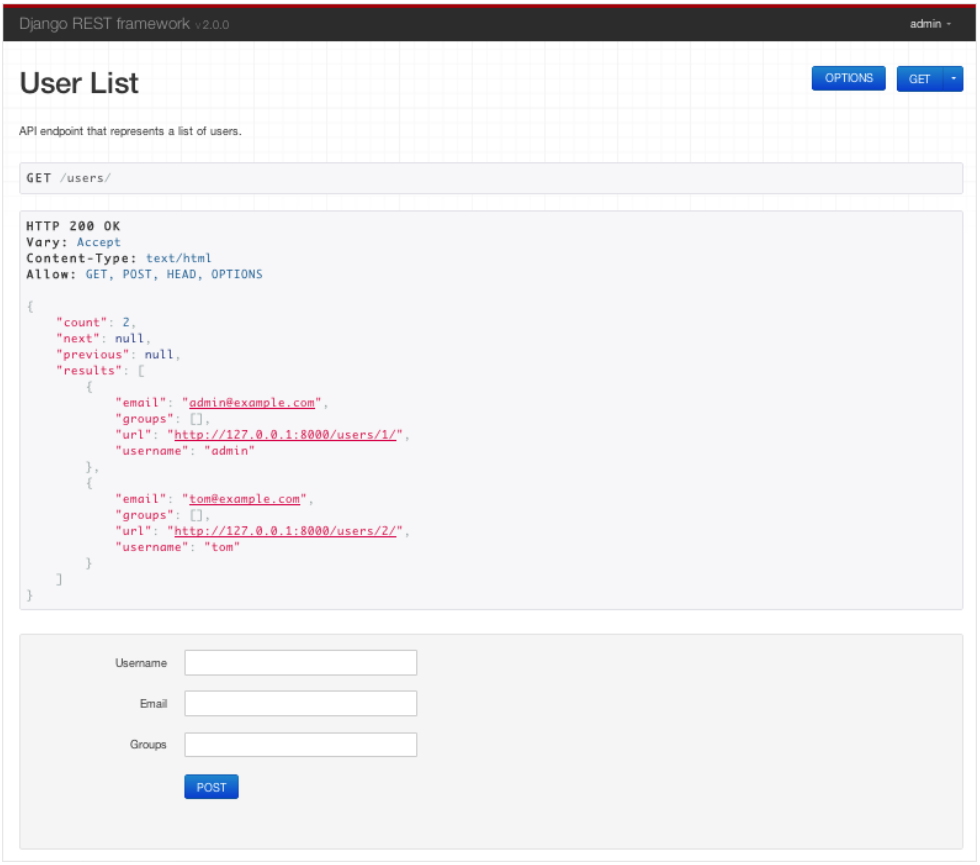

Abbildung 10.1: Die Browsable API

10.5.5 Sicherheit

Durch die Einführung einer netzwerkfähigen Schnittstelle werden neue System-Architekturen möglich (siehe Abschnitt 6.2.2). Die strongTNC Komponente muss nicht mehr auf derselben Plattform wie die übrige strongSwan Infrastruktur betrieben werden.

Für die neugeschaffenen Kommunikationspfade ist Sicherheit nun ein wichtigeres Thema als zuvor, zumal die Kommunikation von nun an auch über ein Netzwerk erfolgen kann. Durch die stärkere Exponierung der Schnittstellen werden diese auch ein potentielles Angriffsziel. Bei der Datenübertragung soll einerseits die Vertraulichkeit und Integrität der Daten gewährleistet sein, andererseits sollte durch entsprechende Authentisierungs- und Authorisierungs-Mechanismen der Zugriff nur für vertrauenswürdige Clients möglich sein.
Die adäquate Verwendung von Transport Layer Security (TLS) würde diese Anforderungen bereits erfüllen. Das grosse Risiko hierbei ist jedoch, dass die Verantwortung der korrekten

Umsetzung aus den Händen der Schnittstelle an den Anwender übergeht. Damit kann die Applikation nicht mehr als grundsätzlich sicher betrachtet werden, da die sachgemässe Konfiguration des Webservers nicht garantiert werden kann[24]. Eine optimale Umsetzung würde daher vorsehen, Sicherheitsaspekte so in die Applikation zu integrieren, dass die Kommunikation ohne entsprechende Sicherheitsmassnahmen nicht möglich ist. Im Folgenden sollen mögliche Authentisierungs- und Authorisierungs-Varianten aufgezeigt und deren Vor- und Nachteile besprochen werden.

HTTP Basic / Digest Diese Lösung ist einfach zu implementieren, bedingt aber den Einsatz von TLS, da sonst Authentisierungsdaten im Klartext übermittelt werden. Das Protokoll sieht keine Integritätsprüfung und Datenverschlüsselung vor.

OAuth 1.0a / OAuth 2.0 OAuth ist sehr komplex, das macht es schwierig, den Standard sicher zu implementieren. Gerade OAuth 2.0 war in der Vergangenheit harscher Kritik ausgesetzt[25, 26]. Zudem bietet OAuth 2.0 nur Authorisierung, keine Authentisierung. OAuth 1.0a wäre eine gangbare Lösung, allerdings werden die meisten der Features von OAuth in der strongTNC API gar nicht benötigt.

HTTP Signatures Dieses Verfahren verwendet zusätzliche HTTP Header, um Signaturen zu übertragen. Die Authentisierung erfolgt anhand einer geleisteten Signatur. Somit kann die Integrität und Authentizität der übertragenen Daten gewährleistet werden. Der IETF Standard [27] befindet sich allerdings noch im Entwurfsstadium.

10.5.5.1 Implementation

Aus zeitlichen Gründen konnte die nahtlose Integration von Sicherheitsfeatures in die Schnittstelle nicht umgesetzt werden. Anstelle wurde Session basierte Authentisierung und HTTP Basic Authentisierung umgesetzt. Wir empfehlen zudem ausdrücklich, TLS zu verwenden. Um die Konfiguration von TLS mit Apache zu erleichtern, haben wir eine ausführliche Deployment Dokumentation verfasst (Abschnitt 15.5). Darin sind unter anderem empfohlene Cipher-Suites, die Aktivierung von Strict-Transport-Security und eine Anleitung zum Erstellen serverseitiger Zertifikate enthalten.

10.5.5.2 Empfehlung

Falls sich der HTTP Signatures Draft zu einem Standard durchsetzt, empfehlen wir dessen Implementation mit zusätzlicher Verwendung von TLS. Obwohl HTTP Signatures ohne TLS verwendet werden können, weist das HTTP Protokoll verschiedene Verletzlichkeiten[28] auf, die durch TLS behoben werden.

11 Rückblick

11.1 Schlussfolgerungen

11.1.1 Erreichte Ziele

Alle in der Aufgabenstellung geforderten Ziele wurden erfüllt. Ebenso sind alle aus der Aufgabenstellung resultierenden Usecases abgedeckt. Darüber hinaus konnten zusätzliche Ideen und Vorschläge eingebracht und umgesetzt werden:

REST API Während der Projektarbeit wurde ein Konzept für eine umfassende REST Schnittstelle für strongTNC ausgearbeitet, mit dem Ziel die Schnittstelle in Form einer gemeinsam genutzten Datenbank abzulösen. Das Konzept konnte auch als Proof of Concept im Bereich der SWID Erweiterung umgesetzt werden.

Code Qualität Durch das Einbringen von Django-Konventionen, besserem Testing, Coding Guidelines und Analyse-Tools konnte die Qualtät des Codes messbar gesteigert werden (Absatz 9.3.1.3).

Zusätzliche Backends für SWID Generator Zusätzlich zur geforderten Unterstützung des unter Debian und Ubuntu genutzten Paketmanagers DPKG, konnte auch die Unterstützung für RPM und Pacman implementiert werden.

Packaging und Deployment für SWID Generator Um das Deployment des SWID Generators zu vereinfachen, haben wir diesen paketiert und im Python Package Index publiziert. Dies ermöglicht eine einfache Installation über ein einziges Kommando (Absatz 8.6).

Rückfluss von Wissen in die Community Durch die intensive Arbeit mit Libraries und Plugins konnten einige Fehler in diesen festgestellt werden. Einige davon konnten behoben werden und wurden als Pull Request in den Entwicklern der Erweiterungen zur Verfügung gestellt.

11.1.2 Learnings

Die intensive Arbeit an einem Projekt fordert, dass bestimmte Technologien im Detail betrachtet werden und dass oft durchgeführte Prozesse optimiert werden. Dabei lernt man Neues und

bestätigt Altes. Im Folgenden sind die wesentlichsten Learnings dieser Arbeit erwähnt.

Git: Branching und Reviews Die Arbeit mit «Feature Branches» und konsequentem Review, bevor ein Branch in den Master-Branch einfliessen darf, hatte viele positive Effekte: Es konnten Fehler gefunden werden, bevor sie in die Produktion gelangten, zudem konnte das Wissen über einzelne Komponenten verteilt werden. Allerdings kostet dieses Vorgehen viel Zeit. Es ist schwierig zu sagen, ob sich der erhöhte Zeitaufwand gegenüber der eingesparten Zeit durch frühzeitiges Finden von Bugs auszahlt. Das Vorgehen hat sich jedoch in unserem Fall bewährt.

SQLite An verschiedenen Punkten dieses Projektes war die Datenbank Bestandteil von Problemstellungen. Nicht selten konnte festgestellt werden, dass die technischen Einschränkungen von SQLite die Verursacher dieser Probleme waren. SQLite eignet sich gut für Prototyping, Embedded-Systeme oder kleine Einzelbenutzer-Systeme, verursacht bei grösseren Infrastrukturen mit Mehrbenutzerzugriff starke Probleme.

12 Ausblick

12.1 Offene Issues

Während der Projektarbeit wurden Bugs, weiterführende Arbeiten und Ideen im Github Issuetracker[1] erfasst, ungeachtet davon, ob sie im Rahmen dieser Arbeit bearbeitet werden konnten oder nicht. Diese Issues können noch bearbeitet werden, eventuell in einer Nachfolgearbeit oder durch die Community.

12.2 Empfehlungen

REST API vollständig umsetzen Die vollständige Trennung von strongTNC und strongSwan sollte auf jeden Fall umgesetzt werden.

Wechsel von SQLite zu MySQL Durch die Anpassungen an strongTNC wurde die Applikation darauf vorbereitet, mit grossen Datenmengen umgehen zu können. Ein Aspekt der dies zur Zeit teilweise noch verhindert, ist die Verwendung von SQLite. Daher empfehlen wir die nötigen Schritte vorzunehmen, um MySQL als Datenbankbackend nutzen zu können. Es wurden teilweise Workarounds implementiert, die lediglich der SQLite Kompatibilität dienen. Diese könnten entfernt werden, sobald die Unterstützung von SQLite eingestellt wird.

Upgrade auf Django 1.7 Da Django 1.7[2] erst im Sommer 2014 veröffentlicht wird, konnten wir ein Upgrade auf diese Version noch nicht durchführen. Django bringt mit der Version 1.7 viele neue Features mit, unter anderem ein eigenes Datenbank-Migrations-Framework. Es wäre sinnvoll, Django auf Version 1.7 zu aktualisieren, sobald die Trennung der Datenbank umgesetzt wurde. Die Datenbank kann dann gleich mit dem Migrations-Framework initialisiert und verwaltet werden.

Class Based Views Die Verwendung von Class Based Views sollte konsequenterweise für alle bestehenden Views eingesetzt werden. Bisher folgen nur die Views der SWID Erweiterungen diesem Konzept.

[1] `https://github.com/tnc-ba/strongTNC/issues`
[2] `https://docs.djangoproject.com/en/dev/releases/1.7/`

Django Forms Framework Die Formularvalidierung wird derzeit noch manuell im Frontend per Javascript und im Backend in den jeweiligen Views gemacht. Django bietet jedoch mit dem «Forms Framework» ein komfortables Hilfsmittel, mit welchem diese aufwändige Arbeit elegant erledigt werden kann. Die Komplexität der Views könnte dadurch erheblich reduziert werden.

API Sicherheit Falls sich der HTTP Signatures Draft zu einem offiziellen Standard entwickeln sollte, empfehlen wir, diesen in der REST API einzusetzen. Damit könnte die Authentisierung und Datenintegrität der Kommunikation besser sichergestellt werden, als dies mit Basic Authentication möglich ist. Zudem sollte auch mit HTTP Signatures stets TLS eingesetzt werden, da das HTTP Protokoll verschiedene Verletzlichkeiten aufweist, die durch TLS behoben werden[28].

Frontend Sicherheit Wir empfehlen auch für das Frontend den universellen Einsatz von TLS. Als Hilfestellung haben wir eine detaillierte Deployment-Dokumentation für die Konfiguration von Apache mit TLS-Verschlüsselung erstellt (Abschnitt 15.5).

13 Verzeichnisse

13.1 Literatur

[1] *ISO CD 19770-2 Draft*. Tech. rep. ISO/IEC JTC 1/SC 7, 2013-12-13.

[2] Stefan Rohner and Marco Tanner. «Cygnet». Bachelor's Thesis. HSR Hochschule für Technik Rapperswil, 2013. URL: http://eprints.hsr.ch/id/eprint/305.

[3] Guido van Rossum, Barry Warsaw, and Nick Coghlan. *Style Guide for Python Code*. 2001. URL: http://legacy.python.org/dev/peps/pep-0008/ (visited on 06/06/2014).

[4] Andrew Hunt and David Thomas. *The Pragmatic Programmer. From Journeyman to Master*. Pearson Education, 1999. ISBN: 9780132119177.

[5] Mark Bartel et al. «XML-signature syntax and processing». In: *W3C recommendation* 12 (2002).

[6] Tim Berners-Lee, Roy Fielding, and Larry Masinter. *Uniform resource identifier (uri): Generic syntax*. RFC 3986. 2005.

[7] *Scrum Artifacts*. Scrum Alliance. 2008. URL: http://www.scrumalliance.org/why-scrum/core-scrum-artifacts-activities (visited on 06/11/2014).

[8] DistroWatch.com. *Top Ten Distributionen. Ein Überblick über die heute führenden Distributionen*. URL: http://distrowatch.com/dwres.php?resource=major (visited on 06/04/2014).

[9] Bill Venners. *Contracts in Python. A Conversation with Guido van Rossum*. 2003. URL: http://www.artima.com/intv/pycontract.html (visited on 06/06/2014).

[10] Marty Alchin. *Pro Python*. Books for professionals by professionals. Apress, 2010. ISBN: 9781430227588.

[11] Scott Meyers. *More Effective C++. 35 New Ways to Improve Your Programs and Designs*. Pearson Education, 1995. ISBN: 9780132797474.

[12] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-oriented Software*. Pearson Education, 1994.

[13] Armin Ronacher and Nick Coghlan. *Explicit Unicode Literal for Python 3.3*. 2012. URL: http://legacy.python.org/dev/peps/pep-0414/ (visited on 06/11/2014).

[14] Lennart Regebro. *Porting to Python 3: An In-Depth Guide*. Createspace Independent Pub, 2013. ISBN: 9781490362229. URL: http://python3porting.com/.

[15] Armin Ronacher. *Porting to Python 3 Redux*. URL: http://lucumr.pocoo.org/2013/5/21/porting-to-python-3-redux/ (visited on 06/11/2014).

[16] Yuval Greenfield. *Python 3 Wall of Superpowers*. URL: http://python3wos.appspot.com/ (visited on 06/11/2014).

[17] Hristo Deshev. *Quick Tips on Making Your Code Python 3 Ready*. 2012. URL: http://stackful-dev.com/quick-tips-on-making-your-code-python-3-ready.html (visited on 06/11/2014).

[18] Craig Larman. *Applying UML and Patterns. An Introduction to Object-oriented Analysis and Design and the Unified Process*. Prentice Hall PTR, 2002. ISBN: 9780130925695.

[19] Markus Schumacher et al. *Security Patterns: Integrating security and systems engineering*. John Wiley & Sons, 2013.

[20] John Nagle. *Congestion Control in IP/TCP Internetworks*. RFC 896. 1984.

[21] Roy Thomas Fielding. «Architectural styles and the design of network-based software architectures». PhD thesis. University of California, 2000.

[22] Daniel Greenfeld. *Choosing an API framework for Django*. 2012. URL: http://pydanny.com/choosing-an-api-framework-for-django.html (visited on 06/12/2014).

[23] Gilad Bracha and William Cook. «Mixin-based inheritance». In: *ACM SIGPLAN Notices*. Vol. 25. 10. ACM. 1990, pp. 303–311.

[24] *Security Misconfiguration*. OWASP Foundation. 2013. URL: https://www.owasp.org/index.php/Top_10_2013-A5-Security_Misconfiguration (visited on 06/12/2014).

[25] Eran Hammer. *OAuth 2.0 and the Road to Hell*. 2012. URL: http://hueniverse.com/2012/07/26/oauth-2-0-and-the-road-to-hell/ (visited on 06/11/2014).

[26] Egor Homakov. *OAuth1, OAuth2, OAuth...?* 2013. URL: http://homakov.blogspot.jp/2013/03/oauth1-oauth2-oauth.html (visited on 06/11/2014).

[27] Internet Engineering Task Force (IETF). *Signing HTTP Messages*. 2014. URL: https://web-payments.org/specs/source/http-signatures (visited on 06/11/2014).

[28] Internet Engineering Task Force (IETF). *Security Considerations for HTTP Signatures*. 2014. URL: https://web-payments.org/specs/source/http-signatures-audit/ (visited on 06/11/2014).

13.2 Abbildungsverzeichnis

13.3 Listingverzeichnis

14 Glossar

DBMS	Datenbankmanagementsystem. Verwaltungssoftware für ein Datenbanksystem.
DTO	Data Transfer Object, eine Klasse, die kein Eigenverhalten aufweist, sondern lediglich Daten in einem Objekt bündelt.
Enforcement	Erzwingt eine Policy auf einer Gruppe von Clients.
IMC	Information Measurement Collector. Sammelt Daten auf dem CLient und sendet diese an den IMV zur Verifizierung.
IMV	Information Measurement Verifier. Empfängt Informationen vom IMC und verifiziert diese. Erzwingt Policies auf Serverseite. Erzwingt Policies auf Serverseite.
Model	Abstraktion eines Datenbankobjektes.
ORM	Object-Relational-Mapping. Layer für die Speicherung von Objekten in einer Relationalen Datenbank.
Package	Ein Software-Paket, das auf einem Client installiert ist. Kann auf eine schwarze Liste gesetzt werden.
Policy	Eine Richtlinie, die ein Client einzuhalten hat, wenn er sich ins VPN einwählen will.
Product	Ein Produkt, bzw. Betriebssystem, das auf einem Client installiert ist.
Regid	Unique registrastion ID. Identifizierer einer Organisation, Format: regid.YYYY-MM.<reverse domain name>. Das Datum ist das Registrierungsdatum der Domain.
REST	Representational State Transfer. Programmierpradigma für die Implementation von zustandslosen Ressourcenorientierten Webschnittstellen.
Software-ID	Eindeutiger Identifier einer Softwarepaketes. Bestehend aus regid des Tag Creators und uniqueID des Softwarepaketes.

SWID Tag	Software Identification Tag. Beschreibung einer Softwarekomponente im XML Format. Standardisiert nach ISO 19770-2.
TCG	Trusted Computing Group. Standardisierungsorganisation, die Standards für das TNC Framework verfasst.
TLS	Transport Layer Security. Hybrides Verschlüsselungsprotokoll zur Sicheren Datenübertragung via TCP.
TNC	Trusted Network Connect. Opensource Architekturframework für die Netzwerkzugangskontrolle.
TNC Client	Gegenstück zum TNC Server. Erlaubt die Verbindungsaufnahme mit einem TNC Server.
TNC Policy Manager	Komponente aus dem TNC Framework, welche die Richtlinien (Policies) verwaltet.
TNC Server	Trusted Network Connect Server. Der Server ist im Falle von strongSwan die Verwaltungsinstanz der IMVs.
uniqueID	Kennung, die eine Software innerhalbt des Namespaces des Tag Creators eindeutig identifiziert.
View	Django Funktion oder Klasse, welche einen Request entgegen nimmt und eine Response zurückliefert.
VPN	Virtual Private Network. Eine verschlüsselte Verbindung, die dem Benutzer Zugang ins lokale Netzwerk einer Firma oder Organisation über einen unsicheren Kanal (z.B. das Internet) ermöglicht.
Workitem	Definiert einen Arbeitsauftrag. Wird von Cygnet für IMVs generiert und von diesen ausgeführt. Die Resultate werden zurück an Cygnet geliefert und dann von Cygnet ausgewertet.

Teil III

Appendix

15 Anhang

15.1 Coding Styleguide

15.1.1 Einleitung

Die Python-Community legte schon von Beginn an viel Wert auf Lesbarkeit und Konsistenz von Source Code. Dazu gehört auch ein einheitlicher Code-Stil. Guido van Rossum, der Autor von Python, schrieb deshalb seine Vorstellungen von sauberem Code in einem *Style Guide for Python Code* nieder. Dieser Style Guide wurde im Jahr 2001 als Python Enhancement Proposal 8 – kurz PEP8 – veröffentlicht[1].

Der PEP8 Style Guide hat seit dann beinahe universelle Verbreitung gefunden. Einer der zentralsten Punkte daraus – die Verwendung von 4 Spaces anstelle von Tabs – wird gemäss einem Analysetool[2] in 95% der Python Projekte auf Github so umgesetzt. Im Rahmen dieser Bachelorarbeit werden wir daher auch gemäss diesen Richtlinien arbeiten, mit einigen kleinen Anpassungen.

15.1.2 Coding Guidelines

PEP8 ist in unserer Software verbindlich, mit folgenden Ausnahmen:

- Maximale Zeilenlänge ist 109 Zeichen, nicht 79. Heutige Bildschirme sind viel grösser, es ergibt keinen Sinn Code umzubrechen, um innerhalb der 80-Zeichen-Grenze zu bleiben, wenn dadurch der Code weniger gut lesbar wird.
- Folgende Einrückungsregeln[3] in den Code-Checking-Tools können in gewissen Fällen zu schlechter lesbarem Code führen und können deshalb ignoriert werden: *E126*, *E127*, *E128*.

[1] `https://python.org/dev/peps/pep-0008/`
[2] `http://sideeffect.kr/popularconvention#python`
[3] `http://pep8.readthedocs.org/en/latest/intro.html#error-codes`

15.1.3 Future Imports

Um in einer Python 2 Codebase möglichst gute Vorwärtskompatibilität mit Python 3 zu erreichen, gibt es Backports von neueren Funktionen nach Python 2.7 – die sogenannten *Future Imports*. Die empfohlenen Imports für strongTNC sind:

```
from __future__ import print_function, division, absolute_import, unicode_literals
```

Die Beweggründe für diese Liste von Imports sind in einem Blogeintrag von Stackful.io[4] sehr gut erläutert.

Da der Wechsel zu diesen Imports in der aktuellen Codebasis potentiell Bugs verursachen kann (gerade beim `unicode_literals` Import), gilt diese Empfehlung nur für neue Module. Der Legacy-Code sollte in einem separaten Refactoring Stück für Stück angepasst werden.

15.1.4 Docstrings

Die meisten Klassen, Methoden und Funktionen sollten mit Docstrings[5] dokumentiert werden (ausser wenn sie komplett trivial sind).

Docstrings sind mit ReStructuredText formatiert und enden grundsätzlich mit einer Leerzeile. Wenn ein Docstring nur einen Absatz enthält, kann diese jedoch weggelassen werden.

Beispiel eines Modul-Docstrings:

```
"""
This module contains DPKG-specific functions for SWID-Tag generation.
"""
```

[4] http://stackful-dev.com/quick-tips-on-making-your-code-python-3-ready.html
[5] http://legacy.python.org/dev/peps/pep-0257/

Beispiel eines Klassen-Docstrings:

```
class Foo(object):
    """
    This class is responsible for foo-ing a bar. The resulting foobar object
    can be used for baz.

    Be careful when doing xyz. The reason for this implementation is Lorem
    Ipsum.

    """
    def __init__(self):
        ...
```

Beispiel eines Funktions- oder Methoden-Docstings:

```
def myfunc(name, state=None):
    """
    This function does something.

    Args:
        name (unicode or str):
            The name to use.
        state (bool):
            Current state to be in.

    Returns:
        Integer return code. Possible return codes::

            0 -- Success!
            1 -- No good.
            2 -- Try again.

    Raises:
        AttributeError:
            Raised if foo happens.
        RuntimeError:
            Raised if bar is too large.

    A really great idea. A way you might use me is

    >>> print myfunc(name='foo', state=None)
    0

    BTW, this always returns 0. **NEVER** use with :class:`MyPublicClass`.

    """
    return 0
```

15.1.5 Tools

15.1.5.1 Flake8

Flake8 (`https://flake8.readthedocs.org/en/2.0/`) verbindet das Style Checking Tool pep8[6] mit dem Static Code Analysis Tool pyflakes[7]. Für unser Projekt kann folgende Konfiguration

[6] `https://pypi.python.org/pypi/pep8`
[7] `https://pypi.python.org/pypi/pyflakes`

(`/.config/flake8`) verwendet werden:

```
[flake8]
ignore = E126,E127,E128
max-line-length = 109
```

15.1.5.2 Pytest

Zum von uns verwendeten Testing-Framework Pytest[8] gibt es ein PEP8 Plugin. Wird dieses aktiviert, werden Style Guide Violations als fehlerhafte Tests gewertet. Folgende Konfiguration wird dafür in `pytest.ini` verwendet:

```
[pytest]
addopts = --pep8
pep8ignore =
    *.py E126 E127 E128
    setup.py ALL
    settings.py ALL
    urls.py ALL
    */migrations/* ALL
    */tests/* ALL
pep8maxlinelength = 109
```

[8] `http://pytest.org/`

15.2 Git Guidelines

Nachfolgend ein paar Regeln zum Umgang mit Git, mit dem Ziel eine möglichst saubere History zu haben.

15.2.1 Commit-Messages

- ...beginnen mit einem Grossbuchstaben
- ...sind in Englisch verfasst
- ...enthalten keine Typos
- ...enthalten keine Smileys
- ...sind kurz und prägnant
- ...enthalten keine relativen Ticket-Referenzen wie `refs #1`, nur absolute Referenzen wie `fixes user/repo#2`. Die Referenzen sollten im Beschreibungstext stehen, nicht in der ersten Zeile.
- ...sind in past tense geschrieben (*fixed bug* statt *fix bug*)

Generell sollte die erste Zeile einer Commit Message in höchstens 72 Zeichen[9] die Änderungen zusammenfassen. Weitere Erläuterungen sollten durch eine Leerzeile getrennt werden.

[9] `http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html`

Beispiel:

```
Capitalized, short (72 chars or less) summary

More detailed explanatory text, if necessary.  Wrap it to about 72
characters or so.  In some contexts, the first line is treated as the
subject of an email and the rest of the text as the body. The blank
line separating the summary from the body is critical (unless you omit
the body entirely); tools like rebase can get confused if you run the
two together.

Further paragraphs come after blank lines.

- Bullet points are okay, too

- Typically a hyphen or asterisk is used for the bullet, followed by a
  single space, with blank lines in between, but conventions vary here

- Use a hanging indent
```

15.2.2 History

15.2.2.1 Rewriting / Reordering / Squashing

Die Git History sollte sauber gehalten werden. Während der Entwicklung ist es kein Problem, wenn man viele und häufige Commits macht, aber vor einem Merge/Rebase in den Hauptcode sollten die Commits sinnvoll reduziert (squashed) werden.

Alles zum Verändern der Git History findet sich hier:
`http://git-scm.com/book/en/Git-Tools-Rewriting-History`

15.2.2.2 Pulling

Beim `git pull` ist es sinnvoll, immer den `--rebase` Parameter zu verwenden. Bei einem Konflikt durch neue Remote Commits wird dann nämlich lokal rebased statt merged. Da der lokale Code noch nicht publiziert wurde, ist dies unbedenklich. Ein Rebase-Konflikt kann bei Problemen jederzeit mit `git rebase --abort` abgebrochen werden.

15.2.2.3 Force Pushing

Während der Entwicklungsphase in einem Branch ist es kein Problem wenn man geänderte History mit `git push --force` pushed, im `master` Branch sollte das jedoch nur in äussersten Ausnahmesituationen geschehen.

15.2.2.4 Merge / Rebase

Ist ein Pull Request abgeschlossen, sollte er vor einem Merge gegen den `master`-Branch rebased werden. Dies verhindert Konflikte und ermöglicht ein *fast-forward merge*, wodurch kein Merge Commit entsteht:

```
git checkout master
git pull --rebase
git checkout <featurebranch>
git rebase master
# fix potential conflicts
# squash commits with git rebase <latest-master-commit>
git push --force origin <featurebranch>
```

Danach sollte der Pull Request sofort gemerged werden:

```
git checkout master
git merge --ff-only <featurebranch>
git push
```

15.3 Workflow

15.3.1 Ablauf

Nachfolgend der Workflow einer Code-Änderung:

1. Ticket wird im Github Issue Tracker erstellt und jemandem zugewiesen.
2. Der zuständige Entwickler erstellt einen Feature Branch und entwickelt darin den benötigten Code.
3. Wenn der entwickelte Code sich im strongTNC Repository befindet, kann der Issue in ein Pull Request umgewandelt werden (15.3.2). Ansonsten einen separaten Pull Request erstellen und darin auf den Issue verweisen (`refs user/repo#issue`).
4. Code wird von jemandem reviewed. Korrekturen werden in den Branch pushed.
5. Commits werden wenn sinnvoll reduziert.
6. Rebase des Branches gegen `master`.
7. Merge in `master` via Kommandozeile.
8. Ticket mit Referenz auf relevante Commits schliessen.

15.3.2 Umwandeln von Issues in Pull Requests

Mit dem Kommandozeilen-Tool hub[10] kann ein Issue in ein Pull Request umgewandelt werden:

```
git checkout <featurebranch>
git push origin <featurebranch>
hub pull-request -b tnc-ba/strongTNC:master -i <issue-number> -h <featurebranch>
```

[10] `https://github.com/github/hub`

15.4 Definition of Done

Ein Task gilt als abgeschlossen, wenn folgende Punkte erfüllt sind:

- Alle Arbeiten gemäss Task-Beschreibung wurden ausgeführt.
- Der Code wurde auf Github in einen Feature Branch committed.
- Der Code wurde von einem Teammitglied reviewed.
- Der Code ist sinnvoll kommentiert, Docstrings für Funktionen und Klassen sind vorhanden.
- Flake8 zeigt keine Fehler oder Warnungen.
- Tests existieren wo sinnvoll. Die Testsuite läuft erfolgreich durch.
- Dokumentation wurde nachgeführt.
- Der Feature Branch wurde in den Master Branch gemerged.
- Die benötigte Arbeitszeit wurde erfasst.

15.5 strongTNC Deployment Manual

Due to security- and performance-considerations, strongTNC should never be deployed into production using the `./manage.py runserver` command. Instead, a proper WSGI Webserver should be used, and all debug settings should be turned off.

Warning: Never deploy strongTNC in production without using SSL/TLS to secure your connections, especially if you use the API.

The following how-to assumes you're using Ubuntu or a similar Linux distribution, but the basic concepts can be applied to any Linux distribution.

(Note: If you just want to try strongTNC, you can also use the automatially configured testing VMs. See `vm/README.rst` for more information.)

Install the base system

Set up your base Ubuntu system. Make sure all the packages are up to date.

Install required packages

Install the dependencies for strongTNC:

```
sudo apt-get install wget build-essential apache2 libapache2-mod-wsgi python2.7 \
    python2.7-dev python-pip python-virtualenv libxml2-dev libxslt1-dev
```

If you want to use MySQL instead of SQLite, you need a few additional dependencies:

```
sudo apt-get install mysql-server mysql-client libmysqlclient-dev
```

Create directories

```
sudo mkdir /var/www/strongTNC /etc/strongTNC
sudo chown $(whoami):www-data /var/www/strongTNC /etc/strongTNC
sudo chmod 775 /var/www/strongTNC /etc/strongTNC
```

Download current strongTNC release

Download the current snapshot of the master branch from Github:

```
cd /tmp
wget https://github.com/strongswan/strongTNC/archive/master.tar.gz
tar xfvz master.tar.gz
mv strongTNC-master/* /var/www/strongTNC/
```

If you want you could also use git to clone the repository.

Install Python dependencies

The recommended way to install the Python dependencies is to put them in a Virtualenv[11].

```
cd /var/www/strongTNC
virtualenv --no-site-packages VIRTUAL
VIRTUAL/bin/pip install -U -r requirements.txt
```

If you use MySQL, you need an additional Python package:

```
VIRTUAL/bin/pip install -U MySQL-python
```

strongTNC configuration

Copy the sample configuration to /etc:

```
cp /var/www/strongTNC/config/settings.sample.ini /etc/strongTNC/settings.ini
cd /etc/strongTNC/
sudo chown $(whoami):www-data settings.ini
sudo chmod 640 settings.ini
```

Now edit the configuration file with your favorite text editor. First of all, update the database configuration. If you want to use SQLite, set them to the following values:

```
DJANGO_DB_URL = sqlite:////var/www/strongTNC/django.db
STRONGTNC_DB_URL = sqlite:////var/www/strongTNC/ipsec.config.db
```

[11] http://virtualenv.readthedocs.org/

For MySQL, use the following values:

```
DJANGO_DB_URL = mysql://strongtnc:<mysql-password>@127.0.0.1/strongtnc_django
STRONGTNC_DB_URL = mysql://strongtnc:<mysql-password>@127.0.0.1/strongtnc_data
```

(Choose a secure password and remember it for later, when we create the MySQL user!)

Now you need to set a value for `SECRET_KEY`, which is used by Django to encrypt all kinds of stuff. A way to generate such a key is the following code snippet:

```
dd if=/dev/urandom bs=128 count=1 2>/dev/null | base64 -w 175
```

Then set `ALLOWED_HOSTS` to a list of hostnames that will be allowed to serve strongTNC, e.g.

```
ALLOWED_HOSTS = 127.0.0.1,strongtnc.example.org
```

If you enable SSL/TLS for your setup (you really should!), enable secure CSRF cookies:

```
CSRF_COOKIE_SECURE = 1
```

You should also take a look at the `[admins]` section and add your name and e-mail address there.

Set up database

SQLite

If you're using SQLite, all you need to do is changing the database permissions:

```
cd /var/www/strongTNC/
sudo chgrp www-data django.db ipsec.config.db
sudo chmod 660 django.db ipsec.config.db
```

MySQL

First, log in to the MySQL console with the root user:

```
mysql -u root -p
```

Create the required databases:

```
mysql> CREATE DATABASE strongtnc_django CHARACTER SET utf8 COLLATE utf8_unicode_ci;
mysql> CREATE DATABASE strongtnc_data CHARACTER SET utf8 COLLATE utf8_unicode_ci;
```

Create a new user (make sure to replace <password> with the previously chosen MySQL password):

```
mysql> GRANT ALL PRIVILEGES ON strongtnc_django.* TO strongtnc@localhost
-> IDENTIFIED BY '<password>';
mysql> GRANT ALL PRIVILEGES ON strongtnc_django.* TO strongtnc@localhost
-> IDENTIFIED BY '<password>';
```

Create the necessary schema in your database:

```
cd /var/www/strongTNC/
VIRTUAL/bin/python manage.py syncdb --database=meta --noinput
VIRTUAL/bin/python manage.py syncdb --database=default --noinput
```

Collect static files

Run the following command to collect all static files in a single directory:

```
cd /var/www/strongTNC/
VIRTUAL/bin/python manage.py collectstatic --noinput
```

Apache configuration

Write the following configuration to `/etc/apache2/sites-available/strongTNC`

```
WSGIPythonPath /var/www/strongTNC:/var/www/strongTNC/VIRTUAL/lib/python2.7/site-packages

NameVirtualHost *:80
<VirtualHost *:80>
    RewriteEngine On
    RewriteCond %{HTTPS} off
    RewriteRule (.*) https://%{HTTP_HOST}%{REQUEST_URI}
</VirtualHost>

<VirtualHost _default_:443>
    # The ServerName directive sets the request scheme, hostname and port that
    # the server uses to identify itself. This is used when creating
    # redirection URLs. In the context of virtual hosts, the ServerName
    # specifies what hostname must appear in the request's Host: header to
    # match this virtual host. For the default virtual host (this file) this
    # value is not decisive as it is used as a last resort host regardless.
    # However, you must set it for any further virtual host explicitly.
    #ServerName strongtnc.example.com

    SSLEngine on
    SSLCertificateFile /etc/apache2/ssl/strongtnc.crt
    SSLCertificateKeyFile /etc/apache2/ssl/strongtnc.key
    SSLProtocol all -SSLv2 -SSLv3
    SSLHonorCipherOrder on
    SSLCompression off
    SSLCipherSuite "EECDH+ECDSA+AESGCM EECDH+aRSA+AESGCM EECDH+ECDSA+SHA384 \
        EECDH+ECDSA+SHA256 EECDH+aRSA+SHA384 EECDH+aRSA+SHA256 EECDH+aRSA+RC4 \
        EECDH EDH+aRSA RC4 !aNULL !eNULL !LOW !3DES !MD5 !EXP !PSK !SRP !DSS"
    Header add Strict-Transport-Security "max-age=15768000"

    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/strongTNC

    <Directory /var/www/strongTNC>
        <Files wsgi.py>
            Order deny,allow
            Allow from all
        </Files>
        Options -Indexes
    </Directory>

    WSGIScriptAlias / /var/www/strongTNC/config/wsgi.py
    Alias /static/ /var/www/strongTNC/static/

    WSGIPassAuthorization On

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

Then disable the default configuration and enable strongTNC:

```
sudo a2dissite 000-default
sudo a2ensite strongTNC
```

Enable necessary plugins and create ssl directory:

```
sudo a2enmod ssl rewrite headers
sudo mkdir /etc/apache2/ssl
```

Copy your TLS certificate and the private key to `/etc/apache2/ssl`. If you want to create self-signed certificates, execute the following command:

```
sudo openssl req -x509 -nodes -sha256 -days 365 -newkey rsa:3072 -utf8 \
    -keyout /etc/apache2/ssl/strongtnc.key -out /etc/apache2/ssl/strongtnc.crt
```

Make sure the permissions are restrictive:

```
sudo chown root:root /etc/apache2/ssl/*
sudo chmod 400 /etc/apache2/ssl/*
```

Now restart Apache and strongTNC should be up and running!

```
sudo service apache2 restart
```

Create default users

In order to be able to login into strongTNC, you need to set a password for a readonly user and an admin user.

```
cd /var/www/strongTNC/
VIRTUAL/bin/python manage.py setpassword
```

Visit `http://yourserver/` to log in.

15.6 strongTNC REST API Konzept

strongTNC REST API

Im Rahmen der strongTNC BA

Danilo Bargen, Christian Fässler, Jonas Furrer

Frühlingssemester 2014

Inhaltsverzeichnis

1 Einleitung

Die Definition der Ressourcen orientiert sich an den Regeln des Buches *REST API Design Rulebook*[1] aus dem O'Reilly Verlag.

URI Definition

Bei der Bezeichnung der URIs[1] wurde folgende Terminologie gemäss RFC 3986[2] verwendet:

```
URI = scheme "://" authority "/" path [ "?" query ] [ "#" fragment ]
```

Ressource-Archetypen

Nachfolgend die Ressource-Archetypen gemäss Masse 2011[1]. Die Erklärungstexte wurden direkt dem besagten Buch entnommen.

Document A document resource is a singular concept that is akin to an object instance or database record. A document's state representation typically includes both fields with values and links to other related resources.

Collection A collection resource is a server-managed directory of resources. Clients may propose new resources to be added to a collection. However, it is up to the collection to choose to create a new resource, or not.

Store A store is a client-managed resource repository. A store resource lets an API client put resources in, get them back out, and decide when to delete them. On their own, stores do not create new resources; therefore a store never generates new URIs. Instead, each stored resource has a URI that was chosen by a client when it was initially put into the store.

Controller A controller resource models a procedural concept. Controller resources are like executable functions, with parameters and return values; inputs and outputs. Like a traditional web application's use of HTML forms, a REST API relies on controller resources to perform application-specific actions that cannot be logically mapped to one of the standard methods (create, retrieve, update, and delete, also known as CRUD).

[1] Uniform Resource Identifier

2 Repräsentation der Archetypen

Die JSON-Repräsentation ist abhängig vom Ressource-Archetypen.

Document Ein Document gibt ein JSON Objekt zurück, welches alle relevanten Felder enthält. Informationen, welche durch die Ressource-URI bereits gegeben sind (z.B. Type), müssen nicht, können jedoch, erneut in der Liste erscheinen.
Falls im Document Unterobjekte auftauchen, wird für das Feld die URI des betreffenden Unterobjekts eingesetzt, damit kann dieses Objekt direkt abgefragt werden. Zusätzlich wird ein Query-Parameter bereit gestellt, welcher es erlaubt, Unterobjekte bis zu einer gewissen Tiefe aufzulösen.
Beispiel:

```
{
  "field1": "<str,value-1>",
  "field2": <int,value-2>,
  "field3": <bool,value-3>,
  "subObject": "<uri,sub-object>"
  "subCollection": [
    {
      "uri": "<uri,sub-object>"
    },
    {
      "uri": "<uri,sub-object>"
    }
  ]
}
```

Beispiel mit Query `depth=1`:

```
{
  "field1": "<str,value-1>",
  "field2": <int,value-2>,
  "field3": <bool,value-3>,
  "subObject": {
    "field1": <int,value-1>,
    ...
    "uri": "<uri,sub-object>"
  }
  "subCollection": [
    {
      "field": <int, field>,
      ...
      "uri": "<uri,sub-object>"
    },
    {
      "field": <int, field>,
      ...
      "uri": "<uri, sub-object>"
    }
  ]
}
```

Collection Eine Collection gibt eine Liste mit allen enthaltenen (ggf. gefilterten) JSON-Objekten zurück. Die Objekte werden zusätzlich um ein Feld `uri` ergänzt, welches die Ressource-

URI des jeweiligen Objektes enthält. Wie bei einem Document gibt es bei Collections einen optionalen Query Parameter welcher es erlaubt Unterobjekte aufzulösen,
Beispiel:

```
{
  {
    "field1": "<str,value-1>",
    "subObject": "<uri,sub-object>",
    "uri": "<uri,objek>"
  },
  {
    "field1": "'<str,value-1>",
    "subObject": "<uri,sub-object>",
    "uri": "<uri,resource>"
  },
}
```

Beispiel mit Query `depth=1`:

```
{
  {
    "field1": "<str,value-1>",
    "subObject": {
      "field": "<str,value>",
    },
    "uri": "<uri,resource>"
  },
  {
    "field1": "'<str,value-1>",
    "subObject": {
      "field": "<str,value>",
    },
    "uri": "<uri,resource>"
  },
}
```

Store Ein Store verhält sich wie eine Collection, wenn sie direkt angesprochen wird (`/store`) und wie ein Document, wenn ein spezifisches Element des Store angesprochen wird (`/store/{element-id}`).

Controller Der Output des Controllers ist abhängig vom Verwendungszweck.

3 HTTP Statuscodes

Das Ergebnis einer Anfrage wird grundsätzlich über einen HTTP Statuscode mitgeteilt, in gewissen Fällen kann auch noch Payload im Body geliefert werden. Folgendes ist eine Auswahl von Statuscodes die zu erwarten sind:

200 OK Request erfolgreich
201 Created Entity wurde erfolgreich erstellt
204 No Content Entity wurde erfolgreich geändert
400 Bad Request Generischer Client Fehler
404 Not Found Entity nicht gefunden
405 Method Not Allowed Die verwendete HTTP Methode ist auf dieser Ressource nicht erlaubt
409 Conflict Zu erstellende Entity existiert bereits
412 Precondition Failed Es sind zusätzliche Schritte nötig um die Anfrage auszuführen
500 Internal Server Error Generischer Server Fehler

Falls zusätzliche Codes verwendet werden oder die Codes eine aussergewöhnliche oder erwähnenswerte Bedeutung haben, ist dies bei der betroffenen Resource vermerkt.

4 Allgemeine Hinweise

4.1 Schreibweise

Für die Definition des Response- und Request-Formats wird eine pseudo JSON Schreibweise verwendet. Die Werte werden als Tupel in spitzen Klammern dargestellt. Der erste Teil des Tupels zeigt den Typ, der zweite Teil ist eine Beschreibung für den Wert. Folgende Typen werden verwendet:

int Integer Zahlenwert
num Dezimalzahl
str String
bool Wahrheitswert
xml Ein XML Dokument
hex Ein HEX String
uri Die vollqualifizerte URI einer Resource
doc Das Dokument einer Resource, wie dieses aussieht kann im jeweiligen Abschnitt zur genannten Resource nachgeschlagen werden

4.2 Standard Verhalten

Folgende Punkte beschreiben das Standardverhalten der Resourcen:

Depth Query-Parameter Jede Resource die als Antwort eine URI auf eine andere Resource zurück gibt unterstützt den `depth` Query-Parameter. Dieser löst die URIs bis zur gewünschten tiefe auf. Möglicherweise wird die maximale Tiefe eingeschränkt. Wenn keine Tiefe angegeben wird, gilt die Auflösungstiefe 0.

Filter Query-Parameter Filter Query-Parameter sind, falls vorhanden, immer optional, sie dienen der Präzisierung des Rückgabesets. Wenn sie weggelassen werden, wird das vollständige Set zurückgegeben.

Generische Filter Collections unterstützen wenn nicht anders erwähnt generische Filter auf alle Feldern mit Ausnahme der Foreign Keys. Das heisst, es kann grundsätzlich nach jedem Attribut des Rückgabedokuments gefiltert werden. Der Parameter ist wie folgt aufgebaut: `attributeName=query`. Filter können auch auf Document-Resourcen genutzt werden, es wird bei nicht passenden Filterwerten ein `HTTP 404` Status Code zurückgegeben.

Batch Create Collections unterstützen, sofern sie nicht readonly sind oder es anders erwähnt ist, das sogenannte 'Batch Create'. D.h zum Erstellen von neuen Dokumenten kann einer Collection auch eine Liste gesendet werden. Wird 'Batch Create' verwendet, besteht die Antwort des Servers nur aus einem Statuscode, es wird nicht eine Liste der erstellten Dokumente zurück geschickt. Fall die Erstellung fehlschlägt wird kein Dokument angelegt. Es wird versucht mitzuteilen, warum die Erstellung fehlschlug.

5 Daten Definition

5.1 Policy Arguments

Folgende Workitem, bzw. Policy Typen sind momentan vorhanden:

`00: RESVD` Deny
`01: PCKGS` Installed Packages
`02: UNSRC` Unknown Source
`03: FWDEN` Forwarding Enabled
`04: PWDEN` Default Password Enabled
`05: FREFM` File Reference Measurement
`06: FMEAS` File Measurement
`07: FMETA` File Metadata
`08: DREFM` Directory Reference Measurement
`09: DMEAS` Directory Measurement
`10: DMETA` Directory Metadata
`11: TCPOP` Open TCP Listening Ports
`12: TCPBL` Blocked TCP Listening Ports
`13: UDPOP` Open UDP Listening Ports
`14: UDPBL` Blocked UDP Listening Ports
`15: SWIDT` SWID Tag Inventory
`16: TPMRA` TPM Remote Attestation

Einige Typen benötigen unterschiedliche Argumente, Gemeinsamkeiten lassen sich wie folgt gruppieren:

Keine Argumente 00, 01, 02, 03, 04
Datei Pfad 05, 06, 07
Verzeichnis Pfad 08, 09, 10
Port Liste 11, 12, 13, 14
SWID Request Flags 15
TPM Attestation Flags 16

Dementsprechend sollen die Argumente der Workitems übermittelt werden. Folgende Objekte werden je nach Type übermittelt, als Verweis wird `<doc,policy-argument>` verwendet.

Keine Argumente

```
{}
```

Datei Pfad

```
{
  "file": "<str,file-path>"
}
```

Verzeichnis Pfad

```
{
  "directory": "<str,directory-path>"
}
```

Port Liste

```
{
  "portList": [
    "<str,port or port-range>"
  ]
}
```

SWID Request Flags

```
{
  "swidFlags": [
    "<str,swid-flag>"
  ]
}
```

TPM Attestation Flags

```
{
  "tpmFlags": [
    "<str,tpm-flag>"
  ]
}
```

5.2 Recommendation Types

Folgende Recommendation Types bzw. Actions sind vorhanden:

`0: ALLOW` Die Empfehlung/Aktion ist, den Client zuzulassen
`1: BLOCK` Die Empfehlung/Aktion ist, den Client zu blockieren
`2: ISOLATE` Die Empfehlung/Aktion ist, den Client in einem isoliertem Segment zu platzieren
`3: NONE`

Diese Werte werden von der API als Integer Id verwendet. Es wird kein Objekt für die Repräsentation erstellt.

5.3 Hash-Set

Ein Hash-Set wird innerhalb eines File-Documents ausgeliefert, der Verweis lautet `<doc,hash-set>` und ist wie folgt aufgebaut:

```
[
  {
    "algorithm": "<str,algorithm-name>",
    "hash": "<hex,hash> ",
    "product": "<uri,product>"
  }
]
```

Falls ein Algorithmus noch nicht existiert, wird er erfasst, momentan sind folgende Algorithmen erfasst: `SHA384, SHA256, SHA1, SHA1-IMA`

6 REST Ressourcen

6.1 Session Steuerung und Ablauf

6.1.1 Controller

URI Path `/sessionss/start/`
Archetype Controller
Methods POST
Request Parameter

`connectionId` strongSwan Connection Id
`clientIdentity` strongSwan Client-Identity
`hardwareId` Die ID, welche das Gerät identifiziert, so zum Beispiel AIK, Android-ID, DBUS Machine-ID, o.ä. Dies entspricht dem `value` Feld in der `device` Tabelle in der Datenbank
`productName` Der Productname ist der Name des OS wie er in der `product` Tabelle der Datenbank steht

JSON Format Response

```
{
  "sessionId": <int,id>,
  "workitems": [
     <doc,workitem>,
     ...
  ],
  "uri": "<uri,session>"
}
```

Beschreibung Dieser Controller erstellt und startet eine Session, das Device, welches der Session zugeordnet werden soll, wird anhand der `hardwareId` und dem `productName` bestimmt. Falls eines der Objekte noch nicht existiert wird dieses durch den Controller erstellt. Die ID, die im Response Dokument zurück geliefert wird, dient zur zukünftigen Identifikation der soeben gestarteten Session. Ausserdem erhält man eine Liste von Workitems, die für diese Session abgearbeitet werden müssen.

URI Path `/sessions/{id}/end/`
Archetype Controller
Methods POST
Request Parameter

`recommendation` Endgültiges Resultat/Empfehlung für diese Session.

Beschreibung Dieser Controller schliesst die Session ab und startet alle zusätzlich nötigen Vorgänge, so werden zum Beispiel die Workitems dieser Session abgeräumt und die jeweiligen Resultate gespeichert und können via `/sessions/{id}/results` abgerufen werden.

6.1.2 Documents und Collections

URI Path `/sessions/{id}/`
Archetype Readonly Document
Methods GET
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "time": <int,time>,
  "identity": "<uri,identity>",
  "connectionId": <int,connection-id>,
  "device": "<uri,device>",
  "recommendation": <int,rec>
}
```

Beschreibung Informationen zu einer bestimmten Session. Sessions sollen nicht direkt geändert werden, sondern nur über die entsprechenden Controller, der Grund dafür ist, dass im Hintergrund noch zusätzliche Operationen vorgenommen werden müssen.

URI Path `/sessions/`
Archetype Readonly Collection
Filter Query
timeFrom `<int,timestamp>`
timeTo `<int,timestamp>`
Methods GET
JSON Format Response

```
[<doc,session>, ...]
```

Beschreibung Liste aller Sessions. Neue Sessions können nur über den entsprechenden Controller erstellt werden.

Workitems

URI Path `/sessions/{id}/workitems/{id}/`
Archetype Readonly Document
Methods GET
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "session": "<uri,session>",
  "type": <int,type>,
  "argument": <policy-argument>
}
```

Beschreibung Ein zu einer bestimmten Session zugehöriges Workitem. Workitems können nicht direkt erstellt werden, sondern werden beim Erstellen einer Session anhand der Enforcements erstellt. Workitems gibt es nur, so lange eine Session nicht beendet wurde. Nach dem Ende der Session wird ein `HTTP 404` Statuscode zurückgeliefert.

URI Path `/sessions/{id}/workitems/`
Archetype Readonly Collection
Filter Query
type `<int,policy-type>`
Methods GET
JSON Format Response

```
[<doc,workitem>, ...]
```

Beschreibung Eine Liste aller Workitems, die zu einer bestimmten Session gehören. Workitems können nicht direkt erstellt werden, sondern werden beim Erstellen einer Session anhand der Enforcements erstellt. Workitems gibt es nur, so lange eine Session nicht beendet wurde. Nach dem Ende der Session wird eine leere Liste geliefert.

URI Path `/sessions/{id}/workitems/{id}/result/`
Archetype Document
Request Parameter

`recommendation` Resultat/Empfehlung für dieses Workitem.

`comment` Kommentar zum Resultat.

Methods GET, POST
Response Statuscodes

201 Created Resultat wurde erfolgreich gespeichert.

409 Conflict Resultat existiert bereits.

JSON Format Response

```
{
  "recommendation": <int,type>,
  "comment": <str,comment>
}
```

Beschreibung Auf dieser Ressource soll das Resultat des jeweiligen Workitems eingetragen werden. Diese Resultate werden beim Beenden der Session von den Workitems in die Session-Resulate übertragen.

Results

URI Path `/sessions/{id}/results/{id}/`
Archetype Readonly Collection
Methods GET
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "enforcement": "<uri,enforcement>",
  "recommendation": <int,type>,
  "comment": "<str,comment>"
}
```

Beschreibung Einzelnes Resultat. Im Gegensatz zur Resultat-Resource unter der Workitem Resource enthält dieses Document auch einen Link zum zugehörigen Enforcement.

URI Path `/sessions/{id}/results/`
Archetype Readonly Collection
Methods GET
JSON Format Response

```
[<doc,result>, ...]
```

Beschreibung Nach dem Beenden einer Session können die eingetragenen Resultate in dieser Collection abgefragt werden. Diese Collection ist readonly, damit die Session-Ergebnisse nicht nachträglich geändert werden können.

6.2 SWID Erweiterung

6.2.1 SWID Tags: Messung

URI Path `/sessions/{id}/swid-measurement/`
Archetype Controller
Methods POST
Content-Type `application/json; charset=utf-8`
Request Parameter
`softwareId` Software-IDs als JSON-Liste.
Response Statuscodes
200 OK SWID Tags der übermittelten Software-IDs sind eingetragen und wurden für die übermittelte Session eingetragen.
404 Not Found Session mit der spezifizierten ID wurde nicht gefunden.
412 Precondition Failed Es existieren nicht alle SWID Tags für die übertragenen Software-IDs. Als Payload werden die fehlenden Software-IDs übertragen.
JSON Format Response

```
["<str,software-id>", ...]
```

Beschreibung Die strongSwan Komponente sendet eine Liste von Software-IDs an die strongTNC Schnittstelle. Die Software-IDs wurden zuvor auf dem Client gemessen und widerspiegeln die momentan installierten Software Pakete.
Wenn bereits SWID Tags für alle übertragenen Software-IDs bestehen, können diese direkt eingetragen werden.

6.2.2 SWID Tags: Erstellung

URI Path `/swid/add-tags/`
Archetype Controller
Methods POST
Content-Type `application/json; charset=utf-8`
Request Parameter
`xmlData` SWID Tag als JSON-Liste.
Response Statuscodes
200 OK SWID Tags wurden erfolgreich verarbeitet.
400 Bad Request Fehlerhafter Request. Details zum Fehler werden im Response-Body zurückgesendet.
Beschreibung Die übermittelten Tags werden gelesen und in die Datenbank gespeichert. Bereits vorhandene Tags werden übersprungen.

6.3 CRUD Resourcen

6.3.1 Products

URI Path `/products/{id}/`
Archetype Document
Methods GET, PUT, PATCH
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "name": "<str,productname>"
}
```

URI Path `/products/`
Archetype Collection
Methods GET, POST
JSON Format Response

```
[
  {
    "id": <int,id>,
    "uri": "<uri,resource>",
    "name": "<str,productname>"
  }
]
```

URI Path `/products/{id}/default-groups/`
Archetype Collection
Methods GET, POST
Request Parameter

`groupId` group-id

JSON Format Response

```
[<doc,group>, ...]
```

6.3.2 Packages

URI Path /packages/{id}/
Archetype Document
Methods GET, PUT, PATCH
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "name": "<str,packagename>"
}
```

URI Path /packages/
Archetype Collection
Methods GET, POST
JSON Format Response

```
[
  {
    "id": <int,id>,
    "uri": "<uri,resource>",
    "name": "<str,packagename>",
  }
]
```

URI Path /packages/{id}/versions/
Archetype Collection
Methods GET, POST
JSON Format Response

```
[<doc,version>, ...]
```

6.3.3 Versions

URI Path `/versions/{id}/`
Archetype Document
Methods GET, PUT, PATCH, DELETE
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "package": "<uri,package>",
  "product": "<uri,product>",
  "release": "<str,release>",
  "securtiy": <int,security>,
  "blacklist": <bool,blacklist>,
  "time": <int,time>
}
```

URI Path `/versions/`
Archetype Collection
Filter Query
productName `<str,product-name>`
packageName `<str,package-name>`
Methods GET, POST
JSON Format Response

```
[
  {
    "id": <int,id>,
    "uri": "<uri,resource>",
    "package": "<uri,package>",
    "product": "<uri,product>",
    "release": "<str,release>",
    "securtiy": <int,security>,
    "blacklist": <bool,blacklist>,
    "time": <int,time>
  }
]
```

6.3.4 Identities

URI Path `/identities/{id}/`
Archetype Document
Methods GET, PUT, PATCH
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "type": <int,type>,
  "value": "<str,value>"
}
```

URI Path `/identities/`
Archetype Collection
Methods GET, POST
JSON Format Response

```
[
  {
    "id": <int,id>,
    "uri": "<uri,resource>",
    "type": <int,type>,
    "value": "<str,value>"
  }
]
```

6.3.5 Devices

URI Path `/devices/{id}/`
Archetype Document
Methods GET, PUT, PATCH
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "description": "<str,description>",
  "value": "<str,value>",
  "product": "<uri,product>",
  "created": <int,created>
}
```

URI Path `/devices/`
Archetype Collection
Methods GET, POST
JSON Format Response

```
[
  {
    "id": <int,id>,\\
    "uri": "<uri,resource>",
    "description": "<str,description>",
    "value": "<str,value>",
    "product": "<uri,product>",
    "created": <int,created>,
  }
]
```

URI Path `/device/{id}/sessions/`
Archetype Readonly Collection
Filter Query
timeFrom `<int,timestamp>`
timeTo `<int,timestamp>`
Methods GET
JSON Format Response

```
[<doc,session>, ...]
```

URI Path `/device/{id}/sessions/{id}/results/`
Archetype Readonly Collection
Methods GET
JSON Format Response

```
[<doc,result>, ...]
```

URI Path `/device/{id}/groups/`
Archetype Collection
Methods GET, POST
JSON Format Response

```
[<doc,group>, ...]
```

URI Path `/device/{id}/swid-tags/`
Archetype Readonly Collection
Methods POST
JSON Format Response

```
[<doc,swid-tag>, ...]
```

6.3.6 Enforcements

URI Path `/enforcements/{id}/`
Archetype Readonly Document
Methods GET
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "policy": "<uri,policy>",
  "group": "<uri,group>",
  "failRecommendation": <int,rec_fail>,
  "noresultRecommendation": <int,rec_noresult>",
  "maxAge": <int,max_age>
}
```

URI Path `/enforcements/`
Archetype Readonly Collection
Methods GET
Filter Query

groupName `<str,group-name>`
policyName `<str,policy-name>`

JSON Format

```
[
  {
    "id": <int,id>,
    "uri": "<uri,resource>",
    "policy": "<uri,policy>",
    "group": "<uri,group>",
    "failRecommendation": <int,rec_fail>,
    "noresultRecommendation": <int,rec_noresult>",
    "maxAge": <int,max_age>
  }
]
```

URI Path `/enforcements/{id}/groups/`
Archetype Readonly Collection
Methods GET
JSON Format

```
[<doc,group>, ...]
```

6.3.7 Policies

URI Path `/policies/{id}/`
Archetype Readonly Document
Methods GET
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "type": <int,type>,
  "name": "<str,name>",
  "argument": <policy-argument>,
  "failRecommendation": <int,rec_fail>,
  "noresultRecommendation": <int,rec_noresult>",
}
```

URI Path `/policies/`
Archetype Readonly Collection
Methods GET
JSON Format

```
[
  {
    "id": <int,id>,
    "uri": "<uri,resource>",
    "type": <int,type>,
    "name": "<str,name>",
    "argument": <policy-argument>,
    "failRecommendation": <int,rec_fail>,
    "noresultRecommendation": <int,rec_noresult>"
  }
]
```

URI Path `/policies/{id}/enforcements/`
Archetype Readonly Collection
Methods GET
JSON Format

```
[<doc,enforcement>, ...]
```

6.3.8 Groups

URI Path `/groups/{id}/`
Archetype Readonly Document
Methods GET
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "name": "<str,name>",
  "parent": "<uri,group>"
}
```

URI Path `/groups/`
Archetype Readonly Collection
Methods GET
JSON Format

```
[
  {
    "id": <int,id>,
    "uri": "<uri,resource>",
    "name": "<str,name>",
    "parent": "<uri,group>"
  }
]
```

URI Path `/groups/{id}/devices/`
Archetype Collection
Methods GET, POST
JSON Format Response

```
[<doc,device>, ...]
```

URI Path `/groups/{id}/enforcements/`
Archetype Readonly Collection
Methods GET
JSON Format Response

```
[<doc,enfocement>, ...]
```

6.3.9 Files

URI Path `/files/{id}/`
Archetype Document
Methods GET, PUT, PATCH
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "name": "<str,name>",
  "hashes: <doc,hash-set>,
  "dir": "<uri,directory>"
}
```

URI Path `/files/`
Archetype Collection
Methods GET, POST
JSON Format Response

```
[
  {
    "id": <int,id>,
    "uri": "<uri,resource>",
    "name": "<str,name>"
    "hashes: <doc,hash-set>,
    "dir": "<uri,directory>"
  }
]
```

6.3.10 Directories

URI Path `/directories/{id}/`
Archetype Document
Methods GET, PUT, PATCH
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "path": "<str,path>"
}
```

URI Path `/directories/`
Archetype Collection
Methods GET, POST
JSON Format Response

```
[
  {
    "id": <int,id>,
    "uri": "<uri,resource>",
    "path": "<str,path>"
  }
]
```

URI Path `/directories/{id}/files/`
Archetype Collection
Methods GET, POST
JSON Format Response

```
[<doc,file>, ...]
```

6.3.11 SWID Tags

URI Path `/swid-tags/`
Archetype Readonly Collection
Methods GET
JSON Format Response

```
[<doc,swid-tag>, ...]
```

URI Path `/swid-tags/{id}/`
Archetype Readonly Document
Methods GET
Filter Query

packageName `<str,package-name>`
version `<str,version>`
uniqueId `<str,unique-id>`

JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "packageName": "<str,package-name>",
  "version": "<str,version>",
  "uniqueId": "<str,unique-id>",
  "entities": [
    {
    "entity": "<uri,entity>",
    "role":  <int,role>
    }
  ],
  "tagXml": "<xml,swid-tag>"
}
```

URI Path `/swid-tags/{id}/files/`
Archetype Readonly Collection
Methods GET
JSON Format Response

```
[<doc,file>, ...]
```

6.3.12 Entities

URI Path `/swid-entities/{id}/`
Archetype Readonly Document
Methods GET
JSON Format Response

```
{
  "id": <int,id>,
  "uri": "<uri,resource>",
  "name": "<str,name>",
  "regid": "<str,regid>"
}
```

URI Path `/swid-entities/`
Archetype Readonly Collection
Methods GET
JSON Format Response

```
[
  {
    "id": <int,id>,
    "uri": "<uri,resource>",
    "name": "<str,name>",
    "regid": "<str,regid>",
  }
]
```

URI Path `/swid-entities/{id}/swid-tags/`
Archetype Readonly Collection
Methods GET
JSON Format Response

```
[<doc,swid-tag>, ...]
```

Literatur

[1] Masse, Mark. *REST API design rulebook.* O'Reilly Media, Inc., 2011.

[2] Berners-Lee, Tim, Roy Fielding, and Larry Masinter. *RFC 3986: Uniform resource identifier (uri): Generic syntax.* The Internet Society (2005).

15.7 SWID Measurement Endpoint REST Manual

strongTNC REST API

SWID Endpoints

Im Rahmen der strongTNC BA

Danilo Bargen, Christian Fässler, Jonas Furrer

Frühlingssemester 2014

Inhaltsverzeichnis

1 Ablauf der Messung

Eine Messung läuft wie folgt ab:

1. Auf dem Client werden die Software-IDs aller installierter Pakete generiert.
2. Die Software-IDs werden als JSON-Liste an den entsprechenden API-Endpoint [4.1] gesendet.
   - Wenn von der API "`200 OK`" zurückgegeben wurde, ist alles OK und die SWID Tags wurden mit der aktuellen Session verknüpft. ENDE.
   - Wenn von der API "`404 Not Found`" zurückgegeben wurde, ist die Session ID ungültig. Problem beheben und weiter zu Punkt 2.
   - Wenn von der API "`412 Precondition Failed`" zurückgegeben wurde, sind noch nicht alle SWID Tags in der Datenbank erfasst. Weiter zu Punkt 3.
3. Die fehlenden SWID Tags werden im Response-Body als JSON-Liste zurückgegeben. Diese werden nun über Targeted Requests generiert.
4. Die neu generierten Tags werden als JSON-Listen an den entsprechenden API-Endpoint [4.2] gesendet. Es können auch mehrere Tags in einem Request mitgesendet werden, empfehlenswert ist allerdings ein Batching in mehrere Gruppen, da die Requests sonst unter Umständen sehr lange dauern können.
   - Wenn von der API "`200 OK`" zurückgegeben wurde, ist alles OK. Weiter zu Punkt 5.
   - Wenn von der API "`400 Bad Request`" zurückgegeben wurde, stimmt etwas mit dem Request nicht. Fehler-Details werden im Response Body zurückgesendet.
5. Nachdem alle fehlenden SWID Tags eingetragen wurden, kann der Measurement-Request erneut gesendet werden. Weiter zu Punkt 1.

2 Beispiel Szenario

Im Folgenden soll der Ablauf einer Messung anhand eines konkreten Beispiels illustriert werden. In diesem Beispiel soll eine Messung für die drei Tags mit den folgenden Software-IDs erfolgen:

- `regid.2004-03.org.strongswan_Ubuntu_12.04-i686-logrotate-3.7.8-6ubuntu5`
- `regid.2004-03.org.strongswan_Ubuntu_12.04-i686-lsb-base-4.0-0ubuntu20`
- `regid.2004-03.org.strongswan_Ubuntu_12.04-i686-strongswan-4.5.2-1.5+deb7u3`

2.1 Request - SWID Measurement

2.1.1 Request

Zuerst wird versucht eine Messung für die Session mit der ID 2 zu erfassen.

```
POST /api/sessions/2/swid-measurement/ HTTP/1.1
Authorization: Basic cm9vdDpyb290
Host: tncserver:8000
Accept: application/json
Content-Type: application/json; charset=utf-8
Content-Length: 232

[
"regid.2004-03.org.strongswan_Ubuntu_12.04-i686-logrotate-3.7.8-6ubuntu5",
"regid.2004-03.org.strongswan_Ubuntu_12.04-i686-lsb-base-4.0-0ubuntu20",
"regid.2004-03.org.strongswan_Ubuntu_12.04-i686-strongswan-4.5.2-1.5+deb7u3"
]
```

Listing 1: Durchführen einer SWID Messung

Curl Befehl:

```
curl -i -X POST http://tncserver:8000/api/sessions/2/swid-measurement/ \
         -u username:password \
         -H "Accept: application/json" \
         -H "Content-Type: application/json; charset=utf-8" \
         -d "{ \"data\" : $DATA }"
```

Listing 2: Durchführen einer SWID Messung mit cURL

2.1.2 Antwort - Status 412 Precondition Failed

Sind noch nicht alle Tags in der strongTNC Datenbank vorhanden, werden die Software-IDs der fehlenden Tags zurückgeliefert. Der HTTP Status Code ist "`412 Precondition Failed`". Es werden zu diesem Zeitpunkt noch keine Tags mit der Session verknüpft. Eine entsprechende Antwort kann wie folgt aussehen:

```
HTTP/1.0 412 PRECONDITION FAILED
Date: Wed, 14 May 2014 15:21:45 GMT
Server: WSGIServer/0.1 Python/2.7
Vary: Accept, Cookie
Content-Type: application/json
Allow: POST, OPTIONS

["regid.2004-03.org.strongswan_Ubuntu_12.04-i686-strongswan-4.5.2-1.5+deb7u3"]
```

Listing 3: Antwort einer nicht erfolgreichen SWID Messung

2.1.3 Antwort - Status 200 OK

Wären für alle Software-IDs bereits Tags in der Datenbank vorhanden, könnte die Antwort wie folgt aussehen:

```
HTTP/1.0 200 OK
Date: Wed, 14 May 2014 15:21:45 GMT
Server: WSGIServer/0.1 Python/2.7
Vary: Accept, Cookie
Content-Type: application/json
Allow: POST, OPTIONS

[]
```

Listing 4: Antwort einer erfolgreichen SWID Messung

2.2 Nacherfassen von SWID Tags

2.2.1 Request

War die Messung nicht erfolgreich (HTTP 412), müssen für die zurückgelieferten Software-IDs zuerst komplette SWID Tags erfasst werden.

Im SWID Generator kann ein solcher wie folgt abgefragt werden:

```
swid-generator swid \
  --software-id="regid.2004-03.org.strongswan\_Ubuntu\_12.04-i686-fortune-mod-1:1.99.1-4"
```

Ein solcher SWID Tag sieht wie folgt aus:

Dieser Tag kann anschliessend in der strongTNC Datenbank mittels POST Request auf die Ressource `/swid/add-tags/` erfasst werden. Die Daten für diesen Request bestehen aus einer JSON Liste von XML Strings, dadurch können auch mehrere Tags auf einmal erfasst werden. Empfehlenswert ist allerdings ein Batching in mehrere Gruppen, da die Requests sonst unter Umständen sehr lange dauern können.

```
<?xml version="1.0" encoding="UTF-8"?>
<SoftwareIdentity name="strongswan"
    uniqueId="debian_7.4-x86_64-strongswan-4.5.2-1.5+deb7u3"
    version="4.5.2-1.5+deb7u3" versionScheme="alphanumeric"
    xmlns="http://standards.iso.org/iso/19770/-2/2014/schema.xsd">
  <Entity name="strongSwan" regid="regid.2004-03.org.strongswan" role="tagcreator"/>
  <Entity name="HSR" regid="regid.2004-03.org.strongswan" role="publisher"/>
  <Payload>
    <File location="/usr/sbin" name="charon-cmd"/>
    <File location="/usr/sbin" name="strongswan"/>
    <File location="/usr/lib/systemd/system" name="strongswan.service"/>
    <File location="/usr/lib64/strongswan" name="libcharon.so.0"/>
    <File location="/usr/lib64/strongswan" name="libcharon.so.0.0.0"/>
    <File location="/usr/lib64/strongswan" name="libhydra.so.0"/>
    <File location="/usr/lib64/strongswan" name="libhydra.so.0.0.0"/>
    <File location="/usr/lib64/strongswan" name="libpttls.so.0"/>
    <File location="/usr/lib64/strongswan" name="libpttls.so.0.0.0"/>
    <File location="/usr/lib64/strongswan" name="libstrongswan.so.0"/>
    <File location="/usr/share/doc/strongswan-5.1.2" name="README"/>
  </Payload>
</SoftwareIdentity>
```

Listing 5: Beispiel eines SWID Tags

Falls bereits ein Tag mit der selben Software-ID existiert, wird er komplett überschrieben. Der Request sieht folgendermassen aus:

```
POST /api/swid/add-tags/ HTTP/1.1
Authorization: Basic cm9vdDpyb290
Host: tncserver:8000
Accept: application/json
Content-Type: application/json; charset="utf-8"
Content-Length: 239

{ "data" :
    [
        "<?xml version=\"1.0\" encoding=\"utf-8\"?><SoftwareIdentity name=\"fortune-mod\"...",
        "<?xml version=\"1.0\" encoding=\"UTF-8\"?><SoftwareIdentity name=\"strongswan\"..."
    ]
}
```

Listing 6: Erfassen eines SWID Tags

Curl Befehl:

```
curl -i -X POST http://tncserver:8000/api/swid/add-tags/ \
        -u username:password \
        -H "Accept: application/json" \
        -H "Content-Type: application/json; charset=utf-8" \
        -d "{ \"data\" : $DATA }"
```

Listing 7: Erfassen eines SWID Tags mit cURL

2.2.2 Antwort - Status 400 Bad Request

Falls der Request nicht in Ordnung ist (zB fehlerhaftes oder unvollständiges XML), könnte die Antwort wie folgt aussehen:

```
HTTP/1.0 400 BAD REQUEST
Date: Thu, 15 May 2014 21:31:10 GMT
Server: WSGIServer/0.1 Python/2.7.6
Vary: Accept, Cookie
Content-Type: application/json
Allow: POST, OPTIONS

{
    "status": "error",
    "message": "version: This field cannot be blank. package_name: This field cannot be blank."
}
```

Listing 8: Antwort beim Auftreten eines Fehlers

2.2.3 Antwort - Status 200 OK

Wenn mit dem Request alles in Ordnung ist, könnte die Antwort wie folgt aussehen:

```
HTTP/1.0 200 OK
Date: Thu, 15 May 2014 21:19:19 GMT
Server: WSGIServer/0.1 Python/2.7.6
Vary: Accept, Cookie
Content-Type: application/json
Allow: POST, OPTIONS

{
    "status": "success",
    "message": "Added 1 SWID tags, replaced 0 SWID tags."
}
```

Listing 9: Erfolgreiches Hinzufügen eines SWID Tags

3 Allgemeine Hinweise

- JSON-Objekte dürfen im Gegensatz zu Python nur in doppelte Anführungszeichen eingeschlossen werden. Beispiel: {`"foo": "bar"`} statt {`'foo': 'bar'`}.
- Alle übermittelten Daten sollten UTF-8 encoded sein. Der `Content-Type` Header sollte entsprechend übermittelt werden. Beispiel: `Content-Type: application/json; charset=utf-8`
- Nichtdruckbare Zeichen sowie doppelte Anführungszeichen innerhalb eines JSON Strings (zB bei den XML Tags) müssen mit einem Backslash escaped werden (z.B newline, tab, double quotes → `\n`, `\t`, `\"`).

4 API Endpoints

4.1 SWID Tags: Messung

URI Path `/sessions/{id}/swid-measurement/`
Archetype Controller
Methods POST
Content-Type `application/json; charset=utf-8`
Request Parameter
`softwareId` Software-IDs als JSON-Liste.
Response Statuscodes
200 OK SWID Tags der übermittelten Software-IDs sind eingetragen und wurden für die übermittelte Session eingetragen
404 Not Found Session mit der spezifizierten ID wurde nicht gefunden.
412 Precondition Failed Es existieren nicht alle SWID Tags für die übertragenen Software-IDs. Als Payload werden die fehlenden Software-IDs übertragen.
JSON Format Response

```
["<str, software-id>", ...]
```

4.2 SWID Tags: Erstellung

URI Path `/swid/add-tags/`
Archetype Controller
Methods POST
Content-Type `application/json; charset=utf-8`
Request Parameter
`xmlData` SWID Tags als JSON-Liste.
Response Statuscodes
200 OK SWID Tags wurden erfolgreich verarbeitet.
400 Bad Request Fehlerhafter Request. Details zum Fehler werden im Response-Body zurückgesendet.