HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

ins INSTITUTE FOR
NETWORKED SOLUTIONS

SWID Generator für Software Identification Tags mit File Hashes

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrssemester 2017

Autoren:	Christof Greiner
	Davide De Giorgio
Betreuer:	Prof. Dr. Andreas Steffen
Projektpartner:	INS – Institute for Networked Solutions

Änderungsgeschichte

Datum	Version	Änderung	Autor
22.2.2017	1.0	Aufbau Grundstruktur	Davide De Giorgio
05.3.2017	1.1	Vorgehen & Analyse ergänzt	Davide De Giorgio
01.6.2017	1.2	Umsetzung ergänzt	Davide De Giorgio und Christof Greiner
13.6.2017	1.3	Überarbeitung & Abschluss Dokumentation	Davide De Giorgio und Christof Greiner

Inhalt

I Einleitung

1 Abstract

Der Software-Identification(SWID)-Generator ist ein Konsolenprogramm, welches Metadaten über Linux Software Pakete in XML beschreibt. Die ursprüngliche Version basierte auf einer Bachelorarbeit aus dem Jahr 2014. Diese entsprach der damals gültigen, frühen Überarbeitung des ISO/IEC-19770-2:2009-Standards [1]. Die im Jahr 2015 erschienene, definitive Version des Standards[2] enthält diverse Änderungen welche umgesetzt werden müssen. Zudem hat sich auch der gewünschte Funktionsumfang des SWID-Generators erweitert.

Bevor die Weiterentwicklung beginnen konnte, musste sowohl die Bachelorarbeit aus dem Jahr 2014, als auch die ISO [2] und NISTIR [3] Standards und Empfehlungen gelesen, analysiert und die Kernpunkte entnommen werden. Anschliessend wurde der mit Python entwickelte SWID-Generator auf den aktuellen Standard portiert und zusätzlich die folgenden neuen Funktionalitäten implementiert:

- Erstellen von SWID-Tags aus Linux-Paketdateien
- Erstellen von SWID-Tags aus Ordnern auf dem Dateisystem
- Auswahl des zu verwendenden Hash-Algorithmus
- Digitale Signatur des SWID-Tags

Zusätzlich zu den aufgelisteten Funktionen wurde eine globale Fehlerbehandlung umgesetzt und die Testinfrastruktur erweitert.

Die finale Version des SWID-Generators ist nun vollkommen standardkonform und beinhaltet alle gewünschten Funktionalitäten. Neben den funktionalen Änderungen wurde das hohe Landscape-Codequalitätsniveau von 99% gehalten. Ausserdem konnte die Testabdeckung um etwa 30%, auf nun 80% verbessert werden. Die Applikation ist unter allen Debian, Redhat und Arch Linux basierten Systemen lauffähig und besitzt wenige externe Abhängigkeiten.

2 Aufgabenstellung

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

Bachelorarbeit 2017

SWID Generator for Software Identification Tags

Studenten: Davide De Giorgio, Christof Greiner

Betreuer: Prof. Dr. Andreas Steffen

Ausgabe: Montag, 22. Februar 2017

Abgabe: Freitag, 16. Juni 2017

Einführung

Im Jahr 2014 wurde in einer Bachelorarbeit ein SWID Generator [1] auf der Basis eines frühen Entwurfs des ISO/IEC 19770-2:2015 Software Identification Tag (SWID) Standards [2] erstellt. Da die entgültige Fassung des Standards stark vom Draft abweicht, soll der SWID Generator im Rahmen dieser Arbeit auf den aktuellen Stand gebracht werden. Dabei sollen auch die NIST Empfehlungen NISTIR 8060 - Guidelines for the Creation of Interoperable Software Identification Tags [3] beachtet werden.

Als neues Feature sollen auf der Basis der gängigen Linux Paket Manager (dpkg und rpm) die SHA-256 Hashes aller in einem Software Paket enthaltenen Files berechnet und in das SWID Tag des Software Pakets integriert werden, damit eine feste Beziehung zwischen Paket-Version und Hashwert etabliert werden kann. Dabei sollen bevorzugt die Hashwerte direkt aus den in den Paketen enthaltenen Files berechnet werden, ohne sie zuerst auf einem Referenzsystem installieren zu müssen.

Weiter sollen Lösungsvarianten aufgezeigt werden, wie Files mit variablen Hashwerten (z.B. Konfigurationsdateien) behandelt werden können (z.B. via das @n8060:mutable Extension Attribute [3]).

Aufgabenstellung

- Einarbeiten in den ISO/IEC 19770-2:2015 SWID Tag Standard und die NISTIR 8060 SWID Tag Empfehlungen.
- Update des bestehenden, in der Skriptsprache Python geschriebenen SWID Generators auf den Stand des publizierten ISO/IEC 19770-2:2015 SWID Tag Standards.
- Zusätzlich zu den Pfadnamen aller Dateien, die durch ein Linux Paket installiert werden, sollen die dazugehörigen SHA-256 Hashwerte der Dateien in das SWID Tag aufgenommen werden.
- Kennzeichnen von Dateien mit variablen Hashwerten in SWID Tags und erarbeiten eines Lösungsvorschlags, wie hostspezifische Konfigurationsdateien behandelt werden können, z. B. durch Patch Tags.

- **Optional:** Verwendung alternativer Hashfunktion wie z.B: SHA-384 oder SHA-512.
- **Optional**: Unterstützung weiterer Paket Manager wie z.B. pacman (Arch Linux).
- **Optional**: Signieren der SWID Tags mit dem private Key des tagCreators.

Links

[1] Git Repository des bestehenden SWID Generators
https://github.com/strongswan/swidGenerator

[2] ISO/IEC 19770-2:2015 Software Identification Tag (SWID) Standard
http://www.iso.org/iso/catalogue_detail?csnumber=65666

[3] NISTIR 8060 – Guidelines for the Creation of Interoperable SWID Tags
http://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8060.pdf

Rapperswil, 22. Februar 2017

Prof. Dr. Andreas Steffen

3 Eigenständigkeitserklärung

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Erklärung

Wir erklären hiermit,

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

Ort, Datum:

Rapperswil, 09.06.2017

Name, Unterschrift:

Christof Greiner

Davide De Giorgio

II Technischer Bericht

4 Analyse

In diesem Abschnitt werden alle Analysen erklärt und die Entscheidungen konkretisiert. Die Informationen, auf welchen die Entscheidungen beruhen, basieren auf folgenden drei Dokumenten:

- ISO/IEC 19770-2:2015 [2]
- NISTIR 8060 Guidelines [3]
- Bachelorarbeit 2014: Endpoint Compliance Monitoring based on Software Identification Tags [4]

4.1 Software Identification Tags

Ein Software-Identification-Tag ist eine standardisierte Datenstruktur, welche Informationen über eine Software beinhaltet. Diese Struktur basiert auf der XML-Syntax und wird von den Entwicklern der Software oder third-party-Providern generiert. Gebraucht werden diese von verschiedenen Stakeholdern, Discovery-Tools oder Services, welche diese Informationen aus Sicherheits-, Compliance- oder auch Logistik-Gründen sammeln.

Beispielszenario: Eine Firma möchte den VPN-Zugang in das Firmennetz sicherstellen. Vor dem Verbindungsaufbau wird der Client aufgefordert, alle SWID-Tags der installierten Pakete an den Server zu senden. Mittels der Version und der Dateien, welche der SWID-Tag beinhaltet, wird entschieden, ob dem Client der Zugang ins Netzwerk gewährt wird oder nicht.

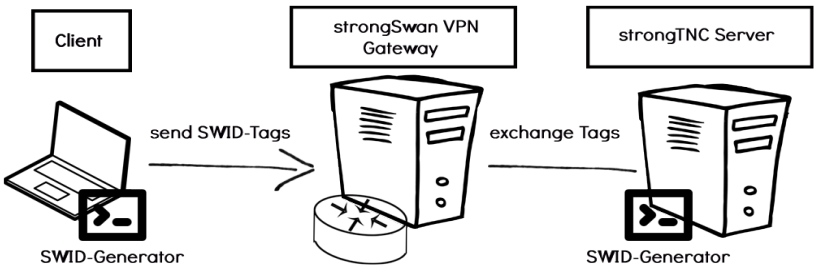

Abbildung 4.1: Beispielszenario - VPN Aufbau

4.2 IST Situation

Nachfolgend wird die Situation vor Beginn der Arbeit aufgezeigt. Dabei wird auf die Struktur der SWID-Tags und die Funktionalität des SWID-Generators eingegangen.

Die alte Struktur der SWID-Tags ist im ISO/IEC 19770-2:2009 [1] Entwurf definiert und die Analyse der Funktionalität stützt sich auf die *Version 0.3.0* des SWID-Generators.

4.2.1 SWID-Generator

Die Version 0.3.0 des Generators bietet die Möglichkeit Software-IDs und SWID-Tags einer installierten Software zu generieren. Dabei kann entweder eine Generierung aller installierten Pakete oder eine spezifische anhand des Paketnamens oder der Software-ID erfolgen.

Folgend wird die gesamte Funktionalität anhand der *Help*-Page aufgezeigt.

```
swid_generator swid : gibt die SWID-Tags aller installierten Pakete aus.
   --pretty : gibt Informationen in Menschenlesbarer Form aus.
   --full : gibt zusaetzlich alle Dateien dieses Pakets an.
   --package <package> : Limitierung auf ein spezifisches Paket.
   --software-id <SW-ID> : Limitierung auf eine spezifische Software-ID.
   --env <auto, dpkg, pacman, rpm> : explizite Angabe des Paketmanagers
   --regid <Regid> : explizite Angabe der Regid
   --doc-seperator <symbol> : explizite Angabe des Trennzeichens zwischen XML-Dokumenten
   --entity-name <entity> : explizite Angabe des im XML verwendeten Entity Namens
```

```
swid_generator software-id : gibt die Software-ID aller installieren Pakete aus.
   --env <auto, dpkg, pacman, rpm> : explizite Angabe des Paketmanagers
   --regid <Regid> : explizite Angabe der Regid
   --doc-seperator <symbol> : explizite Angabe des Trennzeichens zwischen XML-Dokumenten
```

4.2.2 SWID-Tag

Mit Hilfe des folgenden Befehls wird der SWID-Tag für das angegebene Paket generiert und auf der Konsole ausgegeben. Auf diese Weise kann die aktuelle Struktur eines SWID-Tags analysiert werden.

```
root@linux# swid_generator swid --pretty --package <package_name>
```

Nachfolgend wird ein minimaler SWID-Tag aufgezeigt:

```
<?xml version="1.0" encoding="utf-8"?>
<SoftwareIdentity name="texstudio" uniqueId="Ubuntu_16.04-x86_64-texstudio-2.10.8~debian-1"
     version="2.10.8+debian-1" versionScheme="alphanumeric"
     xmlns="http://standards.iso.org/iso/19770/-2/2014/schema.xsd">
  <Entity name="strongSwan Project" regid="regid.2004-03.org.strongswan" role="tagcreator"/>
</SoftwareIdentity>
```

Damit der SWID-Tag zusätzlich alle zugehörigen Dateien auflistet, muss dem vorherigen Befehl das Argument `--full` übergeben werden.

```
root@linux# swid_generator swid --pretty --full --package <package_name>
```

```
<?xml version="1.0" encoding="utf-8"?>
<SoftwareIdentity name="texstudio"
   uniqueId="Ubuntu_16.04-x86_64-texstudio-2.10.8~debian-1"
   version="2.10.8+debian-1" versionScheme="alphanumeric"
   xmlns="http://standards.iso.org/iso/19770/-2/2014/schema.xsd">
<Entity name="strongSwan Project" regid="regid.2004-03.org.strongswan" role="tagcreator"/>
   <Payload>
     <File location="/usr/bin" name="texstudio"/>
     <File location="/usr/share/lintian/overrides" name="texstudio"/>
     <File location="/usr/share/applications" name="texstudio.desktop"/>
     <File location="/usr/share/doc/texstudio" name="copyright"/>
     <File location="/usr/share/icons/hicolor/128x128/apps" name="texstudio.png"/>
     <File location="/usr/share/texstudio" name="de_DE.stopWords"/>
   </Payload>
</SoftwareIdentity>
```

Anhand dieses Outputs können die Unterschiede zum neuen Standard[2] analysiert werden.

4.3 SOLL Situation

Welche Neuerungen dazukommen und welche Änderungen gemacht werden müssen, wird in diesem Abschnitt präzisiert. Die definierten Änderungen basieren auf dem ISO Standard aus dem Jahr 2015 [2].

4.3.1 ISO/IEC 19770-2:2015

Der Standard ISO/IEC 19770-2:2015[2] konzentriert sich auf die Standardisierung der Software Identification Tags. Das Schema der SWID-Tags ist online [1] verfügbar.
Der SWID-Tag muss im Minimum die im ISO/IEC 19770-2:2015 [2] Abschnitt *8.3 Recommended SWID tag Data values* definierten Informationen beinhalten.

- SotwareIdentity
  - name
  - tagVersion
  - tagId
  - version
  - versionScheme
- Entity
  - name
  - regId
  - role

[1] SWID-XML-Schema: http://standards.iso.org/iso/19770/-2/2015-current/schema.xsd

4.3.2 NISTIR 8060 Guidelines

Die NISTIR8060 [3] Guidelines beinhalten Empfehlungen zum Standard [2] und der konkreten Implementation von SWID-Tags. Zudem definiert dieses Paper ein XML-Extension-Schema, welches verschiedene zusätzliche Attribute beschreibt. Das Dokument ist online [2] verfügbar.

4.3.3 Aufbau SWID-Tag

In diesem Abschnitt werden die wichtigsten Elemente und deren Attribute und Inhalte aufgelistet und beschrieben. Dies ist die Basis für die spätere Implementation der SWID-Tags.

SoftwareIdentity Root-Element des Software Identification Tags

xmnls	http://standards.iso.org/iso/19770/-2/2015-current/schema.xsds	Required
name	Name der Software	Required
tagId	Unique-Id des SWID-Tags	Required
version	Version der Software	Optional
versionScheme	Schema der Software Versionsnummer (alphanumeric, etc.)	Optional

Tabelle 4.1: SoftwareIdentity-Tag: Wichtigste Attribute

Entity Informationen über den Aussteller des SWID-Tags

name	Name des Ausstellers	Required
role	Rolle des Ausstellers. Mögliche Auswahl: aggregator, distributor, licensor, softwareCreator, softwareCreator und tagCreator. Die Rolle "tagCreator" muss zwingend enthalten sein. Möglich sind auch mehrere Rollen (Trennung durch Abstand).	Required

Tabelle 4.2: Entity-Tag: Wichtigste Attribute

[2] XML-Extension: http://csrc.nist.gov/schema/swid/2015-extensions/swid-2015-extensions-1.0.xsd

Payload Beinhaltet alle *File* und *Directory* Einträge der Software. Dieser Tag dient lediglich als Container und hat keine Attribute.

Directory Bildet einen Ordner ab und kann beliebig viele *File* und *Directory* Elemente beinhalten.

name	Name des Ordners	Required
root	Start des Pfades. Wenn die Root nicht definiert ist, wird der Pfad gewählt an dem der SWID-Tag als Datei gespeichert ist. Kann auch Umgebungsvariablen beinhalten.	Optional
regId	Unique-Id des Ausstellers. Wenn diese nicht bekannt ist, wird standardmässig *http://invalid.unavailable* als regId gewählt. Dieses Feld soll URI-Konform sein.	Required

Tabelle 4.3: Directory-Tag: Wichtigste Attribute

File Informationen über eine Datei

name	Name der Datei	Required
size	Dateigrösse in Bytes	Optional
SHA256:hash	SHA256 Hash-Wert	Optional
SHA384:hash	SHA384 Hash-Wert	Optional
SHA512:hash	SHA512 Hash-Wert	Optional
@n8060:mutable	*true* oder *false*. Im Extension-Schema des NISTIR8060 [3] definiert. Markiert veränderliche Dateien (config-Files, etc.).	Optional

Tabelle 4.4: File-Tag: Wichtige Attribute

Kompletter SWID-Tag Das neue Schema nach ISO-Standard [2], anhand eines Beispiels aufgezeigt.

```
<?xml version="1.0" encoding="utf-8"?>
<SoftwareIdentity name="texstudio"
   tagId="Ubuntu_16.04-x86_64-texstudio-2.10.8~debian-1" version="2.10.8+debian-1"
   versionScheme="alphanumeric"
   xmlns="http://standards.iso.org/iso/19770/-2/2014/schema.xsd"
   xmlns:SHA256="http://www.w3.org/2001/04/xmlenc#sha256"
   n8060="http://csrc.nist.gov/schema/swid/2015-extensions/swid-2015-extensions-1.0.xsd">
  <Entity name="strongSwan Project" regId="strongswan.org" role="tagCreator"/>
  <Payload>
     <Directory root="/etc" name="texstudio">
        <File name="texstudio.ini" SHA256:hash="P015A3BF4F1B2B..." mutable="true"/>
        <File name="texstudio" SHA256:hash="G015A3BF4F1B2B..." mutable="true"/>
     </Directory>
     <Directory root="/usr/bin/texstudio" name="languagepacks">
        <File name="en_US.lang-pack" SHA256:hash="9F86D081884C7...">
        <File name="de_DE.lang-pack" SHA256:hash="D015A3BF4F1B2...">
     <Directory root="/usr/bin/texstudio" name="icons">
        <File name="32x32.ico" SHA256:hash="CF86D081884C7...">
        <File name="64x64.ico" SHA256:hash="D015A3BF4F1B2...">
     </Directory>
  </Payload>
</SoftwareIdentity>
```

Nach dem ISO-Standard [2] können die *Directory*-Elemente, um redundante Sub-Pfade zu vermeiden, auch beliebig verschachtelt werden. Aus Kompatibilitätsgründen zu strongTNC wurde entschieden, die flache (Directory $\rightarrow$ File) Struktur als Standard zu verwenden und optional die Generierung von verschachtelten Strukturen anzubieten.

4.4 Unique Identifications

Tag Identifier Der Tag Identifier (Attribut *tagId* des *SoftwareIdentity*-Tags) soll global eindeutig sein. Im Abschnitt *6.1.6 Tag identifier* des ISO/IEC 19770-2:2015 [2] wird die "*uniqueness*" wie folgt definiert:

(...) It is recommended that tagId be a minimum of a 16 byte globally unique identifier (GUID), but it may also be a string that is constructed by the tag provider or another globally unique value. If a string construct is used, the tagId shall follow the restrictions for URI character use as specified in IETF RFC 3986 [5], characters.

Es ist jedem Tag Provider erlaubt, eigene Unique Identifier für den SWID Tag zu definieren. In der vorherigen Bachelorarbeit wurde die Unique Identification der Tags im Abschnitt *6.4.3.3 Doppelpunkte in der UniqueId* [4] wie folgt definiert:

`Schema: <OS>_<Arch>_<Package>_<Version>`

Da die Definition im Standard unverändert blieb, wurde an der Entscheidung der Vorgänger festgehalten und dieses Schema weiterverwendet.

Registration Identifier Der Registration Identifier (Attribut *regId* des *Entity*-Tags) ist eine eindeutige Bezeichnung, die den Software-Aussteller identifiziert. Die Struktur dieser ID wurde im Kapitel *6.4.1 Bestandteile eines SWID Tags*[4] wie folgt definiert:

`Schema: regid.YYYY-MM.<reverse domain name>`

Die Struktur hat sich im neuen Standard verändert. Im Abschnitt *6.1.5.2 Structure of regid* [2] werden folgende neuen Restriktionen definiert:

- *Unless otherwise required, the URI should utilize the http scheme.*
- *If the http scheme is used, the "http://" may be left off the regid string (a string without a URI scheme specified is defined to use the "http://" scheme).*
- *Unless otherwise required, the URI should use an absolute-URI that includes an authority part, such as a domain name.*
- *To ensure consistency, the absolute-URI should use the minimum string required (for example, example.com should be used instead of www.example.com).*

Anhand dieser Einschränkungen wurde entschieden, den aktuellen Namen "*regid.2004-03.org.strongswan*" zu "*strongswan.org*" zu ändern.

4.5 Berechnung der Hash-Werte

Neu wird im Standard auch die Option geboten die *File*-Elemente mit Hash-Werten zu ergänzen, was in einem `<Payload>`-Element sogar zwingend ist. Als Standard-Hash-Algorithmus wird der Empfehlung des ISO-Standards [2] gefolgt und der SHA-256-Algorithmus verwendet. Dieser bietet einen guten Kompromiss zwischen Performance, Grösse und Sicherheit. Alternativ soll der Algorithmus mittels Parameter durch einen anderen (SHA-384, SHA-512) ersetzt bzw. ergänzt werden können.

4.6 Konfigurations-Files

Ein wichtiger Punkt ist die Erkennung der veränderlichen Dateien (z.B. Konfigurationsdateien). Denn durch Veränderung dieser stimmen die Hashwerte nicht mehr mit den originalen Werten überein. Daher müssen diese Einträge im SWID-Tag speziell gekennzeichnet werden. Um dies zu erreichen, wurde im NISTIR 8060[3] das Attribut "`@n8060:mutable`" eingeführt.

Das Erkennen der veränderlichen Datei stellte sich als nicht trivial heraus. Die verschiedenen Paketmanager bieten zudem unterschiedlich gute Möglichkeiten an.

dpkg/apt Durch das Abfragen der Paketliste im System können die Konfigurationsdateien mit aufgelistet werden. Dies wird erreicht durch das Ergänzen des Aufrufs mit dem Parameter "`${conffiles}`".

Befehl:

```
dpkg-query -W -f="${conffiles}\\n" <package_name>
```

Output:

```
/etc/apt/apt.conf.d/01autoremove 0b1391c01d75f95fa4ea5ac01219b515
/etc/cron.daily/apt-compat bc4a71cbcaeed4179f25d798257fa980
/etc/kernel/postinst.d/apt-auto-removal 8ad76ae4492b54f0dcbf18c79429dfab
```

rpm Bietet einen konkreten Befehl um die Konfigurationsdateien eines spezifischen Pakets auszugeben.

Befehl:

```
rpm -qa --queryformat "%{name}\n" -c <package_name>
```

Output:

```
/etc/sysconfig/docker-network
/etc/sysconfig/docker-storage
/etc/sysconfig/docker-storage-setup
```

pacman Bietet keinerlei Möglichkeiten an, die Konfigurationsdateien abzufragen.

Backup-Szenario Konfigurationsdateien werden jeweils an einem einheitlichen Ort auf dem Dateisystem abgelegt, in den meisten Fällen im Ordner "`/etc/`". Aus diesem Grund wurden die Dateien in diesem Ordner als veränderlich angenommen.

4.7 Package-Queries

Eine neue Funktion des SWID-Generators soll die SWID-Tags direkt aus den Paketdateien der verschiedenen Distributionen generieren. Das Extrahieren der benötigten Informationen wird mithilfe der Paketmanagern gemacht. Durch die Abfragen sollten folgende Informationen aus den Paketdateien entnommen werden können:

- Paket-Version und Name
- Liste der Dateien
- Liste der Konfigurationsdateien

Im nachfolgenden Abschnitt wird pro Distribution die jeweils beste Vorgehensweise für die Abfrage dieser Informationen aufgelistet und beschrieben.

4.7.1 Debian [6]

Package-Version und Name

```
dpkg -f <path_to_file> Package // Output: zsh-1
dpkg -f <path_to_file> Version // Output: 2:0-2
```

Liste der Dateien

```
dpkg -c <path_to_file>
```

Output:

```
-rw-r--r-- root/root    1300 2016-04-05 21:15 ./etc/docker/README
lrwxrwxrwx root/root       0 2016-07-15 15:33 ./etc/docker/do.gz -> docker.gz
lrwxrwxrwx root/root       0 2016-07-15 15:33 ./etc/docker/control -> ../docker-bin/ctrl
```

Liste der Konfigurationsdateien Mit dem *dpkg*-Tool ist es nicht möglich dem Paket direkt eine Liste der Konfigurationsdateien zu entnehmen. Jedoch beinhaltet das Paket eine Datei namens *conffiles*, welche diese Information beinhaltet. Diese kann direkt aus dem Paket extrahiert und anschliessend interpretiert werden. Die Datei wird folgendermassen entpackt:

```
dpkg -x <path_to_package> <path_to_save_location> # Schritt 1: Inhalt des Pakets entpacken
ar x <path_to_package> <name_of_control_archive> # Schritt 2: "control" entnehmen
tar -zxf <path_to_control_archive> conffiles # Schritt 3: "conffiles"-Datei aus Archiv extrahieren
```

Folgend wird der Aufbau eines .deb-Pakets illustriert:

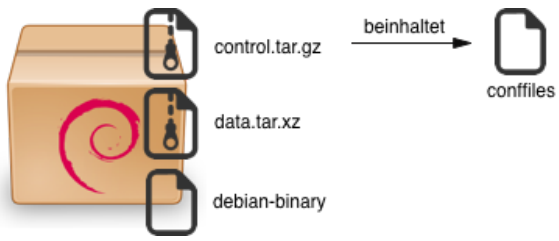

Abbildung 4.2: Aufbau eines Debian-Pakets

Inhalt der *conffiles*-Datei:

```
/etc/apt/apt.conf.d/01autoremove 0b1391c01d75f95fa4ea5ac01219b515
/etc/cron.daily/apt-compat bc4a71cbcaeed4179f25d798257fa980
/etc/kernel/postinst.d/apt-auto-removal 8ad76ae4492b54f0dcbf18c79429dfab
```

4.7.2 Redhat [6]

Package-Version und Name

```
rpm --query --package --queryformat "%{name}" <path_to_file> // Output: zsh-1
rpm --query --package --queryformat "%{version}" <path_to_file> // Output: 1.0-2
```

Liste der Dateien

```
rpm --query --package <path_to_file> -l
```

Output:

```
/etc/docker
/etc/docker/certs.d
/etc/docker/certs.d/redhat.com
/var/lib/docker
```

Liste der Konfigurationsdateien

```
rpm --query --package <path_to_file> -c
```

Output:

```
/etc/sysconfig/docker-network
/etc/sysconfig/docker-storage
/etc/sysconfig/docker-storage-setup
```

4.7.3 Archlinux [7]

Package-Version und Name

```
pacman --query --file <path_to_file> // Output: zsh-1 1.0-2
```

Liste der Dateien

```
pacman -Qlp <path_to_file>
```

Output:

```
docker /usr/bin/docker.ini
docker /usr/bin/docker
docker /usr/bin/docker-containerd
```

Liste der Konfigurationsdateien Auf Arch Linux Systemen gibt es keine Möglichkeit, direkt aus dem Paket eine Liste der Konfigurationsdateien zu extrahieren. Aus diesem Grund wurde nach Absprachen und Recherchen entschieden, auf das Backup-Szenario zurückzugreifen und die Datei-Einträge im "*/etc/*"-Ordner als Konfigurationsdateien zu markieren.

4.8 XML Signatur

Signaturen bieten sich besonders an um sicherzustellen, dass Informationen nicht verändert worden sind und von der gewünschten Person bzw. vom gewünschten Gerät stammen. Speziell im Sicherheitsbereich in dem sich strongSwan/strongTNC und somit auch der SWID-Generator befinden, ist dies von elementarer Bedeutung. Daher soll der SWID-Generator um die Funktion erweitert werden, die erstellten XML-Tags mit einem Zertifkat und zugehörigem Privat-Key digital zu signieren.
Zur Erstellung von XML Signaturen hat die W3C im Jahr 2002 das Dokument "XML Signature Syntax and Processing" [8] herausgegeben. Die aktuell gültige Empfehlung ist die überarbeitete Version aus dem Jahr 2013 [9].

4.8.1 Varianten

Der Standard definiert drei grundsätzlich verschiedene Arten, ein Dokument digital zu signieren. Diese unterscheiden sich in der Art, wo der `<Signature>`-Tag platziert wird.

Detached wird verwendet um eine Signatur über eine externe Ressource zu speichern, welche typischerweise über eine URI spezifiziert ist.

Enveloping verwendet das `<Signature>`-Tag als Root-Element und speichert darin neben den Signatur-Informationen das originale XML.

Enveloped behält als Root-Objekt das Original-XML und inkludiert in diesem ein `<Signature>`-Tag mit allen Signatur-Informationen.

Für die Verwendung mit dem SWID-Generator bietet sich die "*Enveloped*" Variante an, weil dadurch die Grundstruktur des Tags nicht verändert wird und sich die Anpassungen am strongTNC Parser in einem kleinen Rahmen halten.

4.8.2 Tools

Die Evaluation hat gezeigt, dass es zwei akzeptable Varianten gibt, XMLs mit Python digital zu signieren. Diese haben verschiedene Eigenschaften, Vor- und Nachteile auf welche nachfolgend eingegangen wird.

signxml Python Library
Hierbei handelt es sich um eine externe Python Library welche mittels `pip` direkt aus dem *PyPI* [3] installiert werden kann. Anschliessend kann mit folgendem Befehl ein XML ElementTree Objekt direkt in Python digital signiert werden.

```
xmlsigner = signxml.XMLSigner(method=signxml.methods.enveloped)
signed_xml = xmlsigner.sign(plain_xml, key=KEY, cert=CERTS)
```

`KEY` ist dabei eine eingelesene *.key*-Datei und `CERTS` ist eine Liste aller einzeln eingelesenen *.pem*-Dateien der Zertifikats-Kette.

Dies ist auch der grösste Nachteil dieser Variante, denn alle entsprechenden Zertifikate, der Key und das zugehörige Passwort müssen einzeln mit Argumenten entgegengenommen werden.

xmlsec1 Linux Programm
`xmlsec1`[14] ist ein in C geschriebenes Linux Kommandozeilen-Programm, welches mit allen Paketmanagern installiert werden kann. Mit `apt` und `rpm` unter dem Namen *xmlsec1* und mit `pacman` unter dem Namen *xmlsec*.
Der Aufruf sieht wie folgt aus:

```
xmlsec1 --sign --pkcs12 PKCS12 --pwd PASS XML
```

`PKCS12` ist dabei ein *pkcs12*-Container, welcher den Private Key und alle nötigen Zertifikate der Kette enthält. `PASS` enthält das optionale Passwort zum *pkcs12*-Container. `XML` ist die XML-Datei, welche signiert werden soll.

Da es in Bezug auf die Komplexität der Implementation und der Abhängigkeiten keine gravierenden Unterschiede gibt, wurde aufgrund der Einfachheit für den Benutzer entschieden. Da mit `xmlsec1` alle nötigen Dateien in einem *pkcs12*-Container übergeben werden, ist es für den Benutzer wesentlich angenehmer aufzurufen als in die Variante mit `signxml`.

[3] Python Package Index: https://pypi.python.org/pypi

4.9 Requirements

4.9.1 Zweck

Der SWID-Generator verfolgt den Zweck Informationen über Softwarepakete, die entweder installiert sind oder als Paket-Datei vorliegen, in einem standardkonformen Format auszugeben. Verwendung kann der Generator daher in verschiedensten Bereichen finden. Konkret entwickelt wird er allerdings für die Verwendung mit strongSwan und strongTNC. Er findet sowohl Verwendung auf der Client Seite, um die installierten Pakete zu erkennen, als auch auf der Server Seite um die nötigen Referenzwerte zu generieren.

4.9.2 UseCases

Für die Erweiterung des Swid-Generators wurden die folgenden neuen oder angepassten UseCases identifiziert. Diese ergänzen die bereits abgedeckten UseCases der Vorgängerarbeit [4].

Angepasste UseCases Die Änderungen im Vergleich zur Vorarbeit werden mit grauer Hinterlegung hervorgehoben.

UC3 - SWID-Tag Generierung

Akteur	strongSwan IMC
Story	Der Akteur will SWID-Tags aller installierten Pakete generieren. Das Format dieser XML-Dokumente folgt dem ISO-Standard 19770-2:2015[2]. Für jedes installierte Paket wird ein eigenes XML-Dokument generiert.
Success Szenario	Alle Tags werden generiert und auf der Standardausgabe durch Newlines getrennt ausgegeben. Es wird ein Tag pro Zeile dargestellt. Die Attribute des Tag Creators werden mit vordefinierten Werten befüllt. Es ist ein definiertes Set von 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 mit einem dazugehörenden SHA-256-Hash auflistet.
Alternatives Szenario 5	Mittels optionalem Parameter kann der zu verwendende Hash-Algorithmus angegeben werden.
Alternatives Szenario 6	Mittels optionalem Parameter kann ein Zertifikat angegeben werden mit welchem die SWID-Tags signiert werden.

Tabelle 4.5: UC3 - SWID-Tag Generierung

UC4 - Targeted Request

Akteur	strongSwan IMC
Story	Der Akteur möchte nur den SWID-Tag eines bestimmten Paketes erhalten. Das Format dieser XML-Dokumente folgt dem ISO-Standard 19770-2:2015[2].
Success 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 einfacher 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 mit einem dazugehörenden SHA-256-Hash auflistet.
Alternatives Szenario 4	Mittels optionalem Parameter kann der zu verwendende Hash-Algorithmus angegeben werden.
Alternatives Szenario 5	Mittels optionalem Parameter kann ein Zertifikat angegeben werden, mit welchem die SWID-Tags signiert werden.

Tabelle 4.6: UC4 - Targeted Request

Neue UseCases Die folgenden UseCases wurden neu erkannt und werden in der Arbeit ebenfalls ergänzt.

UC5 - Targeted File Request

Akteur	strongTNC Server
Story	Der Akteur möchte den SWID-Tag einer nicht installierten Paket-Datei erhalten. Das Format dieses XML-Dokuments folgt dem ISO-Standard 19770-2:2015[2].
Success Szenario	Mittels Parameter kann dem Generator eine Paket-Datei angegeben werden, aus welcher ohne vorherige Installation der SWID-Tag generiert wird.
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 einfacher 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 mit einem dazugehörenden SHA-256-Hash auflistet.
Alternatives Szenario 4	Mittels optionalem Parameter kann der zu verwendende Hash-Algorithmus angegeben werden.
Alternatives Szenario 5	Mittels optionalem Parameter kann ein Zertifikat angegeben werden mit welchem die SWID-Tags signiert werden.

Tabelle 4.7: UC5 - Targeted File Request

UC6 - Targeted Filesystem Request

Akteur	strongTNC Server & strongSwan IMC
Story	Der Akteur möchte den Inhalt eines Ordners auf dem Dateisystem in einem SWID-Tag abgebildet erhalten. Das Format dieses XML-Dokuments folgt dem ISO-Standard 19770-2:2015[2].
Success Szenario	Mittels Parameter kann dem Generator ein Pfad auf dem Dateisystem angegeben werden, aus welchem ein SWID-Tag generiert wird.
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 einfacher lesbaren Form ausgegeben werden («pretty printing»).
Alternatives Szenario 3	Mittels optionalem Parameter können die XML-Tags mit einem Evidence-Element versehen werden, welches die enthaltenen Dateien mit einem dazugehörenden SHA-256-Hash auflistet.
Alternatives Szenario 4	Mittels optionalem Parameter kann der zu verwendende Hash-Algorithmus angegeben werden.
Alternatives Szenario 5	Mittels optionalen Parametern können der Name, die Version und der "Root"-Ordner verändert werden.
Alternatives Szenario 6	Mittels optionalem Parameter kann ein Zertifikat angegeben werden, mit welchem die SWID-Tags signiert werden.

Tabelle 4.8: UC6 - Targeted Filesystem Request

5 Vorgehen

In diesem Abschnitt wird die geplante Vorgehensweise aufgezeigt. Weiter wird erläutert, wie die Sicherung der Qualität (Organisation und Code-Qualität) sichergestellt wird.

5.1 Projektplanung

Zu Beginn der Arbeit wurde ein Planungsdokument erstellt um das Projekt richtig einschätzen und mögliche Risiken bereits früh erkennen zu können. Dieses Dokument ist im Appendix A angefügt.

5.2 Aktueller Code

Um sauber und getrennt vom offiziellen strongSwan Repository[4] arbeiten zu können, wurde zu Beginn der Arbeit der aktuelle Stand in ein neues Repository[5] auf GitHub geforked.

5.3 Qualitätssicherung

Durch die nachfolgend aufgelisteten Massnahmen wird versucht das Qualitätsniveau beizubehalten.

Continuous Integration Wie im technischen Bericht der Bachelorarbeit aus dem Jahre 2014 [4] zu entnehmen ist, wird für das CI der Online-Service Travis-CI verwendet. Um den funktionierenden Build-Prozess nicht zu verändern, wurde die Verwendung von Travis-CI [6] weitergeführt. Im Rahmen dieser Arbeit wurde allerdings ein neuer Account auf Travis-CI eröffnet und das neue Repository verlinkt.

Code-Qualität Auch in diesem Bereich haben die Vorgänger gute Arbeit geleistet und haben das Environment stabil aufgesetzt. Um eine stetige Codeanalyse und Berechnung der Software Metriken zu garantieren, wurde der online Service Landscape [7] verwendet. Dieser berechnet bei jedem Push auf das Repository die Metriken neu und sendet dem Entwickler das Resultat der Analyse.

Test-Coverage Ziel ist es eine hohe Test-Abdeckung zu erreichen. Damit dies auch analysiert und zeitlich nachverfolgt werden kann, wird auch an dieser Stelle der bestehende Service [8] verwendet.

[4] strongSwan Repository: https://github.com/strongswan/swidGenerator
[5] Privates Repository: https://github.com/degiorgioo/swidGenerator
[6] Travis-CI: https://travis-ci.org
[7] Landscape: https://landscape.io
[8] Coveralls: https://coveralls.io

Aktuelle Situation	Test-Coverage (Coveralls)	50%
	Code-Qualität (Landscape)	99%
	Letzter Build (Travis-CI)	erfolgreich

Fazit und Ziel Wie die Zahlen aufzeigen, wurde sehr gut gearbeitet, lediglich die Test-Abdeckung könnte höher sein. Ziel ist es, die Code-Qualität aufrecht zu halten und die Test-Abdeckung, wenn möglich, zu erhöhen.

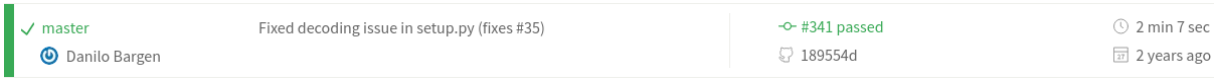

Abbildung 5.3: Letzter Build-Status

Abbildung 5.4: Test-Coverage

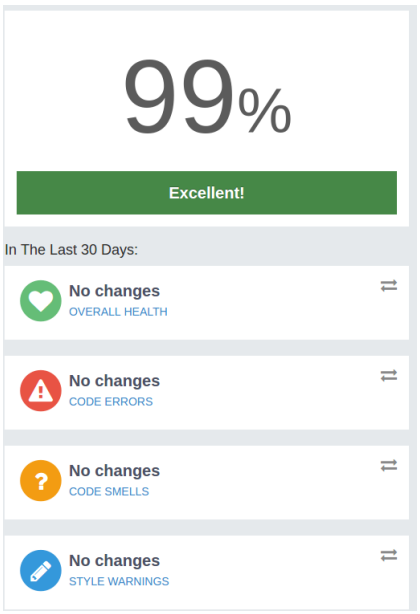

Abbildung 5.5: Landscape-Qualität

6 Umsetzung

Durch sorgfältige Planung der Umsetzung konnte erreicht werden, dass sich die Änderungen in die bestehende Architektur und Struktur einfügt und diese daher grösstenteils erhalten bleibt. Dadurch wurde auch eine unnötige Aufblähung der Codebasis verhindert.

6.1 Aufbau Payload Tag

In visueller Hinsicht sind die Änderungen im `<Payload>`-Tag mit Abstand die grössten. Dieser optionale Tag wurde wie im Abschnitt 4.3 *SOLL Situation* definiert angepasst, um so alle Punkte des neuen ISO-Standards [2] abzudecken.
Wie ursprünglich geplant, wurde zu Beginn lediglich eine Directory-File Stufe implementiert, um das Auslesen für den strongTNC Server möglichst einfach zu gestalten.

```
<Payload>
  <Directory root="/etc" name="texstudio">
    <File name="texstudio.ini" SHA256:hash="P015A3BF4F1B2B..." mutable="true"/>
    <File name="texstudio" SHA256:hash="G015A3BF4F1B2B..." mutable="true"/>
  </Directory>
  <Directory root="/usr/bin/texstudio" name="languagepacks">
    <File name="en_US.lang-pack" SHA256:hash="9F86D081884C7...">
    <File name="de_DE.lang-pack" SHA256:hash="D015A3BF4F1B2...">
  <Directory root="/usr/bin/texstudio" name="icons">
    <File name="32x32.ico" SHA256:hash="CF86D081884C7...">
    <File name="64x64.ico" SHA256:hash="D015A3BF4F1B2...">
  </Directory>
</Payload>
```

Diese flache Struktur hat allerdings den Nachteil, dass zum Teil viel redundanter Text entstehen kann. Wie oben ersichtlich, ist der Directory-Root-Tag "*/usr/bin/texstudio*" bereits in diesem kleinen Beispiel doppelt vorhanden. Dadurch leidet die Lesbarkeit und auch die Grösse des Outputs steigt. Als Resultat wurde in Hinblick auf eine schöne und platzsparende Struktur auch eine hierarchische Verschachtlung als alternatives Ausgabeformat implementiert. Ausgewählt wird sie mit dem `--hierarchic` Argument.

```
<Payload>
  <Directory root="/etc" name="texstudio">
    <File name="texstudio.ini" SHA256:hash="P015A3BF4F1B2B..." mutable="true"/>
    <File name="texstudio" SHA256:hash="G015A3BF4F1B2B..." mutable="true"/>
  </Directory>
  <Directory root="/usr/bin/" name="texstudio">
    <Directory name="languagepacks">
      <File name="en_US.lang-pack" SHA256:hash="9F86D081884C7...">
      <File name="de_DE.lang-pack" SHA256:hash="D015A3BF4F1B2...">
    </Directory>
    <Directory name="icons">
      <File name="32x32.ico" SHA256:hash="CF86D081884C7...">
      <File name="64x64.ico" SHA256:hash="D015A3BF4F1B2...">
    </Directory>
</Payload>
```

6.2 File-Hashes

Neu besitzen alle *File*-Tags mindestens ein Hash-Attribut. Wenn mit dem Aufruf des SWID-Generators keine explizite Auswahl des Hash-Algorithmus erfolgt, wird "*SHA-256*" verwendet. Alternativ kann mittels dem optionalen `--hash` Argument ein oder mehrere Algorithmen angegeben werden.
Wenn mehr als ein Hash benötigt wird, kann dies mit einer kommagetrennten Liste erreicht werden. Dabei werden diese der Grösse nach sortiert.
Die aktuelle Version unterstützt "*SHA-256*", "*SHA-384*" und "*SHA-512*". Zu erwähnen ist, dass die Grösse des SWID-Tags durch die grösseren Hashes schnell zunimmt.

```
root@linux# swid_generator swid --full --pretty --hash sha256,sha384,sha512
```

```
<Directory name="man8" root="/usr/share/man/sv">
<File SHA256:hash="1c74e2b68ecac0568113897..."
        SHA384:hash="f4d980cc87af7aa8cbdec0034f77d46d03f2..."
        SHA512:hash="e7fbee0e644fa20678fe4e7233067cc12ecf7a29bb7eecf381b9..."
        name="start-stop-daemon.8.gz" size="4758"/>
</Directory>
```

Es wurde besonders darauf geachtet, dass eine Erweiterung des SWID-Generators um weitere Algorithmen einfach zu realisieren ist.

6.3 Package File / Package Queries

Die Abfrage-Möglichkeiten der Paketmanager wurden im Abschnitt 4.7 *Package-Queries* bereits erklärt. Unter den Environments von *RPM* und *Pacman* sind keine Schwierigkeiten aufgetreten und das geplante Vorgehen konnte umgesetzt werden.

6.3.1 DPKG - .deb Pakete

Nachfolgend wird der Spezialfall *DPKG* erläutert und was bei der Interpretation beachtet werden musste.

Datei-Liste Basis der Verarbeitung ist folgender Output:

```
-rw-r--r-- root/root    1300 2016-04-05 21:15 ./etc/docker/README
lrwxrwxrwx root/root       0 2016-07-15 15:33 ./etc/docker/do.gz -> docker.gz
lrwxrwxrwx root/root       0 2016-07-15 15:33 ./etc/docker/control -> ../docker-bin/ctrl
```

1. *Normale Datei*: Pfad zur Datei ist der letzte Abschnitt jeder Zeile. Aus dem String des Pfades wurde lediglich der vordere Punkt abgetrennt.
2. *Symbolic-Link*: Hier müssen beide Einträge im SWID-Tag aufgelistet sein: Der Link und die Datei auf welcher der Link verweist.

   (a) Ohne *Ordner-Rücksprung*: `./etc/docker/do.gz` → `docker.gz`
   Symbolic-Link: `/etc/docker/do.gz`
   Verweis auf Datei: `/etc/docker/docker.gz`

   (b) Mit *Ordner-Rücksprung*: `./etc/docker/control` → `../docker-bin/ctrl`
   Symbolic-Link: `/etc/docker/control`
   Verweis auf Datei: `/etc/docker-bin/ctrl`

Konfigurations-Dateien Basis der Verarbeitung ist der Inhalt der Datei *conffiles*, welche wie folgt aufgebaut ist:

```
/etc/apt/apt.conf.d/01autoremove 0b1391c01d75f95fa4ea5ac01219b515
/etc/cron.daily/apt-compat bc4a71cbcaeed4179f25d798257fa980
/etc/kernel/postinst.d/apt-auto-removal 8ad76ae4492b54f0dcbf18c79429dfab
```

Wie im Output ersichtlich, steht hinter dem Pfad auch ein md5-Hash-Wert, welcher für die Weiterverarbeitung abgetrennt wird. Falls keine *conffiles*-Datei vorhanden ist, wird angenommen, dass keine Konfigurationsdateien existieren.

6.4 Evidence

Durch den stark strukturierten Aufbau der Applikation konnte die zusätzliche Anforderung, auch Dateisystempfade in SWID-Tags abzubilden, ohne viel Aufwand implementiert werden. Mit dem Parameter `--evidence PATH` kann ein beliebiger Ordner auf dem Dateisystem angegeben werden. Sowohl Name als auch Version werden mit Default-Werten gefüllt. Diese können mittels optionalen Parametern angepasst werden.

Weiter kann der *Root*-Pfad im SWID-Tag geändert werden mit dem Parameter `--new-root PATH`.

Der Aufruf

```
swid_generator swid --full --evidence /home/swid/generator --new-root /ownpath
```

ändert somit die Ausgabe von

```
<Directory name="projects" root="/home/swid/generator">
```

zu

```
<Directory name="projects" root="/ownpath">
```

6.5 CommandManager

Durch die neuen Funktionalitäten (XML-Signatur und Generierung aus den Package-Files) ist die Anzahl der Befehlsausführungen im SWID-Generator stark gewachsen. Aus diesem Grund wurde eine Klasse erstellt, welche diese Ausführungen übernimmt. Dadurch ergaben sich die Vorteile, dass eine zentrale Stelle für Systembefehle entstand und das Mocking der Funktionen einfacher realisiert werden konnte.

Abbildung 6.6: CommandManager

6.6 Requirements check

Damit die gesamte Funktionalität des SWID-Generators genutzt werden kann, müssen auf dem System die nötigen Programme installiert sein. Wenn eines dieser Tools fehlt, kann der SWID-Tag nicht oder nicht vollständig generiert werden. Aus diesem Grund wurde beim Programmstart ein Requirements-Checker eingebaut, welcher überprüft ob die definierten Tools installiert sind. Fehlt das benötigte Tool, wird die gewünschte Aktion abgebrochen und eine Fehlermeldung ausgegeben.

Nachfolgend ein Beispiel für eine Fehlerausgabe beim Signieren des SWID-Tags.

```
root@debian:/home/swid# swid_generator swid --package dpkg --pkcs12 swidgen.pfx
usage: swid_generator swid [-h] [--env {auto,dpkg,pacman,rpm}]
     [--doc-separator DOCUMENT_SEPARATOR]
     [--full] [--pretty] [--hierarchic]
     [--pkcs12-pwd PKCS12_PWD]
     [--software-id SOFTWARE-ID | --package PACKAGE | --package-file FILE_PATH]
     [--evidence PATH] [--name NAME]
     [--version-string VERSION] [--new-root PATH]
swid_generator swid: error: Please install following packages: xmlsec1
```

Dieser Requirements-Check kommt hauptsächlich beim Signieren und bei der Generierung aus den Paket-Dateien zum Einsatz. Diese Funktionalität war nicht Bestandteil dieser Arbeit. Um dem Benutzer viel Ärger zu ersparen und die Fehlerquellen zu minimieren, wurde dies im Verlauf des Projekts als notwendig eingestuft.
Auf den Environment-Klassen ist es nun möglich, eine Liste der benötigten Tools zu definieren. Bei einem Methodenaufruf wird überprüft, ob die dort definierten Programme installiert sind. Folgend werden die Listen im *RpmEnvironment* aufgezeigt:

```
class RpmEnvironment(CommonEnvironment):

  required_packages_for_package_file_method = [
     "rpm2cpio",
     "cpio"
  ]

  required_packages_for_sign_method = [
     "xmlsec1"
  ]

  # Implementation ...
```

6.7 Temporäre Ablage

Für das Extrahieren der Informationen aus den Paket-Dateien müssen die enthaltenen Dateien auf dem System vorhanden sein. Aus diesem Grund wurde entschieden, dass die Dateien temporär entpackt und nach der Generierung wieder gelöscht werden. Der Vorgang wurde wie folgt definiert:

1. Temporärer Ordner im "*/tmp/*"-Ordner erstellen
   Ordnername: "*/tmp/swid_*****" (5-stellige zufällige Zeichensequenz)
2. Alle nötigen Dateien in diesen temporären Ordner entpacken
3. SWID-Tag erstellen und ausgeben
4. Alle Ordner mit Präfix "*swid_*" im Ordner "*/tmp*" löschen

Der "*/tmp/*"-Ordner wurde gewählt, da bei einem Systemneustart eine automatische Säuberung vollzogen wird und somit von der Generierung zurückgebliebene Ordner entfernt werden.

Garbage collection Um die temporär abgelegten Dateien zu löschen, wird nach der erfolgreichen Ausgabe der SWID-Tags eine Säuberung durchgeführt. Diese beinhaltet folgende Schritte:

1. Alle "*swid_**"-Ordner im "*/tmp/*"-Ordner in einer Liste sammeln
2. Listeneinträge abarbeiten und die Ordner löschen

6.8 Signatur

Wie in Abschnitt 4 *Analyse* bereits erläutert, bietet der SWID-Generator die optionale Möglichkeit, SWID-Tags digital zu signieren. Aufgerufen werden kann dies mit dem Argument `--pkcs12 DATEI`, wobei als `DATEI` ein *pkcs12*-Container angegeben werden muss. Dieser Container muss sowohl den privaten Schlüssel, das entsprechende Zertifikat und optional Zertifikate der Chain of Trust enthalten. Falls die Datei mit einem Passwort verschlüsselt ist, muss zusätzlich das Passwort mit dem Argument `--pkcs12-pwd PASSWORT` übergeben werden.
Implementiert wurde die Signatur mit dem Linux Konsolen Programm "*xmlsec1*". Dies fügt allerdings selbst keine XML-Tags hinzu, sondern setzt voraus, dass diese bereits enthalten sind und lediglich befüllt werden müssen. Dazu wurde ein Template erstellt, welches automatisch vor der Signierung hinzugefügt wird.
Mittels diesem Template erfolgt auch die Konfiguration der Signatur. Unter anderem wird darin bestimmt, welche Algorithmen zur Erstellung der Signatur und des Digests verwendet werden sollen.

```
...
  <SignatureMethod Algorithm="www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
...
  <DigestMethod Algorithm="www.w3.org/2001/04/xmlenc#sha256"/>
...
```

Die Ausgabe des signierten XMLs erfolgt über das stdout des Systems. Dadurch kann er mittels dem `CommandManager` aufgerufen werden. In Python erfolgt der Aufruf wie folgt:

```
folder_info = create_temp_folder(...)
path = folder_info['save_location'] + '/swid_tag.xml'
open(path, 'wb').write(plain_xml)
   command = ["xmlsec1", "--sign", "--pkcs12", "PKCS12","--pwd", "PASS", path]
signed_xml = CommandManager.run_command_check_output(command, cwd=os.getcwd())
```

Da `xmlsec1` lediglich Dateien auf dem Filesystem signieren kann, muss erst ein temporärer Order erstellt und das XML in eine Datei geschrieben werden.

Der finale XML-Output beinhaltet neu den Namespace "*dsig*" und den `<dsig:Signature>`-Tag, welcher alle nötigen Informationen beinhaltet um die Gültigkeit des Dokuments zu verifizieren. Der Output sieht wie folgt aus:

```
<?xml version="1.0" encoding="utf-8"?>
<SoftwareIdentity ... xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" ... >
...
 <dsig:Signature>
  <dsig:SignedInfo>
   <dsig:CanonicalizationMethod
     Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
   <dsig:SignatureMethod
     Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
     <dsig:Reference>
      <dsig:Transforms>
      <dsig:Transform
        Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
      <dsig:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
     </dsig:Transforms>
     <dsig:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
     <dsig:DigestValue>E6EkvftLCObo..</dsig:DigestValue>
   </dsig:Reference>
  </dsig:SignedInfo>
  <dsig:SignatureValue>
     LDt8xaEkmHnlGE6uXFVPYZ3AwDZZMq2mIXbetQ38fEupcuUSNNnqg4lAxDtZ5fr1
     ...
     FC5Sdd9B6BNomSaHxgk9jQ==
  </dsig:SignatureValue>
  <dsig:KeyInfo>
   <dsig:X509Data>
     <dsig:X509Certificate>
       MIIF5jCCA86gAwIBAgIQapvhODv/K2ufAdXZuKdSVjANBgkqhkiG9w0BAQwFADCB
       ...
       6gH+HF8TIhCnH3+zzWuDN0Rk6h9KVkfKehI=
     </dsig:X509Certificate>
   </dsig:X509Data>
  </dsig:KeyInfo>
 </dsig:Signature>
</SoftwareIdentity>
```

6.9 Testing

Die Testabdeckung des Programmcodes war zu Beginn dieser Arbeit bei ca. 50%. Um diese zu erhöhen, wurde das gesamte Testing-Environment umstrukturiert und zusätzlich zu Unit-Tests, Integration-Tests eingeführt. Im folgenden Abschnitt wird erklärt, wie die Testabdeckung auf 80% erhöht werden konnte und wie die Integration-Tests im Build-Prozess eingebunden wurden.

6.9.1 Testplan

Bei einem Push auf das Repository werden auf dem Build-Server insgesamt 5 Build-Jobs (für jede Python-Version einzeln) gestartet. Innerhalb dieser Jobs werden zuerst die Unit-Tests und anschliessend die Integration-Tests für jede Distribution durchgeführt. Nachfolgend der gesamte Build-Prozess graphisch dargestellt:

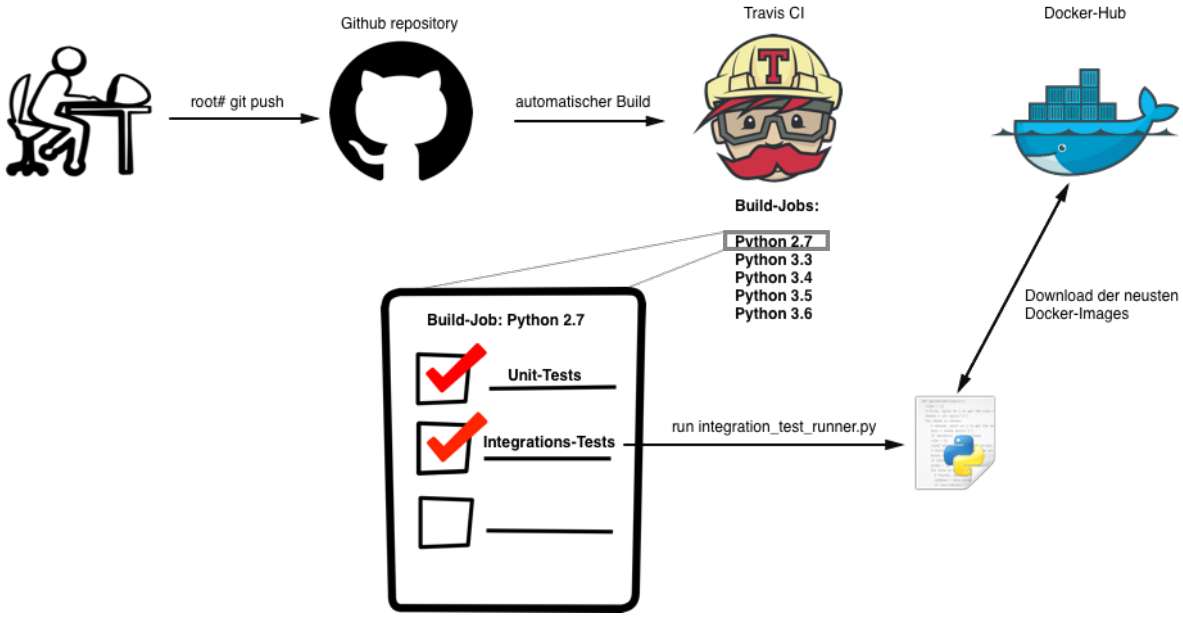

Abbildung 6.7: Graphische Darstellung des Build-Prozesses

6.9.2 Unit-Tests

Das anfänglich verwendete Test-Framework *Pytest* wurde im Verlauf der Arbeit durch die Standard-Test-Bibliothek ersetzt. Denn trotz weiter Verbreitung bot *Pytest* nicht alle benötigten Funktionalitäten. Unter anderem haben die Funktionen `setUp()` und `tearDown()` gefehlt, welche für das geplante Mocking essentiell waren.
Demzufolge entstand folgender Aufbau der Tests:

```
class TestClass(unittest.TestCase):
  def setUp(self):
    # Mocks initialisieren
  def tearDown(self):
    # Mocks abbauen
  def test_method(self):
    # Testen auf Basis der Mocks
```

6.9.3 Mocking

Damit die Test-Methoden den richtigen Input bekommen, werden im Test-Environment die Ausgaben der Systembefehle von einer Mock-Klasse zurückgegeben. Das Mocking-Framework realisiert diesen Workflow, indem bei der Ausführung eines Systembefehls die Methodenaufrufe zu der Mock-Klasse umgeleitet werden. Innerhalb dieser Mock-Klasse können die Systembefehle unterschieden und die richtigen Ausgaben zurückgegeben werden. Nachfolgend der Aufbau der Testumgebung.

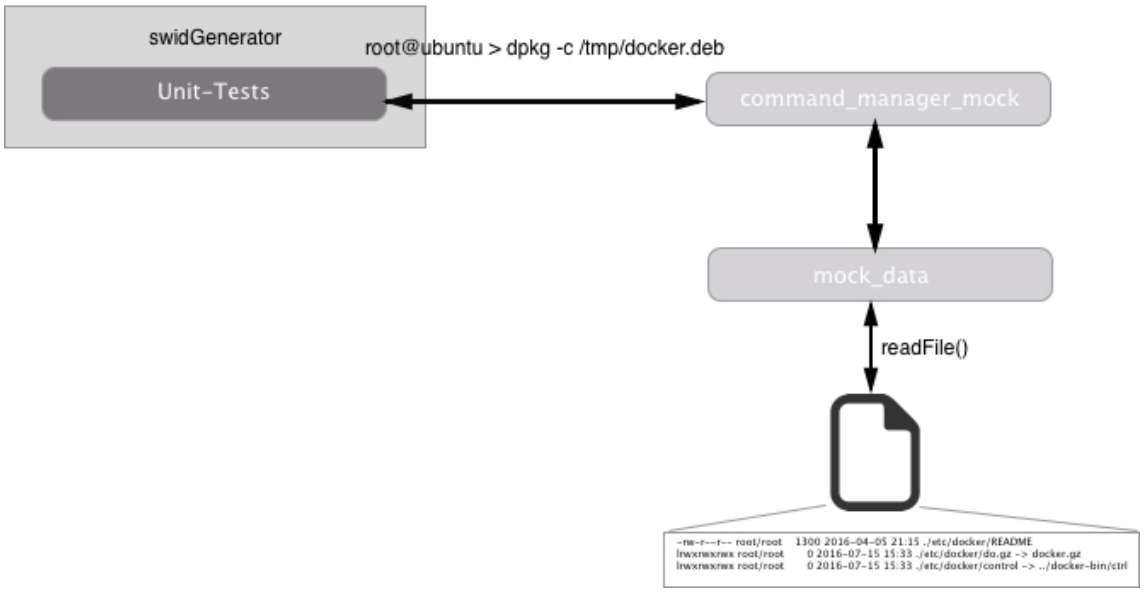

Abbildung 6.8: Workflow CommandMangerMock

Die Mock-Klasse besitzt den Namen `CommandManagerMock` und dient, wie vorhin erwähnt, nur zur Rückgabe der richtigen Ausgaben. Um diese übersichtlich zu strukturieren, wurden die Rückgabewerte der Systembefehle in Dateien ausgelagert. Die Unterscheidung der Systembefehle ist in folgendem Code-Abschnitt ersichtlich:

```
class CommandManagerMock(object):
    @staticmethod
    def run_command_check_output(command_argumentlist):
        if command_argumentlist == ['dpkg', '/tmp/docker.deb', '-c']:
            return mock_data.dpkg_query_package_list_output
```

6.9.4 Docker Integration Tests

Neben Unit-Tests, sollte das Programm zusätzlich als eine komplette Komponente getestet werden. Aus diesem Grund wurde das aktuelle Test-Environment mit Integration-Tests erweitert. Die Schwierigkeit bei der Ausführung dieser Tests sind die unterschiedlichen Distributionen. Damit das Programm auf all diesen getestet werden kann, wurde die Ausführung der Tests in Docker-Container ausgelagert. Für jede Distribution (Debian, Redhat und Arch Linux) wurde ein Docker-Container erstellt, welcher direkt beim Aufruf die Tests ausführt.

Build-Prozess Diese Tests wurden vollumfänglich im Build-Prozess integriert. Um immer die aktuellen Docker-Container zu verwenden, werden vor den Tests jeweils die aktuellsten Docker-Images von DockerHub[9] heruntergeladen.

IntegrationTestRunner Für den Start der Tests wurde ein Skript erstellt, welches die Ausführung so einfach wie möglich gestaltet und konfigurierbar ist. Nachfolgend ein Beispiel für die Ausführung des IntegrationTestRunner:

```
# Ausschnitt aus der Datei .travis.yml
language:  python
install:
  - pip install tox
script:
 - tox
 - if [ $TOXENV != "cov" ];
    then
     echo 'Start Integration-Tests in Docker:';
     python integration_test_runner.py `echo ${PWD}` $TOXENV dpkg pacman rpm; # Ausführung
   fi
```

Parameterliste des *integration_test_runner*-Skripts:

1. Parameter: Pfad zum SourceCode-Ordner
2. Parameter: Python-Version der Testausführung. (z. B py27, py33, etc.)
3. Parameter: Liste mit den Environments, welche getestet werden sollten Whitespace-getrennte Liste von Environment-Bezeichnungen. (z. B dpkg rpm pacman)

[9] DockerHub: https://hub.docker.com/r/davidedegiorgio/swidgenerator-dockerimages/

IntegrationTestRunnerConfiguration Um den IntegrationTestRunner beliebig konfigurieren zu können, wurde eine zusätzliche Konfigurationsklasse erstellt. Eine Instanz dieser Klasse muss bei der Instanziierung mitgegeben werden. Folgende drei Punkte sind konfigurierbar:

1. Docker-Images: Environment-Bezeichnung und Image-Name (Dictionary)
2. Testing-Files: Liste der Dateien welche getestet werden sollten
3. Ordner-Mapping: Mapping des SourceCode-Folders und des lokalen Ordners im Docker-Container

```
# Ordner-Mapping
working_directory_docker = "/home/swid"
# Environment-Bezeichnung und Image-Name
docker_image_names = [
 {"environment": "dpkg", "image": "swidgenerator-dockerimages:debian"},
 {"environment": "rpm", "image": "swidgenerator-dockerimages:redhat"},
 {"environment": "pacman", "image": "swidgenerator-dockerimages:archlinux"}
]
# Liste der Dateien welche getestet werden sollten.
test_files = ['tests/integration_test.py']
# Define Configuration for IntegrationTestRunner
configuration = IntegrationTestRunnerConfiguration(working_directory_docker, docker_image_names,
test_files)
```

Dockerfiles Damit die Integration-Tests auf einer optimalen Umgebung ausgeführt werden, wurde für jede Distribution ein Dockerfile erstellt. Auf Basis dieser werden die Dockercontainer gebaut und gestartet. Der Aufbau sieht wie folgt aus:

1. Basis definieren
2. Dependencies herunterladen
3. Python-Interpreter installieren
4. Locales-Einstellungen vornehmen

Die Abhängigkeiten wurden jeweils mit den Paketmanagern der Distributionen installiert. Lediglich die Installation der Python Interpreter wurde separat vorgenommen. Hierfür wurde auf den Distributionen Debian und Redhat, das Konsolenprogramm *pyenv* verwendet. Denn dies vereinfacht die Verwaltung und Installation. Im Arch Linux-Container mussten die Interpreter aus dem *AUR* [10] heruntergeladen werden. Um auch hier die Installation zu vereinfachen und in Zukunft problemlos weitere Versionen installiert zu können, wurde das Programm *yaourt (Yet Another User Repository Tool)* verwendet.

6.9.5 Testergebnisse

Im Vorher-Nachher-Vergleich wurde festgestellt, dass die Test-Abdeckung stark gestiegen ist. Durch diesen Aufbau wurde eine Code-Abdeckung von ca. 80% erreicht, was beinahe eine 30%-ige Steigerung darstellt.

Abbildung 6.9: Test-Abdeckung nach der Umstrukturierung

[10] Archlinux User Repository: https://aur.archlinux.org/

6.10 Error handling

Mit dem Requirements-Checker wurde bereits eine kleine Anzahl an möglichen Fehlern abgefangen. Um aber ein vollumfängliches Error-Handling zu erreichen, wurden zusätzlich die nachfolgend aufgelisteten Punkte beachtet.

Dateiübergabe Dem Generator können verschiedene Dateien übergeben werden (z.B. Zertifikat beim Signieren). Um hier die Fehler zu minimieren, wird bei der Übergabe der Dateien eine Existenzprüfung durchgeführt. Wird eine Datei übergeben, startet der ArgumentParser automatisch die Prüfung und gibt bei einem Fehler die entsprechende Fehlermeldung aus.

```
usage: swid_generator swid [-h] [--env {auto,dpkg,pacman,rpm}]
  [--doc-separator DOCUMENT_SEPARATOR]
  [--regid REGID] [--entity-name ENTITY_NAME]
  [--full] [--pretty] [--hierarchic]
  [--hash HASH_ALGORITHMS] [--pkcs12 PKCS12]
  [--pkcs12-pwd PKCS12_PWD]
  [--software-id SOFTWARE-ID | --package PACKAGE | --package-file FILE_PATH]
  [--evidence PATH] [--name NAME]
  [--version-string VERSION] [--new-root PATH]
Error: argument --package-file: The file '/tmp/test_package.deb' does not exist
```

Argumentübergabe Um die Korrektheit aller übergebenen Argumente sicherzustellen, werden diese bei der Entgegennahme überprüft. An verschiedenen Stellen wurden die möglichen Übergabeparameter limitiert (z.B Auswahl des Hash-Algorithmus).

Beispiel: Übergabe der Hash-Algorithmen
Hier wird geprüft, ob der übergebene Wert dem Schema entspricht. Als Basis dient eine Regex-Prüfung, welche den String analysiert. Erlaubt sind entweder einzelne Einträge: "*sha256*", "*sha384*", "*sha512*" oder eine Liste von Algorithmen: "*sha256,sha384,sha512*".

Systembefehle Wie bereits erwähnt wurde, führt der SWID-Generator eine Vielzahl von Systembefehlen aus. Treten bei einer solchen Ausführung Fehler auf, muss der Benutzer über die Ursache informiert und die Generierung abgebrochen werden.
Für diesen Fall wurden bei der Ausführung im CommandManager, alle möglichen Exceptions und Errors abgefangen. Tritt ein Fehler auf, wird nicht die explizite Exception delegiert, sondern eine Instanz der Klasse `CommandManagerError`. Somit kann beim Aufruf der `create_swid_tags()`-Methode nur diese Expection auftreten. Dies macht den Code übersichtlicher, da im `try: except:` nur die *CommandManagerErrors* erwartet werden müssen.

UnicodeErrors Bei der Interpretation von speziellen Zeichen (z.B. ğ, y, ı, c) aus dem `stdout` des Systems treten in den meisten Methoden Fehler auf. Die Ursache ist die Umwandlung zum Unicode-Format. Aus diesem Grund wurden alle möglichen *Unicode Errors* (UnicodeEncodeError, UnicodeDecodeError und UnicodeError) über die gesamte Generierung der SWID-Tags abgefangen. Beim Eintreten eines solchen Fehlers wird eine spezielle Fehlermeldung ausgegeben und der Benutzer wird aufgefordert, die *Locales*-Einstellungen richtig zu setzen und die Konsolenausgabe UTF-8 kompatibel einzustellen.

OSError Durch die `--evidence`-Funktion ist es möglich, SWID-Tags basierend auf Ordnerstrukturen generieren zu lassen. Aus diesem Grund können Berechtigungs oder andere Betriebssystem-Fehler auftreten. Diese gehören zu der Kategorie *OSError* und treten nur auf, wenn das darunterliegende Betriebssystem auf einen Fehler trifft.

Gloabales Error handling Da während der gesamten Generierung und Print-Phase Fehler auftreten können, werden diese Phasen in einem `try: except:` ausgeführt. Nachfolgend eine Veranschaulichung des Programmcodes:

```
try:
 swid_tags = create_swid_tags()
 print_swid_tags()
except CommandManagerError as e:
 print(e)
 sys.exit(1)
except (UnicodeEncodeError, UnicodeDecodeError, UnicodeError):
 print(unicode_error_message)
 sys.exit(1)
except OSError as e:
 print(e)
 sys.exit(1)
```

Error return codes Die Liste der Error return codes wurde um die folgenden zwei Einträge erweitert.

- **Error code 4**: An internal error has occured.
- **Error code 5**: An external command has thrown an error.

6.11 Unicode_patch [15]

Der SWID-Generator muss mit internationalen Paketen und somit mit Sonderzeichen anderer Sprachen (z.B. ğ, ı, c) umgehen können. In den verschiedenen unterstützten Python-Version gibt es allerdings keine einheitliche Handhabung dieser Zeichen.

Da Python 2 als Standardcodierung *ASCII* verwendet, existiert für die Verwendung des *UTF-8* Zeichensatzes die spezifische Klasse *Unicode*. Unter Python 3 wurde die Standardcodierung auf *UTF-8* geändert und die Klasse *Unicode* somit entfernt. Um *UTF-8* in beiden Versionen unterstützen zu können muss eine Unterscheidung im Code erfolgen.

Dies wurde mit der Unicode_patch Methode gelöst, welche an allen kritischen Stellen verwendet wird.

6.12 Systemanforderungen

Damit die gesamte Funktionalität des SWID-Generators genutzt werden kann, müssen neben einem unterstützten Python-Interpreter andere Tools auf dem System installiert werden. Zudem müssen die *Locales*-Einstellungen richtig konfiguriert und die UTF-8 Kompatibilität des `stdout` sichergestellt werden. Nachfolgend werden diese Punkte noch im Detail erläutert:

Python-Interpreter Wie im `README` des Repositories erläutert, werden die Python-Interpreter: 2.7, 3.3, 3.4, 3.5 und 3.6 unterstützt. Das Programm wurde auf allen Versionen erfolgreich getestet.

--package-file Für diese Funktion werden auf den verschiedenen Distribution unterschiedliche Tools benötigt.

Distribution	Benötigte Tools
Debian	tar, ar
Redhat	rpm2cpio, cpio
Archlinux	tar

Tabelle 6.9: Benötigten Tools pro Distribution

--pkcs12 Für diese Funktion wird das Tool *xmlsec1* verwendet.

Locales Damit die SWID-Tags vollständig generiert werden, wird empfohlen die *Locales*-Einstellungen auf dem Linux-System richtig zu konfigurieren. Dies bedeutet, dass die Konsolenausgabe UTF-8 unterstützen und die `$LANG` Umgebungsvariable entsprechend gesetzt werden muss. Nachfolgend eine Liste der Befehle, welche auf den einzelnen Distributionen ausgeführt werden müssen.

Distribution	Befehlsfolge
Debian	1. apt-get install locales 2. locale-gen en_US.UTF-8 3. update-locale LANG=en_US.UTF-8 4. export LANG=en_US.UTF-8
Redhat	1. export LANG=en_US.UTF-8
Arch Linux	1. echo "en_US.UTF-8 UTF-8" »/etc/locale.gen 2. locale-gen 3. export LANG=en_US.UTF-8

Tabelle 6.10: Befehlen für die *Locales*-Einstellungen

7 Ausblick

7.1 Travis-CI Build stages

Durch das Hinzufügen der Integration-Tests zum Build-Prozess ist die Dauer des Builds gestiegen, welcher nun etwa 6-8 Minuten benötigt. Dies ist im Vergleich zur Vorgängerversion um einiges länger. Grund für diesen Anstieg ist der Download der Images.
Am 11. Mai 2017 stellte Travis-CI eine neue Funktionalität vor - *Build stages*. Diese erlauben es innerhalb der Stages verschiedene Jobs zu definieren und parallel auszuführen.
Die Stages haben durch Ihren Aufbau die Möglichkeiten untereinander einen gemeinsamen Cache zu nutzen. Dadurch können z.B. Docker-Images vor den Tests heruntergeladen und gestartet werden. Die Jobs in den Stages haben dann die Möglichkeit die Docker-Images aus dem Cache zu nutzen.
Somit wird die Build-Dauer stark verringert, da die Docker-Images nicht vor jedem Test heruntergeladen werden müssen. Leider ist diese Funktionalität sehr spät im Projekt vorgestellt worden und konnte deshalb im Rahmen dieser Arbeit nicht umgesetzt werden.

7.2 Garbage collection

Nach einigen Analysen, wurde eine mögliche Verbesserung bei der Garbage-Collection (löschen der "*swid_****"-Ordner im "*/tmp/*"-Ordner) eruiert. Aktuell ist der SWID-Generator so implementiert, dass nach der gesamten Ausgabe alle Ordner mit dem Präfix "*swid_*" im "*/tmp/*"-Ordner gelöscht werden. Dies ist in einigen Szenarien nicht optimal.

Beispielszenario: Parallele Ausführung des Generators
Wenn der Generator in mehreren Prozessen gleichzeitig ausgeführt wird, löscht der früher terminierte Prozess die temporär abgelegten Dateien der anderen Prozesse. Dies würde zu einem Absturz der Programme führen.

Um diesem Problem entgegenzuwirken, müsste die Garbage-Collection umstrukturiert werden. Der Generator könnte eine Liste, mit den von ihm erstellten Ordnern führen und die Säuberung auf dieser Liste basierend, erledigen. Somit würden nicht alle Ordner mit dem Präfix "*swid_****" gelöscht werden, sondern nur die Ordner aus der Liste.

7.3 Logging

Wegen dem Wachstum des gesamten Programms ist die Implementation eines Loggers zu empfehlen. Nicht nur die ausgeführten Operationen sollten protokolliert werden, sondern auch Informationen zu Fehlern und Abstürzen. Zudem sollten bei Exceptions und Errors der gesamte Stacktrace ins Protokoll eingetragen werden.

7.4 Parallelisierung

Die Erstellung der SWID-Tags erfolgt zurzeit sequenziell. Die Liste der Pakete wird ausgelesen und in der gegebenen Reihenfolge abgearbeitet. Um diesen Vorgang zu beschleunigen, könnte die Erstellung der SWID-Tags parallel ausgeführt werden.
Nach erfolgreicher Abfrage der Liste könnte für jeden Eintrag ein Subprozess zur Erstellung des SWID-Tags gestartet werden und nach erfolgreicher Generierung die Ausgabe erfolgen.
Da die Reihenfolge der SWID-Tags für den Server irrelevant ist, könnte diese Funktionalität durchaus implementiert werden. Die Python Standard-Bibliothek stellt eine Vielzahl von Funktionen und Features im Bereich Multi-Threading zur Verfügung.

7.5 Docker layers

Nach einigen Recherchen zur Docker-Architektur wurde eine Möglichkeit entdeckt die Images zu verkleinern und so die Buildtime zusätzlich zu optimieren.
Die Docker-Images sind im Schichten-Prinzip aufgebaut. Für jeden Command im Dockerfile wird eine Schicht erzeugt. Dieses Prinzip ist nützlich, weil die Schichten im Cache gespeichert werden können. Wenn die Reihenfolge der Befehle nicht verändert wird, müssen nur die neu hinzugefügten ausgeführt werden. Nachfolgend ein Beispiel:

```
# Ausschnitt aus Dockerfile
FROM centos:centos6
# layer 0
RUN yum -y update
# layer 1
RUN yum -y install php
# layer 2
RUN yum -y install apache2
```

Wenn nun ein Command hinzugefügt wird, wird nicht das gesamte Dockerfile neu gebaut, sondern nur die Differenz (alles nach Layer 2).

```
# Ausschnitt aus Dockerfile
FROM centos:centos6
# layer 0
RUN yum -y update
# layer 1
RUN yum -y install php
# layer 2
RUN yum -y install apache2
# layer 3, UPDATE: hinzufügen eines Commands (Bis layer 2 ist alles im Cache)
RUN yum -y install mysql
```

Durch das Zusammenhängen von möglichst vielen Commands, kann der Build durch das Caching beschleunigt werden. Laut dem Online-Artikel von Manuel Vacelet [13] wird auf diese Art und Weise der Cache optimal genutzt. Es wird nicht nur der Build beschleunigt sondern auch die Grösse der Images wird kleiner. Dies würde den aktuellen Build-Prozess wie gewünscht optimieren.

8 Rückblick

8.1 Performance Profiling

Der SWID-Generator wurde im Laufe der Arbeit stark angepasst und um neue Funktionen erweitert. Daher wurde die Performance der Applikation neu gemessen.

Die Messungen wurden in den Docker-Containern der Integration-Tests durchgeführt. Durch die unterschiedliche Installation der Systeme ist ein direkter Vergleich nicht möglich. Aber es bietet dennoch gewisse Werte um die Laufzeit abzuschätzen. Um möglichst akkurate Messwerte zu erhalten, wurden alle Tests dreimal durchgeführt und der Median ermittelt.

- *DPKG*, 271 installierte Pakete, 512 Paketdateien
- *Pacman*, 232 installierte Pakete, 222 Paketdateien
- *RPM*, 326 installierte Pakete, 257 Paketdateien

Unter allen Systemen wurden die gleichen Operationen ausgeführt, allerdings mussten sie zum Teil leicht modifiziert werden um dem entsprechenden Systemen zu entsprechen. Mit folgenden Parametern wurde der SWID-Generator aufgerufen.

ids	`software-id`
tags	`swid`
tags-pretty	`swid --pretty`
tags-pretty-full	`swid --full --pretty`
tags-full-sha256	`swid --full`
tags-full-sha384	`swid --full --hash sha384`
tags-full-sha512	`swid --full --hash sha512`
tags-full-shaAll	`swid --full --hash sha256,sha384,sha512`
tags-full-pkcs12	`swid --full --pkcs12`
tags-full-pkg	`swid --full --package xmlsec`
tags-full-pkg-file	`swid --full --package-file docker.pkg.tar.xz`
tags-full-evidence	`swid --full --evidence "/home/swid/swid_generator"`
tags-full-pkg-file-All	`swid --full --package-file *.tar.xz`

	dpkg	pacman	rpm
ids	0.225s	0.278s	0.405s
tags	0.289s	0.388s	0.513
tags-pretty	0.345s	0.444s	0.624s
tags-pretty-full	9.563s	18.607	74.458s
tags-full-sha256	7.999s	14.551s	69.596s
tags-full-sha384	6.686s	12.336s	63.159s
tags-full-sha512	6.779s	12.296s	62.983s
tags-full-shaAll	12.161s	28.352s	75.899s
tags-full-pkcs12	10.534s	18.510s	81.237s
tags-full-pkg	0.272s	0.301s	0.600s
tags-full-pkg-file	0.270s	1.979s	2.003s
tags-full-evidence	0.249s	0.263s	0.215s
tags-full-pkg-file-All	177.986s	112.527s	211.634s

Tabelle 8.11: Ergebnisse Performance-Tests

8.2 Qualitätssicherung

Mittels den im Abschnitt 5 definierten Massnahmen zur Qualitätssicherung, wurde die Qualität während des gesamten Projekts stetig überprüft.
Durch dies konnte erreicht werden, dass alle überwachten Merkmale auf einem hohen Niveau geblieben sind. Nachfolgend werden die einzelnen Auswertungen zum Ende der Arbeit aufgeführt.

End Situation

Testing-Coverage (Coveralls)	79%
Code-Qualität (Landscape)	99%
Letzter Build (Travis-CI)	erfolgreich

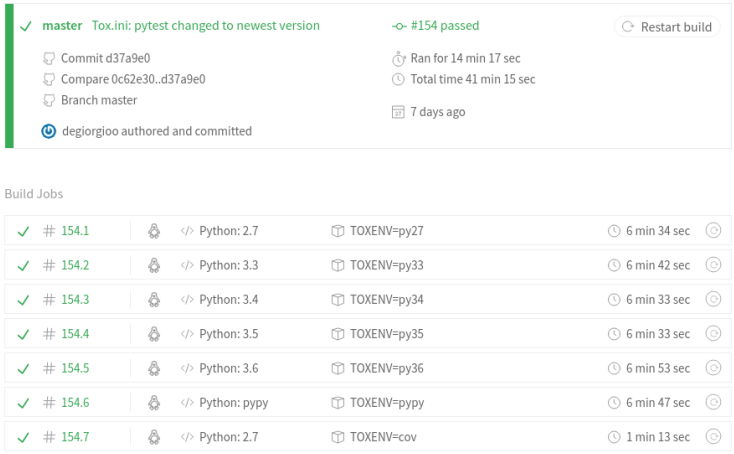

Abbildung 8.10: Letzter Build-Status

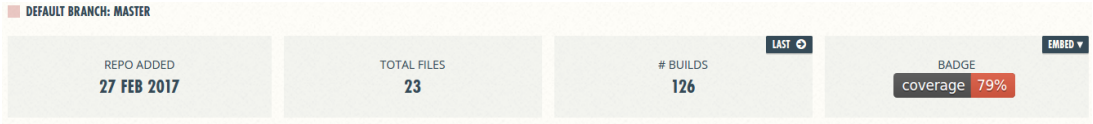

Abbildung 8.11: Test-Coverage

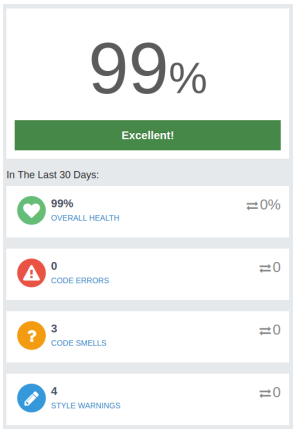

Abbildung 8.12: Landscape-Qualität

8.3 Zeitauswertung

Wie im Appendix A *Projektplan* beschrieben, wurde der Aufwand für das gesamte Projekt mit 720 Stunden angenommen. Effektiv aufgewendet wurden bis zur Zeit der Projektabgabe 707 Stunden was einer Abweichung von nur etwa zwei Prozent entspricht. Nachfolgend wird die aufgewendete Zeit genauer analysiert und die einzelnen Wochen und Phasen dargestellt.

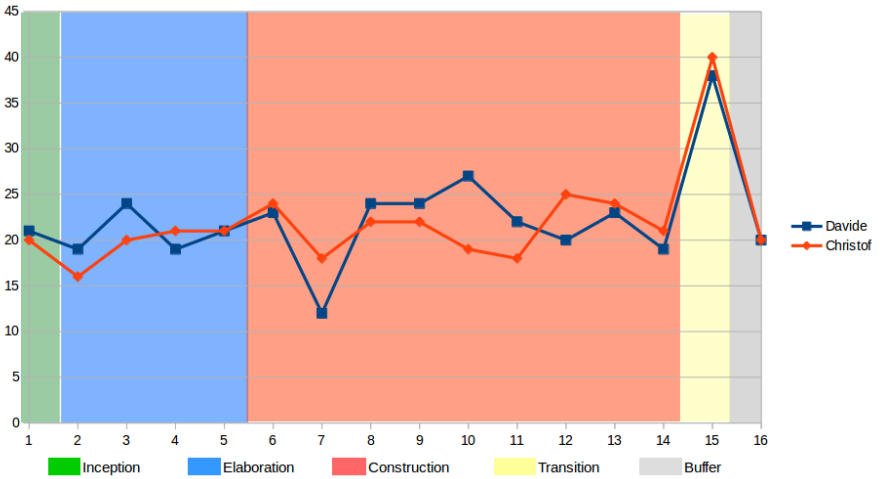

Abbildung 8.13: Zeitauswertung nach Wochen

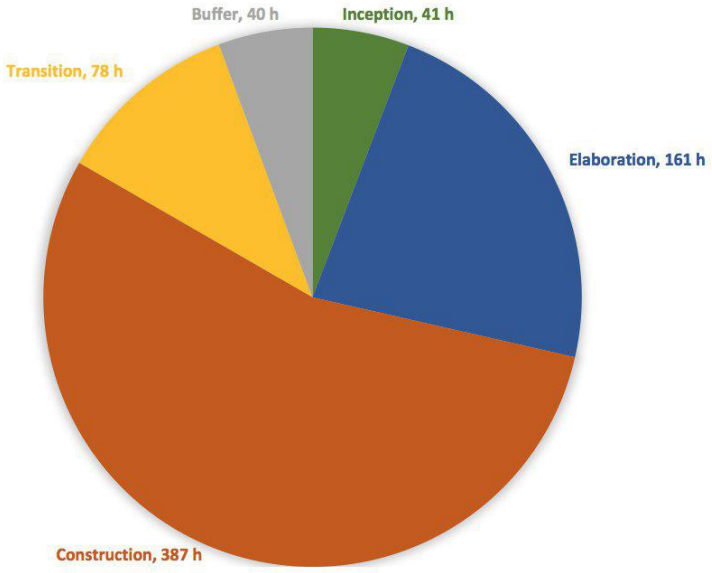

Abbildung 8.14: Zeitauswertung nach Phasen

8.4 Reflexion

8.4.1 Davide De Giorgio

Diese Arbeit war nun das zweite Projekt, welches ich mit Christof Greiner umgesetzt habe. Schon am Anfang merkte ich, dass wir gut ins Projekt einsteigen werden. Die normalen "Einstiegshürden", welche normalerweise in einem Team auftreten, waren in diesem Projekt kaum bis gar nicht vorhanden. Wir kannten uns gegenseitig schon sehr gut und wussten demnach auch die Stärken und Schwächen des Gegenübers.

Zum ersten Mal in meiner Informatiker-Karriere hatte ich das Vergnügen, mich in einen ISO-Standard einzuarbeiten. Erst jetzt wurde mir klar, wie tief ins Detail die gesamte Informatik-Welt standardisiert und wie wichtig das Einhalten der Regeln und Konventionen ist. Weil die gesamte Arbeit auf diesem Standard beruhte, waren wir gezwungen, sehr genau zu arbeiten und keine Entscheidungen voreilig zu treffen. Rückblickend bin ich sehr froh, diese Erfahrung gemacht zu haben, weil ich gelernt habe, Entscheidungen vorher genauestens zu überdenken und mich davor ins Thema einzuarbeiten.

Zusätzlich zum Ganzen ist die Programmiersprache Python dazugekommen. Wir als Team hatten nicht viel Erfahrung mit dieser Sprache und mussten diese im Voraus erlernen. Auch dies wurde meiner Meinung nach von beiden sehr gut gelöst. Wir haben vor Beginn der Projektarbeit die Ferienzeit genutzt und individuell die Sprache und die dazugehörenden Konzepte erlernt. Weil die Arbeit auch sehr Linux-basiert war, hatte ich die Möglichkeit, mich näher mit dem Unix-Betriebssystem zu befassen, was mich in meiner beruflichen Karriere sicherlich weiterbringen wird.

Im Grossen und Ganzen fand ich das Projekt sehr spannend. Ich konnte vieles erlernen und mein bisheriges Wissen noch zusätzlich vertiefen. Ausserdem bin ich froh, dass wir dieses Projekt erfolgreich abschliessen und den Dozenten zufriedenstellen konnten. Ich hoffe, dass ich und Christof Greiner diese Erfahrungen mitnehmen und den Rucksack mit diesem Projekt erfolgreich schliessen können.

8.4.2 Christof Greiner

Meiner Ansicht nach ist die Projektarbeit durchwegs gelungen. Es gab etliche Hürden, welche aber alle durch gute Planung, Zusammenarbeit und mit Unterstützung unseres Betreuers Andreas Steffen gemeistert werden konnten.

Nach der Studienarbeit, welche zum Teil noch ziemlich chaotisch ablief, hat sich unsere Projektplanung und Zeitschätzungen bereits ziemlich verbessert. Wir konnten alle Meilensteine fristgerecht erreichen und alle geplanten Funktionalitäten umsetzen. Durch das Continuouse Delivery habe ich gemerkt, wie motivierend es sein kann, wenn man immer eine funktionierende Software hat, welche in kurzen Abständen nach und nach wächst.

Eine weitere wichtige Erfahrung für mich war es, ein Projekt ohne Vorwissen in diesem Bereich durchzuführen. Die Einarbeitung in die Thematik war ein grosser Teil der Arbeit und es erwies sich nicht immer als einfach den Überblick in den verschiedenen ISO, RFC und NISTIR Standards zu behalten. Da es noch keine Referenzimplementationen gab, auf welche man zurückgreifen konnte, war es umso wichtiger die Standards zu verstehen und richtig umzusetzen.

Neben den themaspezifischen Erfahrungen konnte ich auch meine Linux- und Python-Kenntnisse erweitern, welche sich zu Beginn der Arbeit auf Basiswissen beschränkten. Nach und nach hat man durch "Learning by doing" immer mehr Erfahrung gesammelt und konnte sich so stets verbessern.

9 Glossar

Abkürzung	Beschreibung
ASCII	American Standard Code for Information Interchange
AUR	Archlinux User Repository
CI	Continuous integration
Docker	Software-Container Platform. Bietet Möglichkeit an, Umgebung (Docker-Container) für Softwares aufzubauen.
DPKG	Software-Basis für den Package-Manager von Debian-basierten Systemen. Installiert, löscht Debian Packages und stellt Informationen zu den Packages aus.
Garbage collection	Aufräumarbeiten
ICE	International Electrotechnical Commission
ISO	International Organization for Standardization
NIST	National Institute of Standards and Technology
NISTIR	NIST Internal or Interagency Reports
pacman	Zentraler Paketmanager von Archlinux-Systemen
pyenv	Python-Version Management Tool
patch	Software-Korrektur
RPM	Paketverwaltungs-System für Redhat basierte Systeme
stdout	Standardausgabe
strongSwan	OpenSource IPsec basierte VPN Lösung
strongTNC	Trusted Network Connect ("Sicherer Netzwerkzugriff") Server von strongSwan
SWID	Software Identification
UC	Use-Case
unicode	Internationaler Standard, welcher Schriftzeichen oder Textelement aller bekannten Schriftkulturen und Zeichensysteme in einem digitalem Code festlegt.
UTF-8	8-Bit UCS Transformation Format
W3C	World Wide Web Consortium, Gremium zur Standardisierung der Techniken im World Wide Web
XML	Extensible Markup Language
yaourt	Yet another User Repository Tool

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[1] ISO/IEC 19770-2:2009 *Software asset management - Part 2: Software identification tag*, ISO/IEC JTC 1/SC 7, 2009-11

[2] ISO/IEC 19770-2:2015 *Software asset management - Part 2: Software identification tag*, ISO/IEC JTC 1/SC 7, 2015-10

[3] NISTIR8060, *Guidelines for the Creation of Interoperable Software Identification (SWID) Tags*, 2016-04

[4] D.Bargen, C.Fässler, J.Furrer *«Endpoint Compliance Monitoring based on Software Identification Tags»* Bachelor's Thesis, HSR Hochschule für Technik Rapperswil, 2014, (URL: http://eprints.hsr.ch/id/eprint/367)

[5] T.Berners-Lee, R.Fielding, L.Masinter *Uniform Resource Identifier (URI)*, RFC 3986, 2005 (URL: https://www.ietf.org/rfc/rfc3986.txt)

[6] W.Fischer, *Rpm und dpkg Kommandos*, (URL: https://www.thomas-krenn.com/de/wiki/Rpm_und_dpkg_Kommandos), besucht am 13.04.2017

[7] *Pacman/Rosetta Kommandos*, (URL: https://wiki.archlinux.org/index.php/Pacman/Rosetta), besucht am 13.04.2017

[8] M.Bartel et al. *«XML Signature Syntax and Processing»*, 2002-02-12 (URL: https://www.w3.org/TR/2002/REC-xmldsig-core-20020212/)

[9] M.Bartel et al. *«XML Signature Syntax and Processing Version 1.1»*, 2013-04-11 (URL: https://www.w3.org/TR/xmldsig-core1/)

[10] R.Irani, *Docker Tutorial Series: Writing a Dockerfile*, (URL: https://rominirani.com/docker-tutorial-series-writing-a-dockerfile-ce5746617cd), besucht am 16.05.2017

[11] Ernesti & Kaiser, *Python 3 - Das Umfassende Handbuch*, Rheinwerk Computing, 2015, ISBN:978-3-8362-3633-1

[12] Eric Matthes, *Python Crash Course: A Hands-On, Project-Based Introduction to Programming*, 2016, ISBN: 978-1-5932-7603-4

[13] Manuel Vancelet, *Docker layers cost*, (URL: https://medium.com/@vaceletm/docker-layers-cost-b28cb13cb627), besucht am 10.06.2017

[14] Aleksey Sanin et al. *Documentation xmlsec1 Library*, (URL: https://www.aleksey.com/xmlsec/index.html), besucht am 10.05.2017

[15] Armin Ronacher, *Porting to Python 3 Redux* (URL: http://lucumr.pocoo.org/2013/5/21/porting-to-python-3-redux/), besucht am 03.05.2017

III Appendix

A Projektplan

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

Projektplan
SWID Generator

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrssemester 2017

Autoren:	Christof Greiner
	Davide De Giorgio
Betreuer:	Prof. Dr. Andreas Steffen
Projektpartner:	INS – Institute for Networked Solutions

Änderungsgeschichte

Datum	Version	Änderung	Autor
22.02.2017	1.0	Erstellung Dokument	Davide De Giorgio
27.02.2017	1.1	Projektplanung	Davide De Giorgio & Christof Greiner
09.03.2017	1.2	Anpassung Zeitplanung	Christof Greiner
10.04.2017	1.3	Redmine und ScrumBoard ersetzt	Christof Greiner
22.05.2017	1.4	Ergänzung mit zusätzlichen Arbeitspaketen, Risiken aktualisiert	Christof Greiner

Inhalt

1 Zeitliche Planung

In diesem Abschnitt wird die zeitliche Planung und das Vorgehen erläutert.

1.1 Projektübersicht

Das Projekt wird in den Grundzügen nach RUP geführt und hat insgesamt vier Phasen: Inception, Elaboration, Construction und Transition. Diese sind in Iterationen unterteilt, welche jeweils eine Woche lang sind. Als Ausnahme wir die Construction Phase nach SCRUM geführt um möglichst agil und flexibel vorgehen zu können.

Das gesamte Projekt wird mit 16 Wochen geplant. Die Einteilung nach RUP wurde wie folgend geplant und beinhaltet 3 Meilensteine.

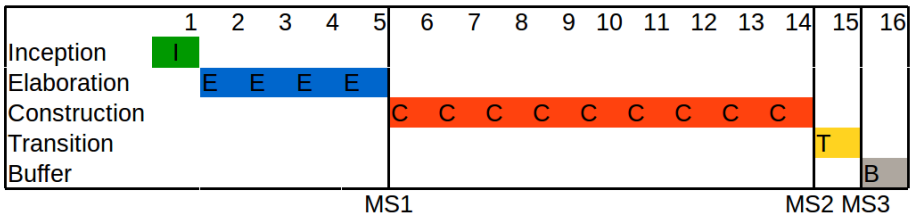

Figure 1.1: Phasen Übersicht

1.2 Wochenpensum

Laut Reglement und den zugeschriebenen ECTS-Punkten ist jeder Student verpflichtet insgesamt 360h zu investieren. Es wird angenommen, dass in der 15. und 16. Woche die doppelte Anzahl Stunden aufgewendet werden kann, da sich diese ausserhalb der Unterrichtszeit befindet. Dadurch ergeben sich pro Student für die ersten 14 Wochen ein Wochenpensum von 20 Stunden und in den letzten beiden Wochen ein Pensum von 40 Stunden.

1.3 Iterationen

Die Dauer einer Iterationen ist eine Woche á 20 Stunden. Zu Beginn jeder Iteration wird besprochen welche Arbeitspakete abgearbeitet werden.
Bei der Definition dieser Arbeitspakete wird die Arbeitszeit geschätzt, was bei der Planung der Iterationen hilfreich ist. Bei der Auswahl der Arbeitspakete fliesst auch die erstellte Risikomatrix ein.

1.4 Puffer

Mithilfe der Risikomatrix und den bisherigen Projekterfahrungen wurde die Pufferzeit auf 10% des gesamten Projekts veranschlagt, sprich 72 Stunden bzw. 36 Stunden pro Student. Diese Zeit wird verwendet falls es zu einem Zeitpunkt im Projekt Schwierigkeiten geben sollte und die Einhaltung der Meilensteine in Gefahr kommt.

1.5 Meilensteine

In diesem Abschnitt werden die Meilensteine und die fertigzustellenden Punkte definiert.

1.5.1 Meilenstein 1 - End of Elab - 26.03.2017

- Projektplan fertiggestellt
- Informationen gesammelt.
  NIST Guidelines for the Creation of Interoperable SWID Tags.
  ISO/IEC 19770-2: Part 2: Software identification tag
  Draft Coffin SACM
  Alte Bachelor Arbeit: Endpoint Compliance Monitoring based on Software Identification Tags
- Toolchain
  Entwicklungsumgebung (lokal) definiert und installiert
  Entwicklungsumgebung (Aktuelle Situation, Stand 2014) analysiert
  Entwicklungsumgebung (Build-Environment) aufgesetzt
  Redmine für Projektmanagement aufgesetzt
  Toolchain im Griff.
- IST und SOLL-Situation
  Aktueller Stand des SWID-Generators analysiert (Aufbau SWID-Tags und Funktionalität)
  SOLL-Situation der SWID-Tags definiert. Basierend auf ISO Standard und NIST Guidelines beachtet.
- Prototyp erstellt
  Hashgenerierung
  Konfigurationsdateien erkennen

1.5.2 Meilenstein 2 - End of Construction - 04.06.2017

- Abschluss aller definierten Arbeitspakete.
  Update des SWID-Generator auf aktuellen ISO Standard
  Ergänzen der SWID-Tags mit File-Hashes
  Veränderliche Dateien speziell markieren
  Optional: Pacman Paketmanager auch anbieten
  Optional: Signierung der SWID-Tags mit Zertifikat
  Optional: Evidence Tag aus Filesystem

1.5.3 Meilenstein 3 - Dokumentation & Präsentation - 11.06.2017

- Pull Request in strongSwan Repository
- Deployment in PyPI
- Die Dokumentation wurde überarbeitet und fertiggestellt.
- Plakat fertiggestellt.
- Präsentation fertiggestellt.

2 Vorgehen

In diesem Abschnitt wird erläuter, wie während dem Projekt in den verschiedenen Bereichen vorgegangen wird und welche Arbeitsweise oder Hilfsmittel dafür verwendet werden.

2.1 Iterationen planen

Elaboration In der Elaboration-Phase werden die Arbeitspakete, welche in der Inception-Phase definiert wurden, den 4 Phasen direkt zugeteilt.

Construction Die Arbeitspakete der Construction-Phase (Funktionalitäten), werden vor Beginn definiert und der Aufwand geschätzt (Kann mit dem Backlog in Scrum verglichen werden). Am ersten Tag der Iteration, wird dann jeweils besprochen welche Arbeitspakete in die Iterationen genommen werden und den Stand der letzten Iteration angeschaut.

2.2 Hilfsmittel

Toggle [1] Für das Zeitmanagement wird Toggle verwendet. Dabei handelt es sich um ein kompaktes Programm welches automatisch die aufgewendete Zeit aufzeichnet. So können zu Schluss des Projektes Auswertungen gemacht werden.

Trello [2] Um möglichst flexibel zu sein und von überall gleichzeitig arbeiten zu können, wurde das zu Begin physische Scrum-Board nach Trello migriert.

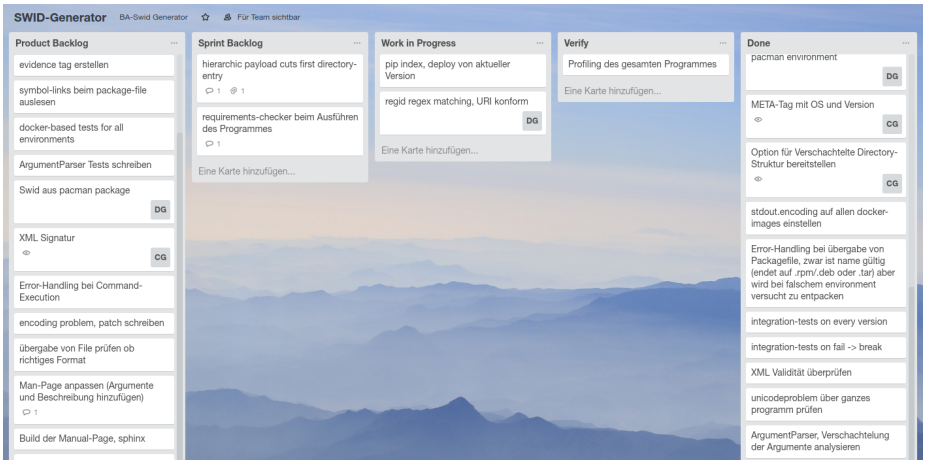

Figure 2.2: Trello Ausschnitt

[1] https://toggle.com
[2] https://trello.com

3 Risiken

3.1 Risikomatrix

Beschreibung	Wahrschein.	max. Schaden	Vorbeugung
Toolchain	5%	20h	Vor der Construction-Phase Toolchain absprechen und im Griff haben.
Python Probleme	10%	40h	Python lernen und gute Nachschlagewerke suchen
Probleme mit File-Hash Generierung	5%	20h	Verschiedene Möglichkeiten und Alternativen mit Betreuer besprechen
Erkennung von Konfigurationsdateien	20%	20h	Detaillierte Evaluation der einzelnen Package-Managern und Plan B Szenario erarbeiten

B Urheberrechtsvereinbarung

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

Vereinbarung

1. Gegenstand der Vereinbarung

Mit dieser Vereinbarung werden die Rechte über die Verwendung und die Weiterentwicklung der Ergebnisse der Bachelorarbeit «SWID Generator for Software Identification Tags» von Christof Greiner und Davide De Giorgio unter der Betreuung von Prof. Dr. Andreas Steffen geregelt.

2. Urheberrecht

Die Urheberrechte stehen den Studenten zu.

3. Verwendung

Die Ergebnisse der Arbeit dürfen sowohl von den Studenten wie von der HSR nach Abschluss der Arbeit verwendet und weiterentwickelt werden

Rapperswil, den 31.05.2017 .....

Rapperswil, den 31.05.2017 .....
Die Studenten

Rapperswil, den 31.05.2017 .....
Der Betreuer der Bachelorarbeit

Rapperswil, den 1. Juni 2017 .....
Der Studiengangleiter

C Sitzungsprotokolle

Sitzungsprotokoll - Woche 1

Mittwoch, 22 Februar 2017
09:00 Uhr

Projekt: **BA - SWID Generator**
Woche: 1

Sitzungsteilnehmer

- Andreas Steffen
- Davide De Giorgio
- Christof Greiner

Traktanden:

- Arbeitsauftrag
- Terminabsprache
- Einführung in SWID Generator
- Abgabe Standards (ISO, NIST, RFC)

Beschlüsse (Diskussion):

- Arbeitsauftrag wird am nächsten Meeting abgegeben
- Meetings finden bis auf weiteres jeden Mittwoch um 09:00 Uhr statt
- Wir sind Frei in Bezug auf die Wahl der Entwicklungsumgebung und Buildprozess
- Ansprechpersonen im Institut sind:
  - Andreas Steffen
  - Tobias Brunner
  - Christian Fässler
- Minimum Anforderung ist „dpkg“ und „rpm“ Paketmanager
- Optimal Generierung der SWID-Tags mit Hash direkt aus Paket ohne Installation
- Verschiedene Varianten prüfen
- Konzentration auf Primary Tag

Offene Punkte (erledigt vor nächster Sitzung):

Was	Verantwortlichkeit
Projektplan erstellen	cgreiner, ddegiorg
Arbeitsauftrag erstellen	asteffen

Kommende Abwesenheiten:

Kürzel	Von	Bis	Grund
keine			

Nächster Termin:

Datum: 01.03.2017
Zeit: 09:00 Uhr
Dauer: 1h

Sitzungsprotokoll - Woche 2

Mittwoch, 01 März 2017
09:00 Uhr

Projekt: **BA - SWID Generator**

Sitzungsteilnehmer

- Andreas Steffen
- Davide De Giorgio
- Christof Greiner

Traktanden:

- Arbeitsauftrag
- Projektplan
- Diverse Fragen zu Arbeit

Beschlüsse (Diskussion):

- Arbeitsauftrag noch nicht ganz fertig → wird am nächsten Meeting abgegeben
- Projektsplan überarbeiten
  - Projekt mit 16 statt 15 Wochen planen
  - Risiko 1 & 2 zu generisch → löschen
  - INS Logo löschen, da fusioniert mit anderem Institut
  - Probleme mit Configfiles in Risikomatrix aufnehmen
- Flag suchen für dynamische Files (Configs)
- Patch-Tags

Offene Punkte (erledigt vor nächster Sitzung):

Was	Verantwortlichkeit
Beschlüsse umsetzen	cgreiner, ddegiorg
Arbeitsauftrag fertigstellen	asteffen

Kommende Abwesenheiten:

Kürzel	Von	Bis	Grund
keine			

Nächster Termin:

Datum: 08.03.2017
Zeit: 09:00 Uhr
Dauer: 1h

Sitzungsprotokoll - Woche 3

Mittwoch, 08 März 2017
09:00 Uhr

Projekt: **BA - SWID Generator**

Sitzungsteilnehmer

- Andreas Steffen
- Davide De Giorgio
- Christof Greiner

Traktanden:

- Arbeitsauftrag
- Projektplan
- Aktueller Stand
  - Struktur SWID Tags
  - Erkennung Config-Files
  - Definierung Reg-ID und Tag-ID
  - Generierung direkt aus Package möglich
  - Redmine aufgesetzt/eingerichtet
  - Code-Analyse und Prototyp begonnen
- Offene Fragen
  - StrongSwan/stronTNC Testumgebung sinnvoll?
  - Wie mit Programmversionierung fortfahren?
  - Payload und Evidence verwenden oder auf eines beschränken?

Beschlüsse (Diskussion):

- Testumgebung nicht sinnvoll, da eine Standalone Software und Schnittstellen definiert.
- Versionierung erst am Ende wenn in Main Git Repo eingefügt.
- Nur Payload Tag verwenden
- @n8060mutable nur einfügen wenn nicht default
- Language für StrongSwan nicht nötig, evtl. optional für andere Benutzer hinzufügen.
- Signatur bereitet evtl. Probleme mit Namespaces → genauer anschauen
- Zusätzliche UseCases zu alter BA definieren
- End of Elaboration wird um eine Woche verschoben

Offene Punkte (erledigt vor nächster Sitzung):

Was	Verantwortlichkeit
Git der Dokumentation an asteffen senden	cgreiner, ddegiorg
Dokumentation für End of Elab mit nötigen Artefakten vervollständigen	Cgreiner, ddegiorg

Kommende Abwesenheiten:

Kürzel	Von	Bis	Grund
asteffen		19.03.2017	Geschäftsreise

Nächster Termin:

Datum: 22.03.2017
Zeit: 09:00 Uhr
Dauer: 1h

Sitzungsprotokoll - Woche 5

Mittwoch, 22 März 2017
09:00 Uhr

Projekt: **BA - SWID Generator**

Sitzungsteilnehmer

- Andreas Steffen
- Davide De Giorgio
- Christof Greiner

Traktanden:

- Feedback Dokumentation
- Aktueller Stand
  - Prototyp
    - DPKG Environment Infos auslesen
    - Direktory/File Struktur
    - SHA-Hashes
  - Dokumentation ausgebaut
- Offene Fragen
  - Default Hash-Algorithmus
- Scrumboard Online

Beschlüsse (Diskussion):

- Feedback Doku
  - z.T. Durcheinander mit ISO Standard Final oder Draft (z.B. in UseCases)
  - auf korrekte Schreibweise von stronSwan achten (camelCase)
  - z.T. NIST geschrieben anstatt NISTIR
- Feedback Prototyp
  - Wenn Standardkonform keine Verschachtelung von Directories
- Offene Fragen
  - Als default wenn kein Hash-Algorithmus angegeben wird → SHA256

Offene Punkte (erledigt vor nächster Sitzung):

Was	Verantwortlichkeit
Einen fertigen SWID-Tag an asteffen senden	cgreiner, ddegiorg

Kommende Abwesenheiten:

Kürzel	Von	Bis	Grund
-	-	-	-

Nächster Termin:

Datum: 29.03.2017
Zeit: 09:00 Uhr
Dauer: 1h

Sitzungsprotokoll - Woche 6

Mittwoch, 29 März 2017
09:00 Uhr

Projekt: **BA - SWID Generator**

Sitzungsteilnehmer

- Andreas Steffen
- Davide De Giorgio
- Christof Greiner

Traktanden:

- Aktueller Stand
  - Direktory-Struktur angepasst
  - File Hashing fertig
  - Environments rpm und pacman fertig
  - Test/Mocks anpassen noch in Arbeit

Beschlüsse (Diskussion):

- Erste lauffähige Version bis nächste Woche zeigen können
- Verschachtelung von Dateien sind übersichtlicher und der strongTNC Parser kann nun doch damit umgehen.
  → Verschachtelung evtl. wieder implementieren und als Option mit Argument anbieten
- Gedanken zum Download von massenweisen Packages (.deb/.rpm/pacman) machen und evtl mit einem Script ausführen
- Weil Informationen zum OS und Architektur nur in der UniqueID prüfen ob Möglichkeit besteht dies in einem separaten Meta-Tag zu platzieren. Unter anderem auch um Kompatibilität mit anderen Generatoren zu gewährleisten welche ein anderes Format in der UniqueID haben.

Offene Punkte (erledigt vor nächster Sitzung):

Was	Verantwortlichkeit
Lauffähige Version des SwidGenerators zeigen	cgreiner, ddegiorg

Kommende Abwesenheiten:

Kürzel	Von	Bis	Grund
-	-	-	-

Nächster Termin:

Datum: 05.04.2017
Zeit: 09:00 Uhr
Dauer: 1h

Sitzungsprotokoll - Woche 7

Mittwoch, 05 April 2017
09:00 Uhr

Projekt: **BA - SWID Generator**

Sitzungsteilnehmer

- Andreas Steffen
- Davide De Giorgio
- Christof Greiner

Traktanden:

- Stand Prototyp (produktiv)
  - Meta tag hinzugefügt
  - Directory Struktur ohne Verschachtelung → bei nächstem Stand optional möglich
  - Regex für modifizierbare RegId deckt fast alle URIs ab

- Stand Projekt
  - Swid aus .deb fast fertig
  - Tests laufen alle wieder und wurden erweitert

- Offene Fragen:
  - Files aus .deb müssen entpackt werden für Hash: ist „/tmp/swid[rand_id]/..“ in Ordnung dafür? Dieser wird bei einem Reboot gelöscht
  - Hashing von gewissen Files braucht root Rechte: wie damit umgehen?

Beschlüsse (Diskussion):

- /tmp/.. Ordner ist in Ordung, evtl. könnte Ramdisk angeschaut werden
- Ausführung des Swid-Generators mit root ist in Ordnung
- Diverse kleine Änderungen im SWID-Tag
  - Metatag product ohne Bindeglieder (- oder _)
  - xmlns:SHA-384 → falsche URL hinterlegt
  - uniqueID in tagId ändern
- Anschauen was mit Files ist, welche direkt im Root Verzeichniss sind

Offene Punkte (erledigt vor nächster Sitzung):

Was	Verantwortlichkeit
Wenn Änderungen deployed, Link zu Repo an asteffen senden	cgreiner, ddegiorg

Kommende Abwesenheiten:

Kürzel	Von	Bis	Grund
asteffen	10.04.17	23.04.17	Geschäftsreise

Nächster Termin:

Datum: 26.04.2017
Zeit: 09:00 Uhr
Dauer: 1h

Sitzungsprotokoll - Woche 10

Mittwoch, 26 April 2017
14:30 Uhr

Projekt: **BA - SWID Generator**

Sitzungsteilnehmer

- Andreas Steffen
- Davide De Giorgio
- Christof Greiner

Traktanden:

- Stand Applikation (produktiv)
  - Tag Generierung aus File ist möglich, verschiedene Package Formate (deb/rpm/tar.xz) werden unterstützt
  - Symbollink hashing sind implementiert, werden am Sonntag in Master gemerged
  - Zwischenprofiling wurde durchgeführt um Performance zu evaluieren

- Stand Projekt
  - Doku wurde weitergeschrieben (Inhalt definiert und befüllt)
  - Tests um Tntegration Tests mit Docker ergänzt
  - Probleme mit Signatur (Keine gültige Signatur)

- Offene Fragen:
    - Wer ist Gegenleser unserer Gruppe?
    - Wann soll der Endstand deployed (git, pip, etc) werden?

Beschlüsse (Diskussion):

- Gegenleser ist Jürg Jucker, und Zwischenpräsentation soll in den nächsten zwei Wochen erfolgen. Am besten zur Zeit des Wochenmeetings wenn möglich.
- End Deployement in ca. zwei Wochen in Zusammenarbeit mit Tobias Brunner.

Offene Punkte (erledigt vor nächster Sitzung):

Was	Verantwortlichkeit
Termin für Zwischenpräsentation mit Jürg Jucker planen.	cgreiner, ddegiorg
Terminplanung mit Tobias machen für Deployement des SWID-Generators	cgreiner, ddegiorg

Kommende Abwesenheiten:

Kürzel	Von	Bis	Grund
-			

Nächster Termin:

Datum: 03.05.2017
Zeit: 09:00 Uhr
Dauer: 1h

Sitzungsprotokoll - Woche 11

Mittwoch, 03 Mai 2017
09:00 Uhr

Projekt: **BA - SWID Generator**

Sitzungsteilnehmer

- Andreas Steffen
- Davide De Giorgio
- Christof Greiner

Traktanden:

- Stand Applikation
  - Signatur implementiert mit xmlsec1 (ext. Tool) nimmt pkcs12 container und Passwort entgegen
  - Symbol-Links gefixed

- ToDo's nächste Woche:
  - Test Coverage erhöhen (Unit & Integration)
  - Refactorings
  - Error-Handlich implementieren

- Offene Fragen:
  - Da alle Punkte der Aufgabenstellung erledigt: Gibt es evtl. noch weitere Usecases oder optionale Aufgabenstellungen welche im Rest der Construction Phase erledigt werden können?

Beschlüsse (Diskussion):

- Erstellung von SWID Tags aus Ordnerstrukturen ist möglicher zusätzlicher Usecase.

Offene Punkte (erledigt vor nächster Sitzung):

Was	Verantwortlichkeit
Zusätzliche Aufgabnstellungen überlegen/formulieren	asteffen

Kommende Abwesenheiten:

Kürzel	Von	Bis	Grund
-			

Nächster Termin:

Datum: 10.05.2017
Zeit: 09:00 Uhr
Dauer: 1h

Sitzungsprotokoll - Woche 12

Mittwoch, 10 Mai 2017
09:00 Uhr

Projekt: **BA - SWID Generator**

Sitzungsteilnehmer

- Andreas Steffen
- Davide De Giorgio
- Christof Greiner

Traktanden:

- Stand Applikation
  - Code teilweise refactored
  - Testcoverage erhöht und Integrationtest mit Docker erweitert
- Zusätzliche Aufgabenstellung besprechen
- Zwischenpräsentation mit Herr Jucker am 16.5.2017 um 11:00 Uhr

Beschlüsse (Diskussion):

- Erstellen eines Evidence Tags aus Ordnern auf dem Dateisystem wird in die Aufgabenstellung aufgenommen.

Offene Punkte (erledigt vor nächster Sitzung):

Was	Verantwortlichkeit

Kommende Abwesenheiten:

Kürzel	Von	Bis	Grund
-			

Nächster Termin:

Datum: 17.05.2017
Zeit: 09:00 Uhr
Dauer: 1h

Sitzungsprotokoll - Woche 13

Mittwoch, 17 Mai 2017
09:00 Uhr

Projekt: **BA - SWID Generator**

Sitzungsteilnehmer

- Andreas Steffen
- Davide De Giorgio
- Christof Greiner

Traktanden:

- Stand Applikation
  - Evidence Tag fertig, ausser kleiner Bug
  - Python 3.5/3.6 neu supported
  - Unicode/ASCII Problem → console muss UTF-8 unterstützen
- Stand Projekt
  - Meeting mit Herr Jucker durchgeführt
- Offene Fragen
  - Darf Python 2.6 aus Liste unterstützter Versionen gestrichen werden?

Beschlüsse (Diskussion):

- Python 2.6 darf gestrichen werden
- strongSwan Release anfang Juni, bis dann sollte SWID-Generator deployed sein.
- Deployment mit Tobias Brunner nächste Woche, genauer Termin wird noch vereinbart

Offene Punkte (erledigt vor nächster Sitzung):

Was	Verantwortlichkeit

Kommende Abwesenheiten:

Kürzel	Von	Bis	Grund
-			

Nächster Termin:

Datum: 31.05.2017
Zeit: 09:00 Uhr
Dauer: 1h

Sitzungsprotokoll - Woche 14

Mittwoch, 31. Mai 2017
15:00 Uhr

Projekt: **BA - SWID Generator**

Sitzungsteilnehmer

- Andreas Steffen
- Davide De Giorgio
- Christof Greiner

Traktanden:

- Stand Applikation
  - End Of Construction erreicht
  - Finaler Stand in Master gemerged
  - Deployment am Donnerstag 01.062017

- ToDo's nächste Woche
  - Dokumentation nachführen

Beschlüsse (Diskussion):

- keine

Offene Punkte (erledigt vor nächster Sitzung):

Was	Verantwortlichkeit
Deployment mit tbrunner	Cgreiner, ddegiorg

Kommende Abwesenheiten:

Kürzel	Von	Bis	Grund
-			

Nächster Termin:

Datum: 07.06.2017
Zeit: 09:00 Uhr
Dauer: 1h

Sitzungsprotokoll - Woche 15

Mittwoch, 07. Juni 2017
09:00 Uhr

Projekt: **BA - SWID Generator**

Sitzungsteilnehmer

- Andreas Steffen
- Davide De Giorgio
- Christof Greiner

Traktanden:

- Stand Projekt
  - Dokumentation, Dokumentaion, Dokumentation
  - Pull-Request durchgeführt

- Offene Fragen
  - Termin Endpräsentation?

Beschlüsse (Diskussion):

- keine

Offene Punkte (erledigt vor nächster Sitzung):

Was	Verantwortlichkeit
Termin Endpräsentation bestimmen	asteffen

Kommende Abwesenheiten:

Kürzel	Von	Bis	Grund
-			

Nächster Termin:

-