**Bachelorarbeit**

im Studiengang  
Informatik

vorgelegt von

**Ferhat Özmen**Matr.-Nr.: 008 204 835

am 29. August 2016   
an der Hochschule Bochum

Erstprüfer/in: Prof. Dr. rer. nat. Rainer Lütticke  
Zweitprüfer/in: M.Sc. Clemens A. Schulz

Implementierung einer Grafischen Benutzeroberfläche für die Konfiguration der Browser in the Box Security Policy

# Kurzfassung

Eine virtuelle Surfumgebung in Zeiten von Web 2.0, ist eine großartige Lösung um sicher und uneingeschränkt im Internet zu surfen. Durch die ständige Weiterentwicklung der Webtechnologien, entstehen immer mehr Webseiten, die nicht nur statisch aufgebaut sind, sondern aktiven gestaltet wurden, wie zum Beispiel mit Java Script. Dadurch ergeben sich allerdings auch neue und gefährliche Angriffsflächen die von Hackern ausgenutzt werden, um das System zu infizieren, welches den Browser ausführt. Selbst Antiviren Software bieten nicht mehr ausreichenden Schutz, da so genannte 0-Day Sicherheitslücken ausgenutzt werden können, die für die Antiviren Software unbekannt sind. Die Signatur von solchen Angriffen zu erkennen, ist deshalb nur sehr schwer möglich. Mit Browser in the Box wird genau dieses Problem in Angriff genommen, um bietet als Lösung eine virtuelle Surfumgebung an, in der ein gekapselter Browser zum Surfen im Internet genutzt wird. Selbst wenn ein Angriff stattfinden sollte, geht dieser ins Leere, da das Betriebssystem auf dem der Browser läuft, eine virtuelle Maschine ist. Doch hatte der Benutzer bislang keine Möglichkeit währen der Browser Sitzung, die Einstellungen der Security Policies zu ändern. Mit dem Ansatz des Security Policy Managers, soll diese Problem gelöst werden. Der Benutzer soll eine Anwendung bekommen, welches während der Browser Sitzung aufgerufen werden kann um die gewünschten Einstellungen vorzunehmen.

Inhaltsverzeichnis

[Kurzfassung 2](#_Toc442344325)

[Abbildungsverzeichnis 6](#_Toc442344326)

[Tabellenverzeichnis 6](#_Toc442344327)

[Abkürzungsverzeichnis 7](#_Toc442344328)

[Vorwort 8](#_Toc442344329)

[1 Einleitung 9](#_Toc442344330)

[1.1 Motivation 9](#_Toc442344331)

[1.2 Zielsetzung 9](#_Toc442344332)

[2 Grundlagen und Begriffsdefnition 10](#_Toc442344333)

[2.1 Endpoint Security 10](#_Toc442344334)

[2.1.1 Begriffe und Konventionen 10](#_Toc442344335)

[2.2 IT - Security Policy 10](#_Toc442344336)

[2.2.1 Definition 10](#_Toc442344337)

[2.2.2 Bedeutung für Software Entwickler 10](#_Toc442344338)

[2.2.3 Bedeutung für Software Anwender 10](#_Toc442344339)

[2.2.4 Aufbau einer Security Policy (Nach was?? Sirrix/SANS/BSI -> Clemens oder Norbert fragen) 10](#_Toc442344340)

[2.2.5 Typen von IT – Security Policies 10](#_Toc442344341)

[2.3 Browser in the Box 11](#_Toc442344342)

[2.4 Vorstellung 11](#_Toc442344343)

[2.4.1 Standalone 11](#_Toc442344344)

[2.4.2 Managed 11](#_Toc442344345)

[2.4.3 Bestehende Möglichkeit zur Konfiguration 11](#_Toc442344346)

[2.5 Technologien 11](#_Toc442344347)

[2.5.1 Virtual Box 11](#_Toc442344348)

[2.5.2 Python 11](#_Toc442344349)

[2.5.3 Abhängigkeiten 11](#_Toc442344350)

[2.6 Software Design 11](#_Toc442344351)

[2.6.1 Architektur 11](#_Toc442344352)

[2.6.2 Plattform 11](#_Toc442344353)

[2.6.3 Security Policy 11](#_Toc442344354)

[3 Implenetierung 12](#_Toc442344355)

[3.1 Grafische Benutzeroberfläche 12](#_Toc442344356)

[3.1.1 Details der Implementierung (Technisch) 12](#_Toc442344357)

[3.1.2 Beurteilung (Wieso letztendlich mit PySide e.t.c) 12](#_Toc442344358)

[3.2 Primäre Ziel der Implementierung 12](#_Toc442344359)

[3.3 Umsetzung (Wie die neue Konfiguration letztendlich umgesetzt wurde und warum e.t.c) 12](#_Toc442344360)

[4 Ergebnis 13](#_Toc442344361)

[4.1 Beispiele 13](#_Toc442344362)

[4.2 Benutzerfreundlichkeit 13](#_Toc442344363)

[4.2.1 Vorher 13](#_Toc442344364)

[4.2.2 Nachher 13](#_Toc442344365)

[5 Zusammenfassung 14](#_Toc442344366)

[6 Zusammenfassung und Ausblick 15](#_Toc442344367)

[Literaturverzeichnis 16](#_Toc442344368)

[Eidesstattliche Versicherung 17](#_Toc442344369)

[Auszug aus dem Strafgesetzbuch (StGB) 17](#_Toc442344370)

# Abbildungsverzeichnis

[Abbildung 1: Sichere Zwischenablage 23](#_Toc460220471)

[Abbildung 2: Sicherer Download 25](#_Toc460220472)

[Abbildung 3: Sicheres Uploaden 26](#_Toc460220473)

[Abbildung 4: Sicheres Drucken 26](#_Toc460220474)

[Abbildung 5: Sicheres speichern der Persistenten Daten 28](#_Toc460220475)

[Abbildung 6: Tray App Einstellungen 29](#_Toc460220476)

[Abbildung 7: BitBox Steuerbefehle 29](#_Toc460220477)

[Abbildung 8: Security Policy Manager – Information flow 38](#_Toc460220478)

[Abbildung 9: Security Policy Manager – Netzwerk 39](#_Toc460220479)

[Abbildung 10: Horizontales Layout Statischer Proxy 41](#_Toc460220480)

[Abbildung 11: Import von QtCore und QtGui 43](#_Toc460220481)

[Abbildung 12: UML - Ui\_MainWindow Klasse 43](#_Toc460220482)

[Abbildung 13: Ui\_MainWindow Klasse 44](#_Toc460220483)

[Abbildung 14: createComponents() Funktion – Elemente für die DNS Einstellung 45](#_Toc460220484)

[Abbildung 15: Central Widget 46](#_Toc460220485)

[Abbildung 16: createLayouts() Funktion - Hinzufügen der Komponenten, in ein Horizontales Layout 48](#_Toc460220486)

[Abbildung 17: de.po Datei für die deutsche Lokalisierung 49](#_Toc460220487)

[Abbildung 18: retranslateUi() Funktion - Strings mit gettext Wrapper-Funktion 49](#_Toc460220488)

[Abbildung 19: UML - Security Policy Manager 51](#_Toc460220489)

[Abbildung 20: if-Abfrage von \_\_name\_\_ 52](#_Toc460220490)

[Abbildung 21: main() Funktion innerhalb des policytool.py Moduls 53](#_Toc460220491)

[Abbildung 22: Signal und Slot Beispiel 54](#_Toc460220492)

[Abbildung 23: Ui\_MainWindow Klasse Initialisierung 55](#_Toc460220493)

[Abbildung 24: Dynamisches erzeugen von Buttons 56](#_Toc460220494)

[Abbildung 25: Registry Pfade der Security Policy Einstellungen 56](#_Toc460220495)

[Abbildung 26: Prüfen ob BitBox installiert wurde 57](#_Toc460220496)

[Abbildung 27: Standardzustand der GUI-Elemente im Adminmodus 57](#_Toc460220497)

[Abbildung 28: Signale und Slots für die unterschiedlichen Modis 58](#_Toc460220498)

[Abbildung 29: Speichern im Benutzermodus 59](#_Toc460220499)

[Abbildung 30: In der Registry speichern 59](#_Toc460220500)

[Abbildung 31: Auf Windows Registry zugreifen und speichern 60](#_Toc460220501)

[Abbildung 32: Speichern in der BitBoxTom.ini 61](#_Toc460220502)

[Abbildung 33: BitBoxTom.ini parsen 61](#_Toc460220503)

[Abbildung 34: BitBoxTom.ini 62](#_Toc460220504)

[Abbildung 35: Policy Datei Konstruktion Information flow 63](#_Toc460220505)

[Abbildung 36: .policy-Datei speichern 63](#_Toc460220506)

[Abbildung 37: Policy-Datei öffnen 64](#_Toc460220507)

[Abbildung 38: Policy Datei auswerten 64](#_Toc460220508)

[Abbildung 39: Windows Registry Benutzerkontensteuerung 66](#_Toc460220509)

[Abbildung 40: UAC Eintragung während der Installation 66](#_Toc460220510)

[Abbildung 41: Watchdog 68](#_Toc460220511)

[Abbildung 42: Signal und Slot für den Watchdog 68](#_Toc460220512)

[Abbildung 43: Experten Modus 69](#_Toc460220513)

[Abbildung 44: Browser in the Box und Security Policy Manager 70](#_Toc460220514)

[Abbildung 45: BitBoxInstallData 70](#_Toc460220515)

# Abkürzungsverzeichnis

EPS Endpoint Security

BitBox Browser in the Box

VBox Virtual Box

# Einleitung

Der Bedarf an Sicherheit ist heute eines der größten Bedürfnisse in der Informationstechnik. Die Bedrohung durch Wirtschaftsspionage und Cyberkriminelle nimmt weiter zu. Das wachsende Bedrohungspotential hat Auswirkung auf die Art und Weise wie Unternehmen mit der Sicherheit in der Informationstechnik umgehen. Die Unternehmen sind sich bewusst geworden, dass Sie im Firmennetzwerk, Internettechnologien wie z. B einen Browser nicht mehr ohne Sicherheitsmaßnahmen einsetzen können. Gerade in Zeiten in denen Antiviren Software keinen verlässlichen Schutz mehr bieten vor Angriffen über den Browser, erlaubt das Konzept von Browser in the Box uneingeschränkten Zugang zum Web und schützt das Betriebssystem vor Schadsoftware. Neben dem eigentlichen Schutz der Virtualisierten Umgebung, bekommt der Benutzer zusätzlich die Möglichkeit, die so genannten Security Policies der BitBox selber festzulegen.

## Motivation

Im letzten Teil der Einleitung wurde darauf hingewiesen, dass der Benutzer die Möglichkeit hat, die Security Policies der BitBox selber zu konfigurieren. Dabei muss klar gestellt werden, dass zwei Varianten von der BitBox entwickelt wurden. Die sogenannte Standalone Variante steht für Privatanwender kostenlos zur Verfügung. Die Managed Variante ist für Unternehmer, die allerdings Geld kostet. Die Unternehmerversion bietet die zentrale Administration aller BitBox Clients. Die Standalone Variante hingegen wird auf dem lokalen System verwaltet, es handelt sich also um eine Einzelplatz Lösung.

Allerdings konnte der Benutzer bei der Standalone Variante, die Einstellungen der Security Policies bisher nur während der Installation festlegen und hatte im Nachhinein keine Benutzerfreundliche Möglichkeit, um diese zu ändern. Der Anwender musste sich also den Umstand machen, die BitBox neu zu installieren, nur um die Policy zu ändern. Diese umständliche Methode sorgt zudem für eine schlechte Benutzererfahrung beim Benutzer, da die heutigen Endanwender es gewohnt sind, Konfigurationen an der Software, jederzeit durchführen zu können. Es ist also sehr wichtig, dem Benutzer diese Möglichkeit bei der BitBox Standalone ebenfalls zu ermöglichen.

## Zielsetzung

Im Rahmen dieser Bachelorarbeit soll die BitBox mit einer eigenständig laufenden Anwendung erweitert werden, die es dem Endbenutzer ermöglicht, die Security Policies, über eine Komfortable und einfach zu bedienende grafische Benutzeroberfläche einzustellen. Ziel ist es vor allem, dass diese Anwendung einen Benutzermodus anbietet und einen Administratormodus. Der Benutzermodus tritt in Kraft sobald die Anwendung aus der BitBox gestartet wird. Die Einstellungen des Benutzers müssen von der BitBox sofort übernommen werden, sofern diese vom Benutzer bestätigt wurden.

Der Administratormodus hingegen, ist eine eigenständige Anwendung und wird vom Administrator außerhalb der BitBox gestartet. Die Einstellungen der Security Policies vorgenommen und in Form einer Policy-Datei am gewünschten Ort gespeichert. Ziel hierbei ist es, dass der Administrator nicht für jede einzelne Installation der BitBox Standalone, die Einstellungen der Sicherheitsrichtlinien vornehmen muss, sondern die Policy-Datei mit den gewünschten Einstellungen, der Installation beilegt und diese während der Installation automatisch übernommen werden. Dadurch soll dem Administrator überflüssige mehr Arbeit, erspart bleiben.

Der Benutzermodus muss zudem in den vorhandenen Quellcode der BitBox vollständig eingebunden sein. Die Anwendung selber startet zwar zur Laufzeit als eine eigenständige Ausführbare Anwendung, der Code selber ist allerdings mit der BitBox gekoppelt. Das bedeutet, dass der Benutzermodus teilweise vom Code der BitBox abhängig ist und somit außerhalb der BitBox nicht im Benutzermodus gestartet werden kann.

Der Administratormodus soll hingegen unabhängig vom Quellcode der BitBox funktionieren und steht deshalb in keiner Beziehung zum Quellcode der BitBox.

# Grundlagen und Begriffsdefinition

Dieses Kapitel bietet einen Einblick in die Grundlagen und Begriffsdefinition der Endpoint Security und Security Policy. Zunächst werden die Grundlagen, sowie einige Begriffsdefinitionen und Konventionen vermittelt, die in Bezug auf diese Bachelorarbeit eine Rolle spielen. Anschließend wird die BitBox näher vorgestellt und auf die Eingesetzten Technologien eingegangen, mit denen die BitBox Entwickelt wurde.

## Endpoint Security

Aufgrund der nicht klaren Definition des Begriffes Endpoint Security [2], ergibt sich die Notwendigkeit für die Klarstellung was denn mit Endpoint Security im Rahmen dieser Arbeit gemeint ist. Es muss also entschieden werden, was denn überhaupt Endpoints sind und welche Rolle die Sicherheit dabei spielt.

### Begriffe und Konventionen

Aus traditioneller Sicht betrachtet sind Endpoint Geräte vor allem Desktop Rechner und Laptops die an das Internet angeschlossen werden. Es sind also Systeme die Menschen benutzen um Daten zu erzeugen, löschen, manipulieren und versenden. Allerdings wissen wir, dass mittlerweile zu diesem Zweck nicht nur Desktop Rechner und Laptops eingesetzt werden, sondern Smartphones, Smart Watches, Mini-PCs und andere Hardware die am Endpunkt eines TCP/IP Netzwerkes angeschlossen werden. Deshalb kann man den Begriff Endpoint nicht auf bestimmte Geräte reduzieren oder erwarten dass dieser Begriff in Zukunft nicht sogar neu verstanden werden muss. Allerdings haben all diese Systeme etwas gemeinsam: Sie sind alle am Endpunkt eines TCP/IP Netzwerkes angeschlossen, die Verbindung kann dabei Kabellos (W-LAN, Mobile Internetverbindung) oder aber auch Kabelgebunden sein [1].

Sobald ein Endgerät mit einem Netzwerk verbunden ist, das eine Verbindung zum Internet hat, ist es automatisch der Gefahr ausgesetzt, dass es Gehackt, infiziert oder in irgendeiner anderen unerwünschten Weise manipuliert wird. Um derartiges zu vermeiden, muss das System die Möglichkeit anbieten, es so zu konfigurieren um derartige Angriffe abzuwehren.

Das System Windows zum Beispiel ist ein Endpunkt, wo der Benutzer über den Browser Zugang zum Internet bekommt und darüber auch mit dem Netzwerk interagiert. Windows, bietet die Kapazität einen speziellen Browser einzusetzen, mit dem die aufgezählten Angriffe abgewehrt werden können. Oder um es mit der Terminologie aus der Sicherheit in der Informationstechnik zu beschreiben: Eine Endpoint Security Lösung kann angewendet werden, für die Abwehr von Angriffen, die als Angriffsfläche den Browser des Endgerätes nutzen. Das Unternehmen Rohde & Schwarz Cybersecurity hat, als Endpoint Security Lösung, Browser in the Box entwickelt. Das heißt Angriffe aus dem Internet die über den Browser laufen sollen, würden nur innerhalb des gekapselten Browsers stattfinden und somit ins Leere gehen.

In der Gesamtheit versteht man unter Endpoint Security: Software, die Endgeräte vor Angriffen schützen soll.

## IT - Security Policy

Vor der eigentlichen Entwicklung von Endpoint Security Software, müssen die Security Policies (Sicherheitsrichtlinien) bestimmt werden. Für eine ausgereifte Endpoint Security Software, ist eine gut definierte Security Policy unabdingbar. Sie schreibt vor mit welcher Strategie, die Ziele, für die Informationssicherheit durch die Endpoint Security Software, erreicht werden. Vor allem in dieser Arbeit spielt die Security Policy eine wichtige Rolle. Schließlich war die Aufgabe, dem Benutzer eine Anwendung anzubieten, mit denen er die BitBox Security Policies konfigurieren kann.

Neben der eigentlichen Definition, ist wichtig zu verstehen welche Bedeutung die IT – Security Policy für den Software Entwickler hat und welche für den Anwender.

### Definition

Eine Security Policy ist eine auf höherer Ebene dokumentierte Anforderung, dessen Einhaltung von den Interessengruppen erwartet wird. Die Policy selber ist in der Regel kurz und formal gehalten. Im Idealfall sollte sie bündig und einfach zu verstehen sein, so dass jeder die in der Security Policy dargelegten Weisung einhalten kann. Wichtig zu verstehen ist hierbei, dass eine Security Policy nicht beschreibt, wie die Policy umgesetzt werden soll, sondern was konkret geschehen muss um die Sicherheit zu gewährleisten. Es ist also kein Dokument, das die technische Umsetzung beschreibt oder allgemein spezifische Lösungen vorgibt, für die Umsetzung der Security Policy in der Software selber [1].

Die Security Policy definiert Absichten und Regeln, deren Einhaltung dabei helfen soll die erforderte Sicherheit zu gewährleisten.

### Bedeutung für Software Entwickler

Die Entwickler bekommen wie bereits oben beschrieben, keine spezifische Lösung für die Implementierung der Security Policy vorgeschrieben. Die Entwickler müssen selber eine Lösung erarbeiten, wie sie die Policy in die Enpoint Security Software Umsetzen und am Ende implementieren.

Die Entwickler sind vor allem dazu verpflichtet bereits in der Entwurfsphase des Softwareentwicklungsprozesses, die Architektur und Entwurfsentscheidungen entsprechend der geforderten Security Policies zu wählen. Das heißt, dass keine Entscheidung getroffen werden darf, welche die gewünschten Policies nach der Implementierung, völlig oder teilweise nutzlos werden lässt [3].

Browser in the Box bietet zum Beispiel eine Security Policy an, wie Dateien die mit der BitBox heruntergeladen wurden, behandelt werden sollen. Der Benutzer hat die Möglichkeit die Policy so zu konfigurieren, dass alle heruntergeladenen Dateien automatisch abgelehnt werden, also die Datei nicht auf dem Host-System gespeichert wird. Die Einhaltung der Policy soll gewährleisten, dass Schadprogramme oder andere schädlichen Dateien, nicht unkontrolliert auf dem Host System landen. Die Entwickler könnten den Entwurf der Software nun so gestalten, dass die Datei automatisch auf dem Host-System sofort gelöscht wird, ohne dass der Benutzer etwas davon mit bekommt. Dadurch wäre aber die Policy nicht eingehalten worden, denn die Policy schreibt vor, dass diese Datei innerhalb der gekapselten Umgebung der BitBox bleiben soll und auf keinen Fall das Host-System betreten darf, auch wenn diese Datei sofort gelöscht wird.

Für die Entwickler ist also sehr wichtig, dass sie Ihren Entwurf der Software so gestalten, dass auf alle Fälle die Security Policy genau eingehalten wird. Eine spätere Änderung, sofern dies überhaupt noch möglich ist, kann zu teilweise aufwendige Veränderung an der Software führen [1]. Die Entwickler müssen dafür sorgen, dass die Software die sie entwickeln, jegliche Aktionen, die der Security Policy widersprechen, verhindert. Die Umsetzung der Security Policy auf dem Endgerät über Software, ist zugleich der einfachste und sicherste Weg um zu gewährleisten dass die geforderten Security Policies eingehalten werden [1].

### Bedeutung für Software Anwender

Sofern die Security Policy durch die Software umgesetzt wird, braucht sich der Anwender keine großen Gedanken über die Einhaltung der Policy machen. Denn wie bereits erwähnt, ist es am einfachsten und sichersten diese Aufgabe einfach der Software zu überlassen. Der Software Anwender bekommt im Falle der BitBox lediglich nur die Möglichkeit angeboten, die Policy nach Bedarf zu konfigurieren. Welche Security Policies genau zum Einsatz kommen, wird im Browser in the Box Kapitel erläutert.

### Aufbau einer Security Policy

Der Aufbau einer Security Policy sollte immer klar, verständlich und einfach zu verstehen sein [1]. Das Dokument mit den Policies ist nicht nur für Personen mit technischem Hintergrund bestimmt. Jede relevante Person, soll die Policy auf dem Dokument verstehen können. Der Aufbau kann aus folgenden Komponenten bestehen:

* Autor: Der Policy Autor.
* Sponsor: Führungsperson.
* Bevollmächtigter: Verantwortliche Person für die Freigabe.
* Zulassung: Ab wann die Policy gilt und zulässig ist.
* Nachprüfung: Datum für erneute Prüfung der Policy.
* Ziel: Wieso existiert diese Policy und welches Ziel hat es.
* Geltungsbereiche: Wo wird die Policy angewendet.
* Ausnahmen: Wer oder was ist nicht von der Policy betroffen.
* Durchführung: Wie die Policy durchgeführt wird.
* Definitionen: Begrifflichkeiten die der Lese nicht kennen könnte.
* Referenzen: Verknüpfungen zu anderen Policies, sofern vorhanden.

Die aufgezählten Punkte sind auf keinen Fall die einzig richtige Möglichkeiten Security Policies aufzustellen. Allerdings werden dabei derartige oder ähnliche Punkte aufgestellt und bearbeitet [1].

#### Policy Bereiche

Die Security Policy bezieht sich nicht nur auf einen einzigen Bereich. Vielmehr widmet man sich den unterschiedlichen Bereichen eines Unternehmens, wo ein Bedarf an IT-Sicherheit vorliegt. Zur Verdeutlichung werden einige Beispiele aufgezählt, welche Bereiche darstellen, für die Security Policies angefertigt werden könnten [1].

* **Computer Policies**: Dieser Bereich umfasst alle PCs, Laptops und Informationssysteme. Hier müssen also Security Policies entworfen werden, in der festgehalten wird, wie sich jeder Benutzer Authentifizieren muss. Beispielsweise mit einem Benutzernamen und dem dazugehörigen Passwort, welches nicht weitergegeben werden darf. Eine weitere Policy könnte vorschreiben, dass die Benutzer bzw. Mitarbeiter nicht in der Lage sein dürfen, eigene Accounts zu erstellen oder ihre eigenen zu Modifizieren. Selbst welche Art von Passwörtern benutzt werden dürfen und wie diese aussehen müssen, kann vorgegeben werden. Dazu gehört auch der Zeitraum in der die Passwörter gültig sein sollen. Eine weitere Policy in diesem Bereich könnte das gleichzeitige einloggen von Unterschiedlichen Benutzern auf einem Server verbieten. Sollte dennoch eine Notwendigkeit dafür vorliegen, müssen derartige Ausnahmen, explizit behandelt werden.
* **Netzwerk Policies:** Die Netzwerkstruktur, welches ein Unternehmen nutzt, stellt natürlich ein hohes Risiko dar. Es kann von Hackern als Angriffsfläche genutzt zu werden. Die Security Policies konzentrieren sich hier vor allem auf die Regelung von eingehendem und ausgehendem Datenverkehr. Ein Mitarbeiter zum Beispiel, der von Zuhause aus auf das Firmennetzwerk zugreifen möchte, darf dies nur nach den Vorgaben der Security Policies. Eine Policy könnte zum Beispiel vorschreiben nur über VPN (Virtual Private Network) eine Verbindung zum Firmennetzwerk aufzubauen. Und das alle anderen Verbindungen geblockt werden.
* **Datenschutz Policies:** Datenschutz ist natürlich ein großes Thema für ein Unternehmen. Unterschiedliche Policies müssen hier also genau festlegen, welche Daten dem Unternehmen gehören, die von den Mitarbeitern erstellt werden. Das heißt die Urheberrechte des Unternehmens sollten durch die Policy vorgegeben werden. Des Weiteren können Policies vorschreiben, dass bestimmte Daten nur verschlüsselt permanent gespeichert werden dürfen. Also alles was die Handhabung der Daten eines Unternehmens angeht, wird anhand der Security Policies geregelt.
* **Datenintegrität Policies:** Informationen müssen stets erreichbar sein und nicht unerwünscht manipuliert, unbrauchbar oder gelöscht werden. Deshalb sind für diesen Bereich Security Policies notwendig, die derartige Angriffe verhindern. Eine Policy könnte so aussehen, dass der Einsatz von Antiviren Software für jeden PC bzw. Laptop Pflicht ist. Eine andere Policy hingegen könnte vorschreiben, wie der E-Mail Server Mails mit einer dubiosen Überschrift oder Dateianhang, behandeln soll.

# Browser in the Box

In diesem Kapitel wird die Endpoint Security Software Browser in the Box genauer vorgestellt. Dieser Teil der Arbeit, soll dem Leser dabei helfen, die Idee und das Konzept, sowie die für diese Arbeit relevanten Besonderheiten besser zu verstehen. Bisher wurden die Begriffe und Definitionen der IT-Sicherheit behandelt, die in Bezug zu dieser Arbeit eine Rolle spielen. Welche unterschiedlichen Versionen der BitBox entwickelt wurden, worin sie sich unterscheiden und welche Bedeutung jene Version für die Bachelorarbeit hat, wird in diesem Teil erläutert. Zudem wird ein Einblick gewährt, aus welchen Technologien die BitBox entstanden ist und wie das Design der virtualisierten Surfumgebung aussieht.

## Varianten

Bereits mehrfach wurde auf die unterschiedlichen Versionen der BitBox hingewiesen. Sie unterscheiden sich vor allem in Ihrer Einsatzumgebung. Es existiert die Standalone Variante als Einzelplatz Lösung und die Managed Variante für das Unternehmen und Behördenumfeld.

### Standalone

Die Standalone Variante der BitBox ist als Einzelplatz Lösung konzipiert worden. In erster Linie sollen damit Heimanwender im Privaten Umfeld sowie sehr kleine Unternehmen die Möglichkeit bekommen die BitBox zu nutzen. Diese Variante macht vor allem Sinn, wenn die Infrastruktur nicht aus Internetgateways oder Firewalls besteht, sondern lediglich ein Bedarf nach einem Browser existiert, mit dem man sich geschützt im Internet bewegen kann. Die Art der Verbindung kann dabei ganz normal über die jeweilige Schnittstelle des Systems stattfinden. Das kann entweder die vorhandene DSL Leitung sein oder aber auch eine mobile Internetverbindung.

### Managed

Die Managed Variante der BitBox ist speziell für Unternehmen sowie das Behördenumfeld konzipiert, welche eine größere Netzwerkinfrastruktur haben. Der Einsatz der Managed Version macht vor allem Sinn, wenn eine Vielzahl von Anwendern im jeweiligen Unternehmen oder Organisation, die BitBox verwenden sollen. Zudem haben derart große Unternehmen oder Organisationen zusätzlich einen Administrator, der für das Netzwerk zuständig ist. Der Einsatz der BitBox soll daran nichts ändern, und dem Administrator die Möglichkeit anbieten die Administration der BitBox Clients zentral vornehmen zu können. Die Managed Variante erlaubt zudem den Einsatz, in einer Infrastruktur mit einem Internetgateway oder eines VPN (Virtual Private Network).

## Technologien

Bislang wurde allgemein das Konzept der BitBox vorgestellt. Doch aus welchen Technologien die Entwicklung erst überhaupt möglich war, soll in diesem Kapitel erklärt werden. Die Virtualisierung selber wurde nicht von Rohde & Schwarz entwickelt, sondern mit Hilfe der Open Source Virtualisierungssoftware *Virtual Box,* realisiert. Alle anderen Funktionalitäten der BitBox wurden mit der höheren Programmiersprache Python 2.7 entwickelt.

### Virtual Box

Die Open Source Virtualisierungssoftware VirtualBox, wurde ursprünglich entwickelt vom Unternehmen InnoTek Systemberatung GmbH, welche von Sun Microsystems übernommen wurde. Sun Microsystems wurde dann später im Januar 2010 von Oracle aufgekauft. Erst seit dieser Zeit, wird VirtualBox offiziell von Oracle vertrieben [1.1].

VirtualBox ist Plattformübergreifend und kann derzeit auf den Betriebssystemen Windows, Mac, Linux und Solaris betrieben werden. VirtualBox erlaubt den Einsatz von mehreren unterschiedlichen Virtuellen Maschinen, die einzige Einschränkung ist die Kapazität der Festplatte auf der VirtualBox installiert wurde bzw. auf der die Virtuellen Maschinen angelegt werden [4]. Wichtig zu erwähnen ist, dass bei VirtualBox zwischen Host-System und Gast-System unterschieden wird. Das Host-System ist das Betriebssystem, welches auf dem PC oder Laptop nativ installiert wurde und auf dem wiederum VirtualBox installiert und betrieben wird. Das Gast-System hingegen, ist das Betriebssystem der Virtuellen Maschine, welches innerhalb der virtualisierten Umgebung von VirtualBox läuft.

Je nach Bedarf können Virtuelle Maschinen angelegt werden, auf denen Betriebssysteme wie Windows oder Linux laufen. VirtualBox erweitert somit die Einsatzmöglichkeiten des eigenen PCs oder Laptops. Die Bedienung von VirtualBox ist Einsteigerfreundlich aufgrund der übersichtlichen und einfach zu bedienenden grafischen Benutzeroberfläche. Es werden also keine speziellen Kompetenzen vorausgesetzt.

Die Einsatzszenarien von VirtualBox sind vielfältig. Entwickler können zum Beispiel Ihre Software auf mehreren Betriebssystemen gleichzeitig testen, ohne dafür unterschiedliche PCs oder Laptops einzusetzen. Es müssen lediglich unterschiedliche Virtuelle Maschinen, für das gewünschte Betriebssystem erstellt werden und das funktioniert auf nur einem PC oder Laptop. Das ist sehr praktisch und spart vor allem Ressourcen, was immer im Interesse eines Unternehmens ist. VirtualBox besitzt sogar eine virtuelle Netzwerkkarte, mit der jegliche Netzwerkkommunikation zwischen den Virtuellen Maschinen und dem Host Betriebssystem gewährleistet wird. Dabei unterstützt VirtualBox fünf Netzwerkmodi [3]:

* Der *NAT* (Network Address Translation) Modus von VirtualBox emuliert einen Netzwerkrouter, für die Verbindung zwischen Gast und Host Netzwerk.
* Der *bridged* Modus von VirtualBox emuliert auf dem Host ein Netzwerkinterface, welches es erlaubt sowohl Routing als auch Bridging zwischen dem Gast und dem restlichen Netzwerk herzustellen.
* Der *interne* Netzwerk Modus, gewährleistet die Kommunikation zwischen den Virtuellen Maschinen, die sich allerdings im selben internen Netzwerk befinden.
* Der *host-only* Modus baut die Verbindung zwischen Host und einem oder mehreren Gast Systemen über eine gesonderte Netzwerkverbindung auf. So kann der Host auf einfache Weise eine Verbindung zu den Gast Systemen aufbauen, zum Beispiel um die Gast Systeme mittels SSH zu verwalten.
* Der Anwender hat auch die Option, das Netzwerkinterface vollständig zu deaktivieren.

Der Anwender hat zudem die Möglichkeit, Verzeichnisse vom Host Betriebssystem in das Gast Betriebssystem einzubinden und somit als Gemeinsamer Ordner, für den Datenaustausch zu benutzen. Selbst bei der Zwischenablage braucht sich der Anwender keine Sorgen zu machen, denn auch dieser Kommunikationskanal zwischen Host und Gast Betriebssystem wird von VirtualBox unterstützt [3]. Die gemeinsame Zwischenablage kann folgendermaßen konfiguriert werden und dadurch nach Bedarf eingeschränkt werden:

* Host zu Gast
* Gast zu Host
* Bidirektional
* Deaktiviert

### Python

Python ist eine interpretierte höhere Programmiersprache, die im Jahre 1991 erschienen ist und unter der *Python-Software-Foundation* Lizenz vertrieben wird [1.2].

Die gesamten Funktionalitäten der BitBox wurden mit Python 2.7 entwickelt. Die Sprache ist zwar stark dynamisch und erinnert mehr an eine simple Skriptsprache, doch die Unterstützung mehrerer Programmierparadigmen und das große Angebot an Bibliotheken, ermöglichte die Entwicklung einfacher sowie komplexer Funktionalitäten für die BitBox.

Python liefert alle notwendigen Bibliotheken, für die Entwicklung von Desktop Anwendungen im Windows Umfeld. Die Entwicklung beschränkt sich nicht nur auf die Python Laufzeit, sondern die Benutzung der Windows Programmierschnittstellen ist sogar möglich. Die *Win32* Erweiterung für Python ist eine Ansammlung von Bibliotheken, welche die Windows Programmierschnittstellen zur Verfügung stellen. Vor allem bei der Entwicklung der BitBox ist dies von großer Bedeutung. Da die Windows API relevanten Funktionalitäten der BitBox, über die Win32 Erweiterung von Python realisiert wurden. Als Beispiel sei hier der Umgang mit Dateien erwähnt. Wenn der Benutzer Daten zwischen BitBox und Host-System austauschen möchte, werden diese nicht ohne weiteres auf das Host-System gespeichert, sondern entsprechend der konfigurierten Policy behandelt. Ein weiteres Beispiel, welches sogar im Rahmen dieser Bachelorarbeit genutzt wurde, ist die Möglichkeit mittels der Python Win32 Erweiterung, die Windows Registry zu manipulieren und zu überwachen. Denn die Anwendung die Entwickelt wurde im Rahmen dieser Arbeit, muss in der Windows Registry die Security Policy relevanten Registry Einträge neu setzen und auf Veränderung überwachen. Im Verlauf dieser Bachelorarbeit, wird noch auf die eigentliche Entwicklung mit Python eingegangen. Für die Entwicklung in dieser Bachelorarbeit wurde ebenfalls Python 2.7 eingesetzt.

## Sicherheitskonzept

Das Grundlegende Sicherheitskonzept der BitBox zeichnet sich, wie bereits genannt, dadurch aus, dass der eigentliche Browser in einer isolierten Umgebung ausgeführt wird. Diese Browser-VM welches als Gast-System vorgestellt wurde, ist das System dessen Infizierung durch Schadsoftware akzeptiert wird. Allerdings dient diese Akzeptanz nur dafür, um in erster Linie das eigentliche Host-System vor dieser Infizierung zu schützen [5].

Das bedeutet allerdings nicht, dass die Browser-VM völlig ungeschützt einem Angriff überlassen wird. Falls ein Angriff stattgefunden hat, so soll der Angreifer nur mit enormen Aufwand aus der virtuellen Surfumgebung ausbrechen und in das eigentliche Ziel, dem Host-System, zugreifen können. Die Browser-VM also das Betriebssystem auf dem der Browser innerhalb der virtuellen Surfumgebung läuft, ist ein minimiertes und gehärtetes Linux Betriebssystem. Zudem wird für die Ausführung der Browser-VM, ein gesonderter Benutzer *BitBox* angelegt, so dass im Falle eines Ausbruches aus der virtuellen Surfumgebung, die Schadsoftware die eingeschränkten Benutzerrechte des BitBox Benutzers hat [5].

Das Anwenderkonto hat keinen direkten Zugriff auf die Browser-VM und deren Schnittstellen. Um dennoch die Steuerbefehle an die virtuelle Surfumgebung zu übertragen, wird ein Proxy zwischen BitBox und Anwender benutzt. Dieses Proxy Programm übermittelt die relevanten Befehle an den BitBox-Systemdienst, der wiederum diese Befehle an die virtuelle Maschine weiterleitet. Durch dieses Konzept wird sichergestellt, dass der Anwender bzw. das Anwenderkonto keinen direkten Zugriff auf die Schnittstellen der virtuellen Maschine hat [5].

### Security Policies und Architektur

Die Security Policies für den Informationsaustausch zwischen der Browser in the Box und dem Host-System, welche in Bezug zu dieser Arbeit stehen, sollen in diesem Kapitel erläutert werden. Es geht vor allem darum aufzuzeigen, welche Security Policies der Anwender selber konfigurieren kann.

#### Zwischenablage

Das kopieren und einfügen über die Zwischenablage darf nicht unkontrolliert stattfinden. Daher wird dieser Informationsaustausch zwischen Host-System und der virtuellen Surfumgebung überwacht [5].

* Der erlaubte Zugriff auf die Zwischenablage sieht vor, dass der Benutzer einen markierten Text aus der virtuellen Surfumgebung in das Host-System oder aus dem Host-System in die virtuelle Surfumgebung kopiert.
* Der verbotene Zugriff auf die Zwischenablage sieht vor, dass der markierte Text aus dem Host-System in die virtuelle Surfumgebung oder umgekehrt, unterbunden wird.
* Der eingeschränkte Zugriff sieht vor, dass der markierte Text aus dem Host-System in die virtuelle Surfumgebung oder umgekehrt, über einen Dialog mit dem Benutzer stattfindet. Der Benutzer bekommt einen Hinweis, dass gerade der Inhalt der Zwischenablage kopiert wird.

Es soll noch angemerkt sein, dass diese Policies von ihrer Informationsflussrichtung getrennt konfiguriert werden können. Für welche Richtung (Host to Guest oder Guest to Host) welche Policy gelten soll.

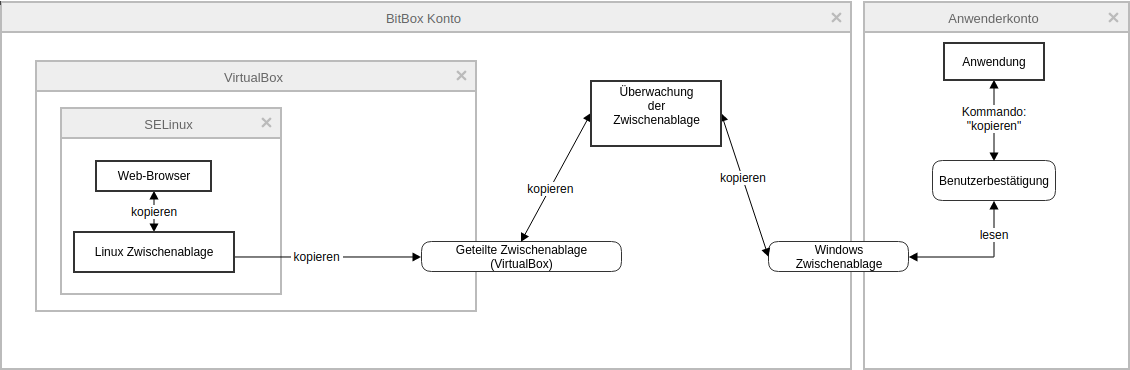


Abbildung 1: Sichere Zwischenablage

#### Downloaden

Das Downloaden von Dateien aus dem Internet innerhalb der virtuellen Surfumgebung ist abgesichert. Jegliche Downloads werden von der BitBox überwacht. Der Benutzer kann die Security Policy der Downloadüberwachung selber konfigurieren und somit bestimmen, welche angewendet werden soll [5].

* Der erlaubte Download, verschiebt eine Datei die innerhalb der virtuellen Surfumgebung heruntergeladen wurde, automatisch auf das Host-System.
* Der eingeschränkte Download, verschiebt eine Datei die innerhalb der virtuellen Surfumgebung heruntergeladen wurde nicht automatisch auf das Host-System. Ein Dialog gibt dem Benutzer den Hinweis, dass eine Datei auf das Host-System verschoben werden soll. Der Benutzer kann der Verschiebung zustimmen oder verweigern.
* Die Überprüfung auf Viren, ist eine zusätzliche Policy zu den ersten beiden genannten. Diese kann zusätzlich aktiviert werden, wenn vor der Verschiebung auf das Host-System die Datei zusätzlich auf Viren geprüft werden soll. Handelt es sich bei der Datei um eine infizierte, wird diese automatisch abgelehnt, ansonsten wird sie entsprechend der gewählten Policy behandelt.
* Der verbotene Download schreibt vor, dass jegliche Datei die heruntergeladen wurde innerhalb der virtuellen Surfumgebung automatisch abgelehnt wird. Der Benutzer bekommt lediglich nur den Hinweis, dass der Download von Dateien verboten ist.

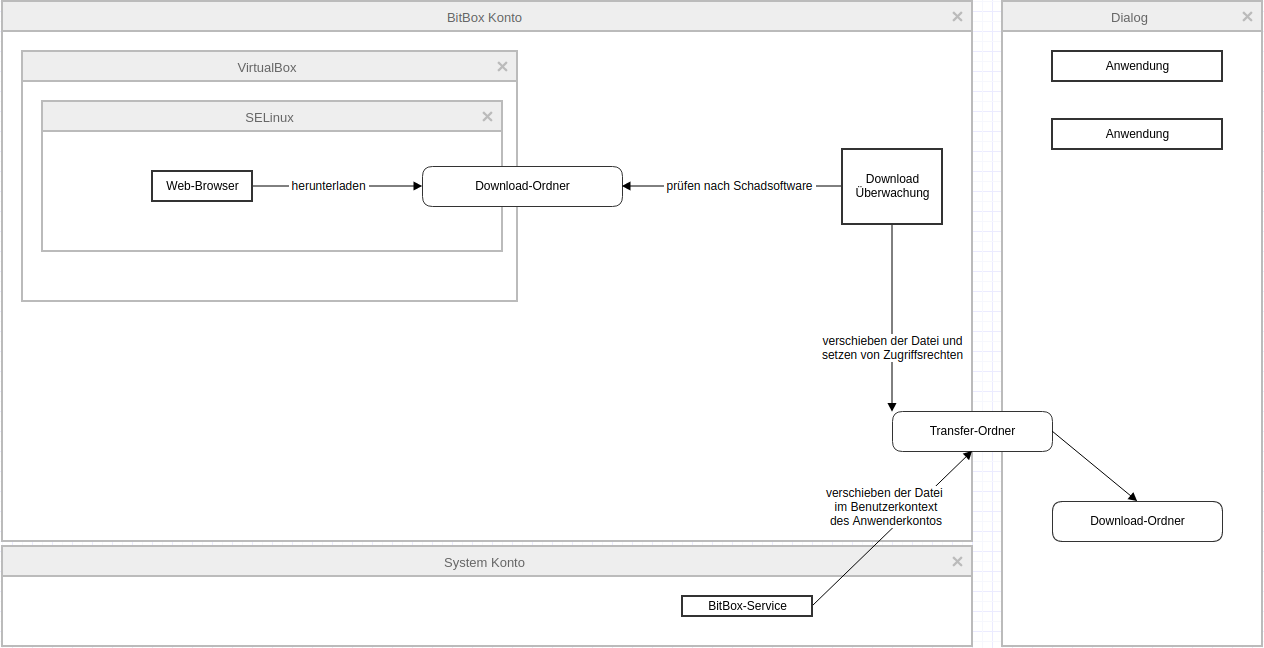


Abbildung 2: Sicherer Download

#### Uploaden

Genau wie der Download ist auch der Upload von Dateien abgesichert und wird ebenfalls vom BitBox Systemdienst überwacht. Sobald der Benutzer über die Grafische Benutzeroberfläche der BitBox eine Datei hochladen möchte oder über den Windows-Explorer mit „Senden an BitBox“, wird die Datei erst in den Transfer-Ordner kopiert. Sobald die Datei sich in diesem Transfer-Ordner befindet, wird dem BitBox Benutzer die nötigen Benutzerrechte übergeben, der BitBox Systemdienst wird darüber informiert, welcher wiederum die Security Policy überprüft. Ist die Datei freigegeben wird diese in den gemeinsamen Ordner verschoben, welcher auch aus der virtuellen Surfumgebung aus zu erreichen ist [5].

* Der erlaubte Upload, verschiebt eine Datei vom Host-System sofort in die virtuelle Surfumgebung.
* Der eingeschränkte Upload, verschiebt eine Datei nur dann vom Host-System in die virtuelle Surfumgebung, wenn der Benutzer über das Dialog dem Upload zustimmt und das Passwort seines Windows Benutzerkontos richtig eingegeben hat.
* Der verbotene Upload, verhindert jeglichen Upload versuch.

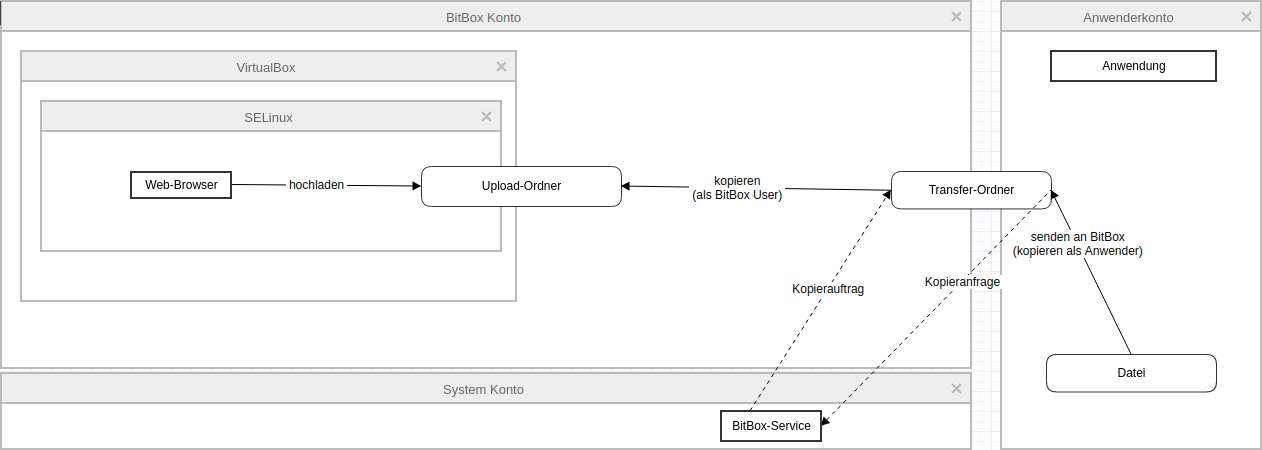


Abbildung 3: Sicheres Uploaden

#### Drucken

Das Drucken von Dokumenten und Webseiten aus der BitBox heraus, ist möglich und auch hier kann der Benutzer konfigurieren, wie gedruckt werden soll. Der Druckvorgang ist abgeleitet vom Downloadvorgang. Der Druckvorgang sieht vor, dass wenn ein Dokument gedruckt wird, dieser automatisch vom Druckdienst des Gast-Systems in eine PDF umgewandelt wird. Diese PDF wird dann in den Druck Ordner abgelegt. Die Druck-Überwachung prüft die Datei und die konfigurierte Security Policy wird angewendet [5].

* Das erlaubte Drucken, ermöglicht dem Benutzer innerhalb der virtuellen Surfumgebung zu drucken.
* Das verbotene Drucken, unterbindet das Drucken innerhalb der virtuellen Surfumgebung.

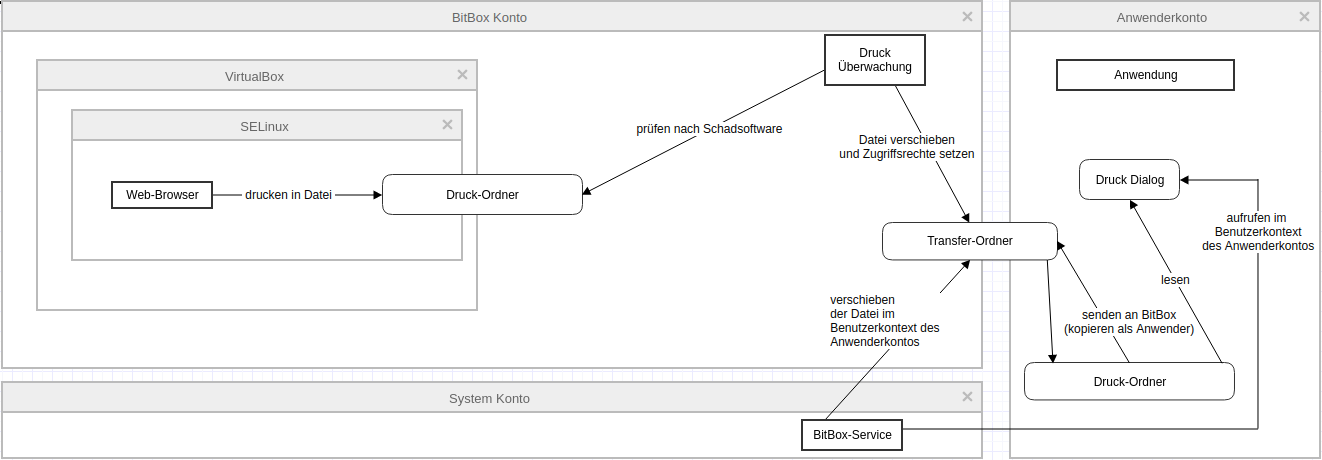


Abbildung 4: Sicheres Drucken

#### Persistente Daten

Mit der virtualisierten Surfumgebung der BitBox, wurde ein Konzept realisiert, mit dem es möglich ist, den Browser innerhalb des Gast-Systems immer wieder mit einem sauberen Startzustand zu beginnen. Möglich ist das durch die Snapshot Funktion der VirtualBox. Dazu wird das komplette Speicherabbild der Virtuellen Maschine eingefroren, von wo aus es dann wiederholt gestartet werden kann. Der Benutzer muss einfach nur die BitBox neustarten, wenn er sich während der Browser Sitzung irgendwelche Schadsoftware, innerhalb der Virtuellen Surfumgebung eingefangen hat. Nun hat der Benutzer aber auch den Nachteil, dass all seine Persistenten Daten: Lesezeichen, Browser Plugins und die Browserkonfiguration, die er während der Browser Sitzung gespeichert hat, verloren gehen [5].

Deshalb wurde dem Benutzer die Option gegeben, ob er diese einschränkte Einstellung beibehalten möchte, oder ob bestimmte Persistente Daten gespeichert und nach einem Neustart der BitBox wiederhergestellt werden sollen. Dem Benutzer stehen dazu folgende Konfigurationsmöglichkeiten zur Verfügung:

* Die Einstellung dass nur die Lesezeichen persistent bleiben sollen, speichert nur die die gesetzten Lesezeichen. Das heißt Browser Plugins die während der Browsersitzung gespeichert wird, gehen verloren.
* Die Persistenten Daten können aber auch so konfiguriert werden, das alles gespeichert wird. Das heißt Lesezeichen und die installierten Browser Plugins werden nach dem Neustart der BitBox vollständig wiederhergestellt.
* Als letzte Konfigurationsmöglichkeit, kann der Benutzer auch festlegen, dass gar nichts gespeichert werden sollen. Das bedeutet, dass nachdem die Browser Sitzung beendet wurde vom Benutzer, startet die BitBox ohne die Widerherstellung der Persistenten Daten.

Die Speicherung der Persistenten Daten, erfolgt über den Gemeinsam Ordner des Gast-Systems. Dort werden diese zunächst gespeichert. Der Sinn dahinter ist, dass diese Persistenten Daten vom BitBox-Konto zugreifbar sind und somit auch nach jedem Neustart der BitBox wiederhergestellt werden können. Dieser Gemeinsame Ordner wird in das Gast-System mit der Einschränkung, dass keine ausführbaren Dateien gestartet werden können, eingebunden. Die Persistenten Daten die der Benutzer gespeichert haben möchte, werden zusätzlich beim Beenden der BitBox Sitzung zusätzlich gepackt, verschlüsselt und signiert. Diese wiederum wird dann in das Benutzerverzeichnis des Anwenderkontos vom Host-System verschoben [5].

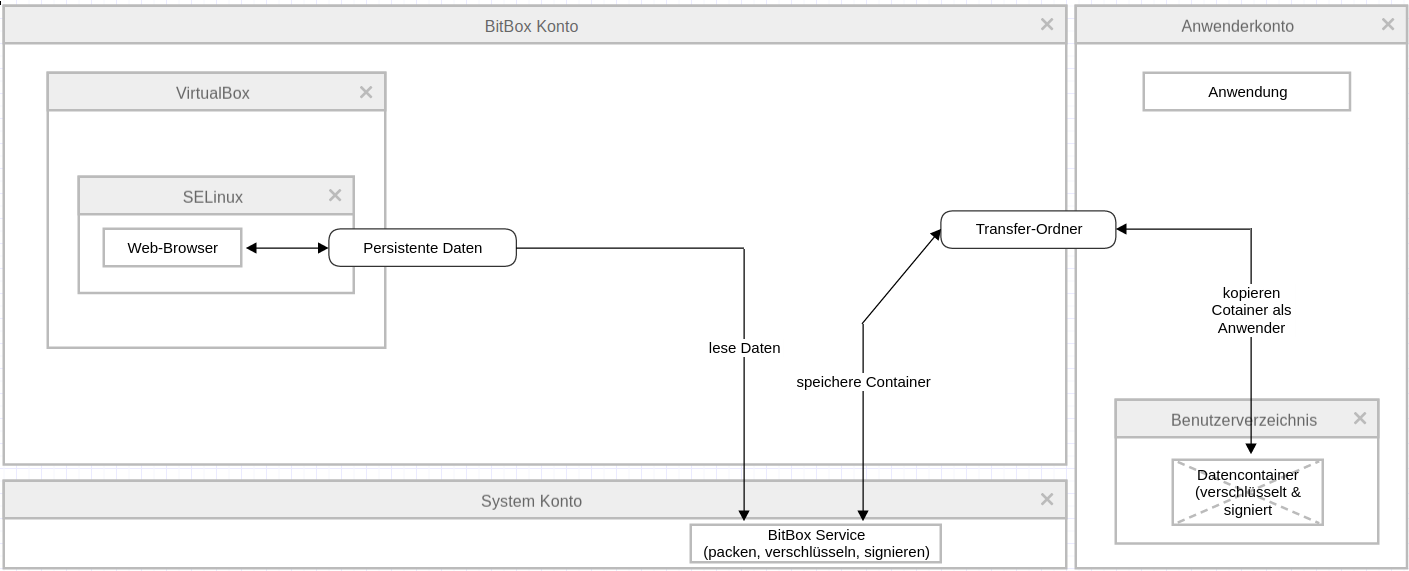


Abbildung 5: Sicheres speichern der Persistenten Daten

## BitBox Tray App

Die Tray App ist eine Anwendung die zur BitBox gehört, welche in der Benachrichtigungsleiste von Windows ausgeführt wird. Bei jedem Start der BitBox wird diese Tray App mit gestartet und läuft auch im Hintergrund weiter, selbst wenn die BitBox beendet wird. Der Benutzer hat aber die Möglichkeit die TrayApp selber zu beenden, sofern er dies wünscht.

Über die Tray App Einstellungen, kann der Benutzer diverse Konfigurationen vornehmen sowie die aktuelle Konfiguration sich anzeigen lassen, wie auf der Abbildung 6 zu sehen ist. Im Bereich *Zentral verwaltete Einstellungen*, sind die Security Policies gelistet, die der Benutzer derzeit eingestellt hat. Im Bereich der Benutzereinstellungen, sind die Benutzerspezifischen Einstellungen für die BitBox zu sehen. Und im letzten Teil Status, wird der allgemeine Status der BitBox gezeigt.

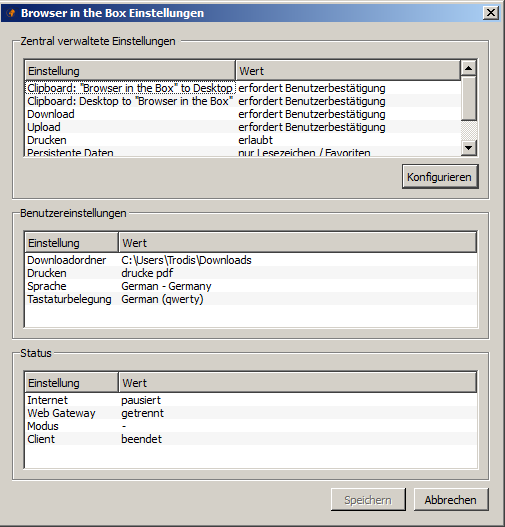


Abbildung 6: Tray App Einstellungen

Neben der eigentlichen Konfiguration der BitBox, kann der Benutzer auch Steuerbefehle an die BitBox senden. Zu diesen Steuerbefehlen gehört:

* Starten der BitBox
* Eine gewünschte Datei über ein Dialog hochladen
* Die BitBox zurücksetzen, auf einen frischen Zustand
* Das Beenden der BitBox erzwingen, falls es nicht mehr reagiert bzw. abgestürzt ist.

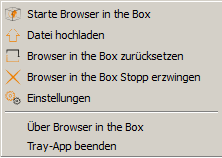


Abbildung 7: BitBox Steuerbefehle

Vor dieser Bachelorarbeit, war es nicht möglich die Security Policies zu konfigurieren, ohne die BitBox erneut zu installieren. Auf der Abbildung 6 ist nun in der Rubrik *Zentral verwaltete Einstellungen*, ein Button zusehen mit der Beschriftung *Konfigurieren.* Sobald der Benutzer diesen Button betätigt, öffnet sich die Anwendung die im Rahmen dieser Arbeit entwickelt wurde. Auf die eigentlich Anwendung sowie dessen Implementierung wird im nächsten Kapitel eingegangen.

# Implementierung

In diesem Kapitel wird die eigentliche Entwicklung behandelt, die im Rahmen dieser Bachelorarbeit durchgeführt wurde. Im ersten Teil dieses Kapitels werden die Anforderungen an die Anwendung vorgestellt. Welche Programmiersprache für die Entwicklung und welche Bibliothek für die Grafische Benutzeroberfläche zum Einsatz gekommen sind.

## Anforderung

Die Ausgangssituation war, dass es keine Möglichkeit gab, über eine grafische Benutzeroberfläche die Policy Einstellungen der BitBox vorzunehmen. Die Tray App ist zwar in der Lage, jegliche Einstellungen und konfigurierten Policies dem Benutzer anzuzeigen, allerdings musste der Benutzer die BitBox Standalone Varianten immer mit dem Experten Modus neu installieren um wiederum dort die nötigen Einstellungen vorzunehmen.

Die Anforderung ist in erster Linie, dem Benutzer eine grafische Benutzeroberfläche in Form einer Anwendung zur Verfügung zu stellen über diesen er dann die gewünschten Konfiguration vornehmen kann. Die Anwendung muss während der Laufzeit der BitBox über die Tray App gestartet werden können. Möglich soll das sein, indem der Anwender einfach nur auf einen Button klickt, welcher wiederum die Anwendung zur Konfiguration der Security Policies öffnet. Diese Anwendung wird ab diesem Kapitel *Security Policy Manager* genannt.

## Rahmenbedingungen

**Software** Der Security Policy Manager soll unter Windows 7 mit der x86 Architektur sowie der x64 Architektur lauffähig sein. Der Security Policy Manager muss auch außerhalb der BitBox Umgebung von einem Administrator benutzt werden können. Dieser so genannte Administratormodus, dient nur für die Erstellung von Policy Dateien.

**Benutzeroberfläche** Um die Bedienung des Policy Managers für den Anwender einfach zu gestalten, muss eine vollständige grafische Benutzeroberfläche entwickelt werden, mit dessen Bedienung der Benutzer von anderen Windows Anwendungen vertraut ist. Alle Konfigurationsmöglichkeiten müssen übersichtlich und geordnet aufgebaut werden. Zusätzlich soll es möglich sein, dass der Administrator über die BitBox.ini den Button zum Starten des Security Policy Managers deaktivieren kann.

## Akteure

Dieser Abschnitt stellt die Akteure vor, die für die Benutzung des Security Policy Managers in erster Linie in Frage kommen.

* **BitBox Benutzer**: Die Person, die Browser in the Box verwendet um auf das Web zuzugreifen und die Security Policies der BitBox jederzeit ändert.
* **Administrator**: Die Person, die den Security Policy Manager im Administratormodus startet und eine oder mehre Policy Dateien erstellt, in dem oder denen die BitBox Security Policy Konfiguration festgelegt sind. Diese Dateien kann der Administrator der BitBox Installation beilegen.

## Anwendungsfälle

In diesem Teil der Arbeit werden die Anwendungsfälle vorgestellt, die im Rahmen der Bachelorarbeit für den Security Policy Manager erarbeitet wurden. Die beiden Akteure sind der Benutzer und der Administrator. Wobei der Benutzer nur für den Benutzermodus in Frage kommt und der Administrator nur für den Admin Modus.

|  |  |
| --- | --- |
| Eindeutige ID | /AF 10/ |
| Titel | BitBox Security Policy ändern |
| Beschreibung | Der Benutzer startet den Security Policy Manager, konfiguriert die Security Policy. |
| Akteure | Benutzer |
| Vorbedingungen | Browser in the Box läuft und der Security Policy Manager wurde gestartet. |
| Nachbedingungen | Der Benutzer kann die Security Policies konfigurieren. |
| Normaler Ablauf | 1. Der Benutzer öffnet die Tray App. 2. Die Tray App ist geöffnet und der und der Button *Konfigurieren* ist sichtbar. 3. Benutzer klickt auf den Button *Konfigurieren* 4. Security Policy Manager wird im Benutzermodus gestartet. 5. Der Benutzer setzt die gewünschten Einstellungen. 6. Der Benutzer klickt auf Speichern, ein Dialog bittet um Bestätigung. 7. Der Benutzer klickt auf Ja, die Einstellungen werden alle für die BitBox gesetzt. 8. Security Policy Manager schließt sich automatisch. 9. Die neuen Einstellungen werden von der Tray App sofort erkannt und aktualisiert. |
| Alternative Ablauf | 1. Der Benutzer klickt auf Speichern im Security Policy Manager. 2. Der Dialog wird verneint und der Benutzer gelangt wieder zum Security Policy Manager. |

|  |  |
| --- | --- |
| Eindeutige ID | /AF 20/ |
| Titel | Policy Dateien erzeugen im Admin Modus |
| Beschreibung | Der Administrator starten den Security Policy Manager im Admin Modus um Policy Dateien zu erstellen. |
| Akteure | Administrator |
| Vorbedingungen | Security Policy Manager wurde im Admin Modus gestartet. |
| Nachbedingungen | Der Administrator kann die gewünschten Konfigurationen vornehmen. |
| Normaler Ablauf | 1. Administrator startet den Security Policy Manager im Admin Modus. 2. Die gewünschten Einstellungen für die Security Policy werden gesetzt. |
| Alternative Ablauf |  |

|  |  |
| --- | --- |
| Eindeutige ID | /AF 30/ |
| Titel | Policy Datei installieren |
| Beschreibung | Der Administrator legt die Policy Datei der BitBox Installation bei. |
| Akteure | Administrator |
| Vorbedingungen | Die Policy Datei wurde mit dem Security Policy Manager im Admin modus erstellt. |
| Nachbedingungen | Policy Datei liegt im BitBoxInstallData Verzeichnis. |
| Normaler Ablauf | 1. Policy Datei wurde in das BitBoxInstallData Verzeichnis vom Installer abgelegt. 2. *BitBox\_Setup.exe* wird ausgeführt vom Administrator. 3. Installer prüft ob die Datei existiert, falls ja wird diese in das Verzeichnis *SetupData* der BitBox Installation kopiert. |

|  |  |
| --- | --- |
| Eindeutige ID | /AF 40/ |
| Titel | Policy Datei in SetupData ablegen |
| Beschreibung | Der Administrator erstellt eine Policy Datei mit dem Security Policy Manager im Admin Modus. Diese erstellte Policy legt er nicht der BitBox Installation bei, sondern direkt in das SetupData Verzeichnis, von der BitBox Installation. Sobald die BitBox gestartet wird, liest die BitBox diese Datei ein, sofern es vorhanden ist, übernimmt die Security Policies und löscht anschließend die Policy Datei wieder aus dem SetupData Verzeichnis. |
| Akteure | Administrator |
| Vorbedingungen | Security Policy Manager wurde im Admin modus gestartet. |
| Nachbedingungen | Security Policy Datei wurde erstellt und in das SetupData Verzeichnis abgelegt. |
| Normaler Ablauf | 1. Administrator kopiert die erzeugte Policy Datei in das SetupData Verzeichnis der BitBox Installation. 2. BitBox wird gestartet. 3. BitBox prüft ob eine Policy Datei im SetupData Verzeichnis vorliegt. 4. Falls eine Datei vorliegt werden die Policies darin ausgewertet und von der BitBox übernommen. 5. Policy Datei wird nach der Auswertung gelöscht. |

|  |  |
| --- | --- |
| Eindeutige ID | /AF 50/ |
| Titel | Doppelklick auf Policy Datei |
| Beschreibung | Der Administrator klickt doppelt auf die erzeugte Policy Datei. Woraufhin sich der Security Policy Manager automatisch im Admin modus öffnet. |
| Akteure | Administrator |
| Vorbedingungen | Security Policy Manager ist auf dem System installiert. |
| Nachbedingungen | Security Policy Manager ist lauffähig und eine Policy Datei liegt vor. |
| Normaler Ablauf | 1. Administrator klickt doppelt auf die Policy Datei. 2. Security Policy Manager öffnet sich automatisch. 3. Security Policy Manager lädt automatisch die Einstellung der Policy Datei. |
| Alternative Ablauf | 1. Administrator startet den Security Policy Manager. 2. Policy Datei wird über den Button Laden übernommen. |

## Grafische Benutzeroberfläche

Die Grafische Benutzeroberfläche muss für den Benutzer sowie dem Administrator selbsterklärend aufgebaut werden. Jegliche Beschriftungen müssen logisch und nachvollziehbar gewählt werden. Die Benutzeroberfläche des Security Policy Manager muss alle Konfigurationsmöglichkeiten der BitBox Standalone Version, dem Benutzer und Administrator zur Verfügung stellen.

Der Administrator -und Benutzermodus unterscheiden sich nur sehr gering in ihrer grafischen Oberfläche. Wie bereits im vorherigen Kapitel näher beschrieben wurde, ist der Administratormodus nur für die Erstellung von Policy Dateien bestimmt. Der einzige Unterschied liegt nur in den beiden Buttons unten links, die auf der Abbildung 8 zu sehen sind. Der Benutzermodus hat den Button *Speichern* und *Zurücksetzen*, wohingegen der Adminmodus anstelle des Buttons zurücksetzen, den Button *Laden* zur Verfügung stellt.

Klickt der Benutzer im Benutzermodus auf Speichern, werden die Einstellungen sofort für die BitBox gesetzt. Im Administratormodus erscheint beim Klicken auf Speichern ein Dateidialog wo der Speicherort der Policy Datei bestimmt werden kann. Der *Lade* Button öffnet einen Dateidialog für das Laden einer Policy Datei.

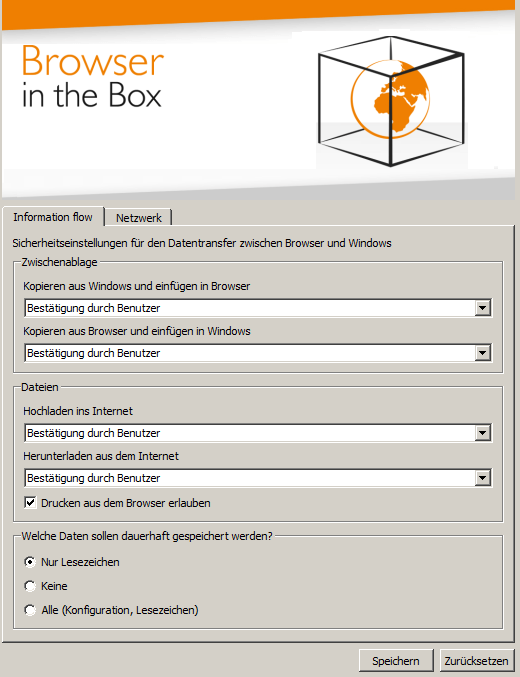


Abbildung 8: Security Policy Manager – Information flow

Abbildung 8 zeigt die grafische Benutzeroberfläche des Security Policy Managers. Die Oberfläche wurde in zwei so genannte Tabs unterteilt:

* Information flow
* Netzwerk

Der Tab *Information flow* stellt dem Benutzer die Security Policy Konfigurationsmöglichkeiten zur Verfügung, die in Kapitel 3.3.1 vorgestellt wurden. Dabei wurden die Kategorien, in der Art des Informationsflusses zwischen Host und Gast-System, unterteilt.

Der zweite Tab *Netzwerk* beinhaltet Netzwerkrelevante Konfigurationsmöglichkeiten. Der Benutzer bzw. Administrator kann hier einstellen ob die BitBox während der Browser Sitzung einen Proxy benutzen soll oder nicht und welche DNS Einstellungen für die BitBox übernommen werden sollen. Außerdem besteht die Möglichkeit die Proxy Einstellung des Browsers innerhalb der virtuellen Surfumgebung für den Anwender zu sperren.

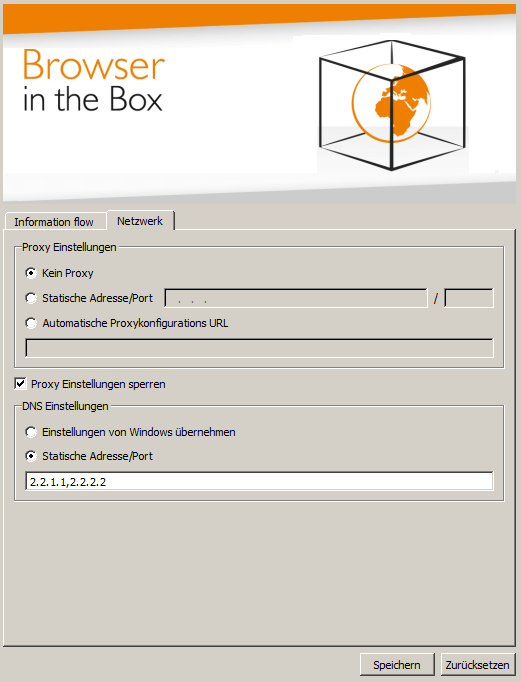


Abbildung 9: Security Policy Manager – Netzwerk

Die gesamte Entwicklung der Grafischen Benutzeroberfläche wurde mit dem Python GUI-Framework *PySide* implementiert.

### PySide

*PySide* ist ein mächtiges GUI-Framework, womit sich vollständige Windows Desktop Anwendungen in Python realisieren lassen. Die Benutzeroberfläche ist nicht zu unterscheiden von Anwendungen, die in nativen Windows spezifischen Technologien entwickelt wurden, wie zum Beispiel mit .NET oder der MFC Entwicklung.

*PySide* als Framework ist bis auf wenige Unterschiede, dasselbe Framework wie *PyQt*. *PyQt* hingegen ist eine Portierung des C++ GUI-Frameworks: *Qt*. Intern unterscheiden sich PySide und PyQt allerdings sehr stark. Grund dafür ist, dass die Entwicklung von PySide durch Nokia finanziert wird und dadurch auch intern eigene Verfahren anwendet, wie aus Qt Bibliotheken Python Module erzeugt werden. Die eigentliche Motivation von Nokia war, das PySide unter der freien LPGL-Lizenz vertrieben wird, so dass kommerzielle Software damit entwickelt werden kann ohne die Bedingung den Quellcode öffentlich zugänglich zu machen [6].

### Komponenten der GUI

In PySide bzw. der Welt von Qt allgemein, werden einzelne GUI-Elemente als Widgets bezeichnet. Eingabefelder, Buttons oder Text-Label sind zum Beispiel Widgets, nur um einige zu nennen. Für Ihre Anordnung auf der Oberfläche werden Layouts bereitgestellt. Der Entwickler muss entscheiden, welches Layout sich am besten eignet, für die gewünschte Anordnung der GUI-Elemente.

#### Freie Layouts

#### Die GUI-Elemente von PySide können auch frei auf dem Fenster platziert werden. Der Entwickler muss lediglich, für jedes Element eine fixe Größe und Position festlegen. Die einzelnen Elemente unterliegen nur dem Haupt Fenster der Anwendung. Das bedeutet, dass die Elemente nicht erst einem Layout zugewiesen werden müssen, welches dann als Layout mit seinen Elementen dem Hauptfenster zugeordnet wird.

#### Die Herausforderung für den Entwickler liegt darin, dass er abschätzen muss wie die Elemente sich platzieren, wenn die Fenstergröße verändert wird. Die Elemente werden zur Laufzeit nicht automatisch angepasst, was für eine schlechte Benutzererfahrung beim Anwender sorgen kann. Praktischer sind dagegen die Layouts, die diese Anordnung der Elemente automatisch übernehmen. Die folgenden Layouts die vorgestellt werden, übernehmen genau diese Aufgabe, wenn sich die Fenstergröße der Anwendung ändert [6].

#### Horizontales Layout

#### Das Horizontale Layout ermöglicht die automatische Horizontale Anordnung der GUI-Elemente. Alle Elemente innerhalb dieses Layouts werden automatisch nebeneinander angeordnet. Sobald sich die Fenstergröße zur Laufzeit der Anwendung ändert, wird die Größe der GUI-Elemente automatisch der neuen Fenstergröße korrekt angepasst. Im Netzwerk Bereich des Security Policy Managers, erweist sich dieses Layouts als sehr hilfreich um die Statische Proxy Konfiguration übersichtlich und für den Benutzer logisch anzuordnen. Das Eingabefeld für die Adresse des Proxys sowie dem zugehörigen Port konnten so nebeneinander angeordnet. Auffällig ist vor allem der Schrägstrich um Adresse und Port voneinander visuell zu trennen. Dieser Schrägstrich selber ist ebenfalls ein sogenanntes Label. Alle drei Elemente sind einem Horizontalem Layout zugeordnet und passen automatisch Ihre Größe an.

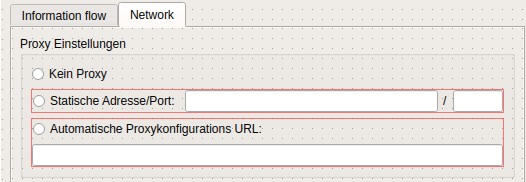


Abbildung 10: Horizontales Layout Statischer Proxy

#### Vertikales Layout

#### Das Vertikale Layout verhält sich in seiner Anpassung der Elemente, genau wie das Horizontale. Der Unterschied liegt allein darin, dass die Elemente Vertikal angeordnet werden. Möchte man die unterschiedlichen Komponenten untereinander automatisch anordnen und anpassen lassen, muss man alle Elemente innerhalb diesen Layouts platzieren. In Abbildung 10 zum Beispiel ist wurden die Elemente der Automatischen Proxy Konfiguration untereinander angeordnet. Das Radio Button mit der Beschriftung *Automatische Proxykonfigurations URL* ist ein Element und das Eingabefeld darunter ein weiteres. Beide Elemente sind innerhalb eines Vertikalen Layouts platziert und somit korrekt Angeordnet.

Vertikale und horizontale Layouts werden in PySide als *Box Layouts* zusammengefasst. Diese Box Layouts können miteinander verschachtelt werden, bis die gewünschte Anordnung aller Elemente erreicht ist. Der Entwickler wird nicht darauf eingeschränkt, dass innerhalb der Layouts nur Widgets platziert werden dürfen [6]. Dadurch lassen sich sehr aufwendige und anspruchsvolle Benutzeroberflächen realisieren, die den meisten Anforderungen gerecht werden.

#### Rasterlayout

#### Das Rasterlayout hat ein völlig anderes Konzept der automatischen Element Anordnung. Für die Platzierung der Elemente, wird ein Raster aufgespannt. In diesem Raster lassen sich dann die Elemente platzieren, der Entwickler muss lediglich die Spalte und Zeile angeben, in welchem das Widget platziert werden soll. Die Elemente werden also nicht der Reihe nach Nebeneinander oder Übereinander platziert, sondern dessen Position wird innerhalb des Rasters fest vergeben [6].

#### Stapellayout

Layouts können auch gestapelt werden. Dabei werden die Widgets auf einzelne Seiten angeordnet, diese Seiten sind dann wiederum gestapelt. Der Entwickler kann für ein Element genau eine Seite des Stapels festlegen. Eine Seite kann aber auch weitere Layouts sowie mehrere Elemente beinhalten. Die Komplexität des Aufbaus einer Seite, ist also dem Entwickler freigestellt. Im Prinzip können die vorherigen vorgestellten Layouts auf solche Stapel verteilt werden. Der Entwickler kann bestimmen, welche Seite zuerst während der Laufzeit der Anwendung angezeigt werden soll [6].

### Implementierung der GUI

Während der Softwaredesignphase wurde entschieden, dass die grafische Benutzeroberfläche und die Funktionalitäten des Security Policy Managers in voneinander getrennten Python Modulen implementiert werden. Alle Komponenten für die GUI wurden im Python Modul *gui.py* implementiert. Durch diese Trennung, soll die Verwaltung und Pflege des Quellcodes für die Entwickler erleichtert werden. Zudem sollen die nachfolgenden Entwickler, die nicht seit Beginn der Implementierung mitgewirkt haben, einen erleichterten Einstieg in den Code bekommen. In diesem Teil gilt es vor allem aufzuzeigen, wie Elemente bzw. Widgets aus dem PySide Framework implementiert werden.

Das Modul *gui.py* beginnt mit der Einbindung der beiden PySide Modulen *QtCore* und *QtGui.* Diese beiden Module, sind die Hauptmodule aus dem PySide Framework, um die gewünschte GUI zu entwickeln. Das Modul *QtGui* beinhaltet alle nötigen GUI-Klasse sowie Komponenten für die Entwicklung der GUI [6]. Dazu gehören zum Beispiel die Layouts aus Kapitel 4.5.2. Als Komponenten aus diesem Modul, sind Zum Beispiel die Radio Buttons (siehe Abbildung 10) oder aber auch die Combo Box (siehe Abbildung 8 – Dateien und Zwischenablage) zu verstehen. Das *QtCore* Modul aus dem PySide Framework beinhaltet alle Komponenten, die zur Laufzeit der Anwendung nicht sichtbar sind. Dazu gehören zum Beispiel Konstanten, die als Wert genutzt werden zum Festlegen der Fenstergröße oder der sichtbaren Komponenten [6].

E:\Hochschule Bochum\Bachelorthesis\diagramme\qtgui_und_qtcore_import.PNG

Abbildung 11: Import von QtCore und QtGui

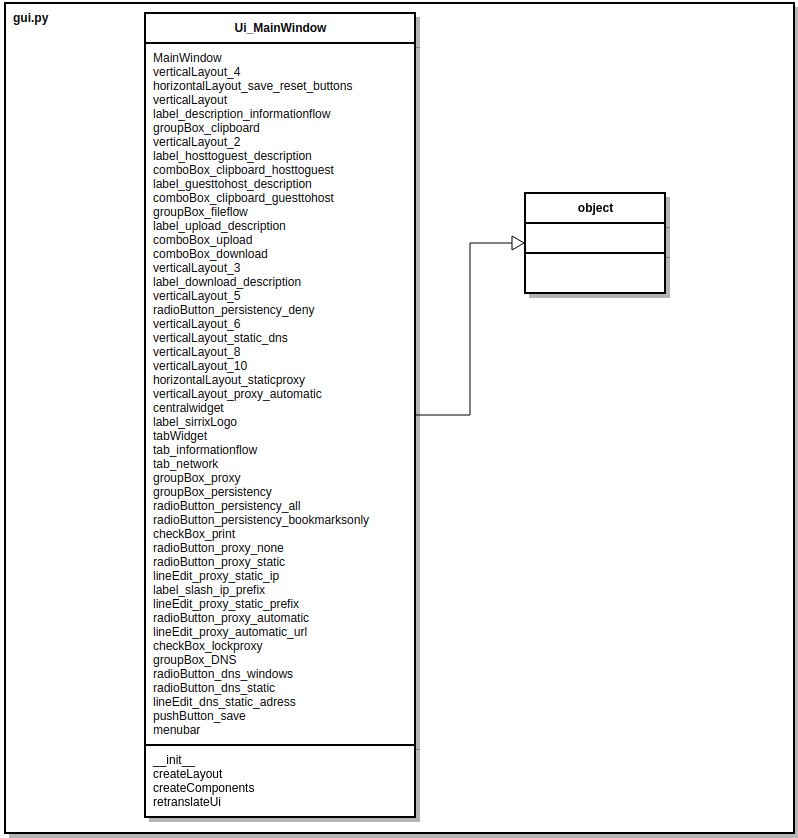


Abbildung 12: UML - Ui\_MainWindow Klasse

Innerhalb des *gui.py* Moduls wurde auf eine übersichtliche Struktur, bei der Implementierung der einzelnen Elemente, Wert gelegt. Die Hauptklasse innerhalb des Moduls ist die *Ui\_MainWindow* Klasse. Innerhalb dieser Klasse werden alle GUI-Elemente erstellt. Insgesamt drei Funktionen erzeugen aufeinander aufbauend, alle GUI-Elemente des Security Policy Managers. Für jemanden der sich mit Objekt orientierter Entwicklung in Python nicht beschäftigt hat, ist die *\_\_init\_\_* Funktion innerhalb einer Klasse, auf den ersten Blick verwirrend. In Python gibt es keinen expliziten Konstruktor wie man es z. B aus Java kennt. Die *\_\_init\_\_* Funktion verhält sich ähnlich wie ein Konstruktor, genauer gesagt in der Initialisierung des Objektes verhält es sich wie ein Konstruktor. Das Objekt selber wird allerdings nicht durch *\_\_init\_\_* erzeugt, weshalb es wiederum kein Konstruktor sein kann. Allerdings kann diese Funktion dazu genutzt werden, um das Objekt zu initialisieren, nachdem es erzeugt wurde. Diese Eigenschaft von *\_\_init\_\_* sorgt bei Einsteigern, der Programmiersprache Python, oft für Verwirrung [7].

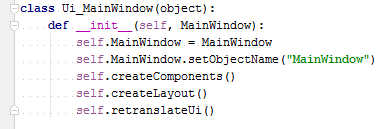


Abbildung 13: Ui\_MainWindow Klasse

Die Klasse *Ui\_MainWindow* wird nicht von irgendeiner Klasse aus dem PySide Framework abgeleitet, sondern direkt von der Python Basisklasse *object.* Der Hintergrund ist, dass *Ui\_MainWindow* selber abgeleitet wird, innerhalb der Hauptklasse im *policytool.py* Modul. Die *Ui\_MainWindow* Klasse, hat die Aufgabe alle GUI-Elemente zu erzeugen und diese der Klasse innerhalb des policytool.py Modules zur Verfügung zu stellen.

An dieser Stelle sei noch erwähnt, was genau eigentlich die Variable *self* ist die immer wieder genutzt wird im Quellcode. Die Variabel *self* ist vor allem im Kontext der Objekt Orientierten Programmierung mit Python, ein Thema. Diese Variable wird aufgerufen gefolgt von einer Punktnotation und einem Attribute bzw. einer Funktion auf die man Zugreifen möchte. In Jeder Funktion muss die Variable *self* als Parameter angegeben werden, dass ist Pflicht. Das Konzept dahinter ist, das die Instanz zu welcher eine Funktion gehört, automatisch der aufgerufenen Funktion übergeben wird. Allerdings wird diese Instanz nicht automatisch von einer Python Funktion erwartet. Der *self* Parameter innerhalb einer Funktion, sagt also explizit, dass eine Instanz von einem Objekt erwartet wird, zu dem das Objekt gehört, welches die Funktion aufruft. Man entwickelt in Python also nicht Instanz Funktionen, sondern Klassen Funktionen die eine Instanz als ersten Parameter erwarten müssen [7].

Alle Funktion auf die man innerhalb der Klasse *Ui\_MainWindow* aufrufen möchte, muss man mit der Variable *self,* explizit aufrufen. Dadurch weiß Python genau, dass die Funktionen innerhalb der Klasse gemeint sind.

Die *createComponents()* Funktion, erzeugt alle sichtbaren GUI-Elemente. Dazu gehören unter anderem Labels, Buttons, Checkboxen und Comboboxen, mit denen der Anwender, zur Laufzeit der Anwendung, interagieren kann. Sobald diese Funktion aufgerufen wird, werden alle Objekte hauptsächlich aus dem Modul *QtGui* instanziiert. Da alle Instanz Variablen mit der *self* Variable initialisiert werden, sind diese GUI-Elemente über die gesamte Klasse *Ui\_MainWindow* sichtbar. Auf diese Art können die anderen Funktionen, auf die erzeugten GUI Objekte zugreifen.

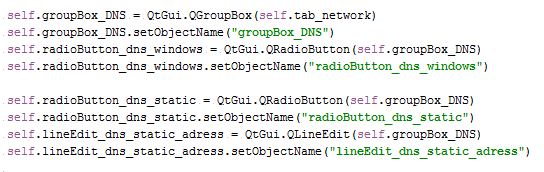


Abbildung 14: createComponents() Funktion – Elemente für die DNS Einstellung

In Abbildung 13 ist der Code Abschnitt, in denen die GUI-Elemente für die DNS Einstellungen (siehe Abbildung 9 unten), erzeugt werden. In der ersten Zeile der Abbildung 13, wird self.*groupBox\_DNS* mit einem *QGroupBox* Objekt aus dem Modul *QtGui* instanziiert. *QGroupBox* hingegen bekommt als Paramater ein Tab Objekt übergeben. Dieses Tab Objekt, ist ein Widget in dem die Elemente für die Netzwerkeinstellungen, platziert werden. Deshalb wird die *groupBox* dem Tab *tab\_network* zugewiesen. In Abbildung 9 ist dieses Tab aktiv, und in genau diesem Tab wurden auch die DNS Einstellungen zugeordnet.

Die Groupbox dient dazu, um die Radio Buttons in einer eigenen Gruppe zusammenzufassen, damit die gewünschte Ordnung der Radio Buttons erreicht wird. Durch diese Trennung können unterschiedliche Kategorien, in der grafischen Benutzeroberfläche, dargestellt werden. Anschließend werden die eigentlichen Radio Buttons aus dem *QtGui* Modul erzeugt. Dazu wird einfach nur das Modul *QtGui* aufgerufen und mit der Punktnotation das gewünschte GUI Objekt instanziiert. In diesem Falle, das *QRadioButton* Objekt. Als Parameter wird die Gruppe übergeben, in dem das Radio Button zugeordnet werden soll.

Nachdem die *createComponents()* Funktion abgearbeitet und alle GUI-Elemente bzw. Widgets erstellt wurden, folgt der Aufruf der *createLayout()* Funktion. Wie der Namen schon ahnen lässt, hat diese Funktion die Aufgabe alle gewünschten Layouts zu erstellen, die benötigt werden um die automatische Platzierung der GUI-Elemente zu gewährleisten. Das gewünschte Layout muss dafür zunächst als Objekt erzeugt werden, welchem dann später über die Funktion des Layout Objektes die gewünschten GUI-Elemente als Objekt hinzugefügt werden. Doch bevor auf den Code der *createLayout()* Funktion eingegangen wird, muss das Prinzip des Zentralen Widgets in PySide näher erläutert werden. In der *createComponents()* Funktion werden wie bereits erklärt, alle sichtbaren Komponenten erzeugt. Unter diesen Komponenten gibt es ein besonderes Widget, welches als *centralWidget* inAbbildung 14 zu sehenist*.*



Abbildung 15: Central Widget

Ein Zentrales Widget muss immer, bei der Verwendung des PySide Frameworks, als zentraler Einstiegspunkt einer PySide Anwendung definiert werden [6]. Ohne dessen Definierung, wären die Layouts nicht sichtbar und dadurch auch alle anderen Elemente innerhalb des Layouts nicht. In Abbildung 14 ist zu sehen, dass der Funktion QWidget, das Argument *MainWindow* übergeben wird. Diese Übergabe, definiert das *QWidget* Objekt zum zentralen Widget der Anwendung. Diesem Widget werden später alle anderen GUI-Elemente, zu denen auch die Layouts gehören, hinzugefügt.

Das *MainWindow* Objekt, welches von der *QMainWindow* Klasse instanziiert wurde, ist als Parameter innerhalb der *\_\_init\_\_* Funktion der *Ui\_MainWindow* Klasse festgelegt, wie auf Abbildung 12 zu sehen ist. Das heißt die erbende Klasse *GUIMainWindow,* aus dem *policytool.py* Modul, welches der Einstiegspunkt der Anwendung ist, muss das Hauptfenster Objekt *QMainWindow* aus dem *QtGui* Modul, der vererbten Klasse als Argument übergeben. Dadurch ist festgelegt, welche Klasse, das Hauptfenster innerhalb der Anwendung, ist. Jegliche Implementierung der GUI in der *Ui\_MainWindow* Klasse, bezieht sich immer auf genau dieses übergebene Hauptfenster. Das *MainWindow* Objekt ist demnach das Hauptfenster, in der alle GUI-Elemente initialisiert werden [6].

Die Reihenfolge der Funktionsaufrufe, in der *\_\_init\_\_* Funktion der U*i\_MainWindow* Klasse, ist nicht willkürlich gewählt worden. Wie bereits erwähnt, benötigt man die Layouts, um darin die GUI-Elemente bzw. Widgets zu platzieren. Doch würde man vor der Erstellung der Komponenten, die Layouts festlegen, müsste man alle Komponenten die erzeugt werden, innerhalb der Funktion *createComponents()* auch den jeweiligen Layouts sofort hinzufügen. Das würde dazu führen, dass man keine eindeutige Trennung mehr hätte im Quellcode. Ziel ist es nämlich, dass alle Layout relevanten Codeabschnitte und Komponenten Relevanten Abschnitte weitestgehend in voneinander getrennten Funktionen implementiert werden. Dieses Vorgehen liefert den Vorteil, das spätere Änderungen oder Verbesserung am Quellcode für die Benutzeroberfläche, viel einfacher ist. Denn die Positionierung der Komponenten innerhalb der Layouts, wird vom Entwickler festgelegt, indem er bestimmt, welches Element dem Layout in welcher Reihenfolge, über die *addWidget()* Funktion, übergeben wird. Diese Reihenfolge im Quellcode bestimmt, in welcher Reihenfolge die Elemente während der Laufzeit, angezeigt werden sollen. Falls eine spätere Veränderung an der Platzierung der Elemente vorgenommen werden soll, muss der Entwickler einfach in der *createLayouts()* Funktion, die Reihenfolge für das entsprechende Layout ändern. Im Rahmen dieser Arbeit, wurde diese Strukturierung gewählt, mit der Absicht dass der Quellcode besser zu verstehen ist und dadurch Änderungen an der Benutzeroberfläche einfacher zu gestalten. Allen Änderungen können direkt in *createLayouts()* Funktion unternommen werden, um die Elemente anders zu platzieren. Andernfalls müsste man im Quellcode ständig das GUI-Elemente und das dazugehörige Layout suchen, nur um die Platzierung zu ändern.

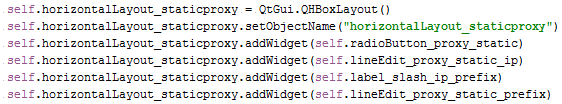


Abbildung 16: createLayouts() Funktion - Hinzufügen der Komponenten, in ein Horizontales Layout

Die Beschriftung der einzelnen Elemente, wird in der letzten Funktion *retranslateUi()* vorgenommen. Alle Komponenten die sich Beschriften lassen, besitzen dafür eine Funktion. Besteht die Komponente aus nur einem einzigen Element, wie zum Beispiel ein Button, so wird die Funktion *setText()* aufgerufen und als Argument ein String mit der gewünschten Beschriftung übergeben. Elemente die aus mehreren Eintragungen bestehen, müssen einzeln Beschriftet werden. Solch eine Komponente stellt die Funktion *setItemText()* zur Verfügung. Als Argument muss das Index des gewünschten Elements, als Integer übergeben werden und als zweites Argument, der gewünschte String.

Der Benutzer hat die Möglichkeit, die Sprache der BitBox umzustellen. Sollte der Benutzer die Sprache z. B von Deutsch auf English ändern, muss auch der Security Policy Manager diese Sprachänderung übernehmen. Die korrekte Lokalisierung musste also implementiert werden. Python liefert als Lösung, für die Internationalisierung, das Modul *gettext*. Das Modul bietet dem Entwickler eine Wrapper-Funktion an, dem ein nicht lokalisierter englischer String übergeben wird. Die Wrapper-Funktion liefert daraufhin, den String mit der korrekten Übersetzung als Rückgabewert zurück. Das Modul arbeitet nach einem Katalogprinzip um die gewünschte Übersetzung zu finden. Dafür müssen so genannte Sprachkompilats erstellt werden. Für die Erstellung, wird das Python Programm *pygettext.py* mitgeliefert. Wird diese Ausgeführt, stehen den Entwickler *.po* Dateien zur Verfügung, in denen die Übersetzung vorgenommen werden kann [7].

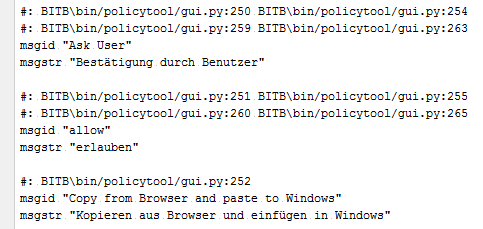


Abbildung 17: de.po Datei für die deutsche Lokalisierung

Bei der Erstellung der .po-Datei, wird automatisch durch das Programm *pygettext.py* ermittelt, welche Strings übersetzt werden müssen. Andernfalls wenn die Entwickler selber die .po-Datei erstellen müssten, wäre der Aufwand immens. Deshalb müssen alle für die Übersetzung relevanten Strings, nur noch mit der Wrapper-Funktion \_\_ implementiert werden.

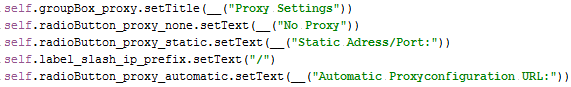


Abbildung 18: retranslateUi() Funktion - Strings mit gettext Wrapper-Funktion

Auf Abbildung 17 werden die einzelnen Komponenten, mit den gewünschten Beschriftungen versehen. Auffällig ist, dass den Funktionen für die Beschriftung keine Strings übergeben werden, wie z. B bei *label\_slash\_ip\_prefix.setText(“/“),* welches keine Lokalisierung benötigt. Stattdessen wird die Funktion \_\_ aufgerufen und dieser wird der Englische nicht lokalisierte String übergeben, als Rückgabewert liefert die Funktion den lokalisierten String, welche dann wiederum der Funktion, für die Beschriftung übergeben wird. Der Vorteil liegt darin, dass für die Internationalisierung, nicht mit dem Quellcode der Anwendung gearbeitet werden muss, sondern nur die .po-Datei editiert wird. So hat man die Möglichkeit, dass man die Übersetzung der Software, auch nicht Software-Entwicklern überlassen kann.

### Beurteilung PySide und Qt

Qt ist das Framework, für die Entwicklung einer plattformübergreifenden grafischen Benutzeroberfläche in C++. Das Qt Framework beinhaltet sogar mehr als nur GUI Bibliotheken. Der Entwickler bekommt alle notwendigen Bibliotheken für die Entwicklung von Netzwerk, Datenbank und OpenGL Anwendungen. Neben den Bibliotheken, wird auch ein Designer zur Verfügung gestellt, mit der die gesamte Erstellung der Benutzeroberfläche realisiert werden kann, ohne selber Code dafür zu schreiben. Dieser kann automatisch vom Designer generiert werden. All diese Funktionalitäten wurden vollständig in Python portiert. So dass auch Python Entwickler, alle Vorteile und Angebote von Qt nutzen können um mächtige Anwendungen zu verwirklichen. PyQt und PySide sind die Portierungen für Python. Für ein Unternehmen, bietet sich in erster Linie PySide als das optimale Werkzeug für GUI Anwendungen an. Denn PySide unterliegt der LGPL-Lizenz, und das Unternehmen ist somit nicht gezwungen, den Quellcode seiner kommerziellen Software zu veröffentlichen. Der Security Policy Manager wurde auch in erster Linie deshalb mit PySide entwickelt, weil für die gesamte BitBox Entwicklung bereits PySide verwendet wurde. Ein anderes GUI-Framework einzusetzen, ergibt deshalb keinen Sinn. Zudem wurde der Benutzermodus des Security Policy Managers vollständig in den Quellcode der BitBox integriert. Für einen reibungslosen Zusammenspiel und Erweiterung der BitBox Funktionalität, musste also auch das GUI-Framework verwendet werden, welches Bereits eingesetzt wurde.

Es spricht also nichts gegen den Einsatz von PySide. Ein Nachteil wäre es sogar gewesen, wenn ein anderes Framework eingesetzt worden wäre, mit dem es nicht so trivial ist zu arbeiten. Oder dessen Dokumentation nicht so ausführlich und professionell gestaltet ist, wie das von PySide bzw. Qt. Auch konnten mit PySide alle ästhetischen Ansprüche an den Security Policy Manager erfüllt werden. Die Anwendung erfüllt den Anspruch, wie eine native Windows Anwendung präsentiert zu werden, was äußert wichtig für Benutzererfahrung beim Endanwender ist.

Besonders hilfreich erwies sich der PySide Designer. Mit diesem sogenannten GUI-Builder, lassen sich nach dem Drag & Drop Prinzip grafische Benutzeroberflächen entwerfen. Nachdem die GUI im Designer fertig erstellt wurde, hat man die Möglichkeit den Python Code automatisch generieren zu lassen. Als Entwickler muss man sich nur noch, wenn überhaupt erforderlich, um die Strukturierung des generierten Quellcodes kümmern. PySide ist ebenfalls wie Qt Plattformunabhängig, welches ebenfalls ein riesiger Vorteil ist, vor allem weil Python selber eine interpretierte plattformunabhängige Programmiersprache ist.

## Implementierung der Anforderungen

Bisher wurde das Modul *gui.py* in der Klasse *Ui\_MainWindow* vorgestellt, in der alle GUI-Elemente für die grafische Benutzeroberfläche mit den gewünschten Eigenschaften implementiert wurden. Das Modul *policytool.py* hingegen, beinhaltet die Klasse *GUIMainWindow*, in der die GUI-Elemente Ihre eigentliche Funktion implementiert bekommen. Die gesamten Anforderungen an den Security Policy Manager, wurden innerhalb dieses Moduls realisiert. Dazu gehört auch die *main()* Funktion, welches der Einstiegspunkt einer Python Anwendung ist. Die eigentliche Ausführbare Datei *policytool.exe*, wird daher aus diesem Modul erstellt.

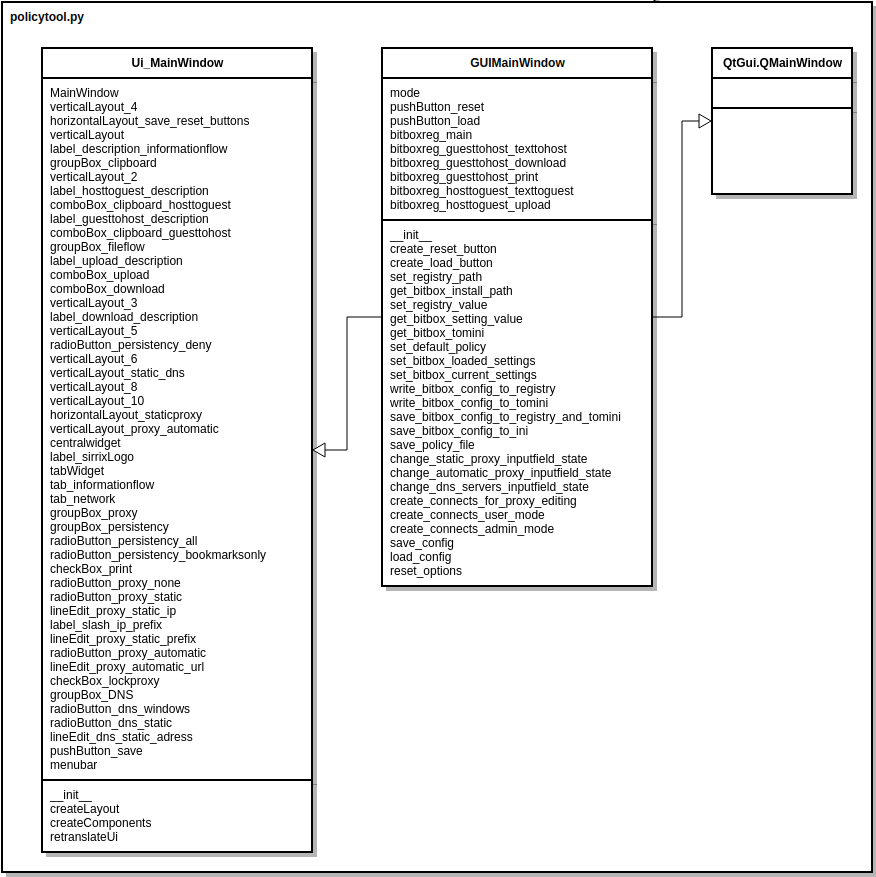


Abbildung 19: UML - Security Policy Manager

Auf der Abbildung 18 ist das UML-Diagramm abgebildet, welches die Beziehung zwischen den Klassen aufzeigt. Dabei geht es vor allem aufzuzeigen, in welchem Zusammenhang das *gui.py* zum *policytool.py* Modul steht. Die Attribute und Funktionen, der *QMainWindow* Klasse aus dem *QtGui* Modul, wurde bewusst nicht aufgelistet. Die vollständige Dokumentation, befindet sich in der offiziellen PySide Dokumentation [1.3].

Die Klasse *GUIMainWindow* erbt von *Ui\_MainWindow* und *QMainWindow.* Eine Mehrfach Vererbung wie in diesem Falle, wird von Python unterstützt. Wieso die Nutzung der Mehrfachvererbung während der Entwicklung für Sinnvoll befunden wurde, wird später erläutert, wenn die Klasse *GUIMainWindow* behandelt wird.

Die *main()* Funktion ist der Einstiegspunkt, von der aus die Ausführung des Security Policy Managers beginnt. Bevor der Quellcode von Python ausgeführt wird, werden besondere Variable definiert. Eines von diesen besonderen Variablen ist *\_\_name\_\_*. Diese Variable wird mit dem String *“\_\_main\_\_“* initialisiert, sofern das Modul selber ausgeführt wird. Importiert man das Modul allerdings in einem anderen Modul, wird die *\_\_name\_\_* Variable mit dem Name des Moduls initialisiert. Deshalb beginnt der erste Codeabschnitt mit einer if-Abfrage, in der genau dieser Fall untersucht wird.



Abbildung 20: if-Abfrage von \_\_name\_\_

Die Main Methode wird genau dann ausgeführt, wenn das Modul selber als Hauptprogramm ausgeführt wird und nicht anderswo importiert wurde. Dadurch wird die Möglichkeit, das Modul *policytool.py,* später vielleicht doch in einem anderen Modul zu importieren, nicht ausgeschlossen. Das *sys* Modul, ist eine Bibliothek in Python, die Zugang zu Variablen gewährt, die vom Python Interpreter verwaltet oder genutzt werden [1.4]. An dieser Stelle wird dieses Modul genutzt, um die Argumente, die der ausführbaren Datei des Security Policy Managers übergeben werden, der *main()* Funktion weiterzuleiten. Anhand der Argument wird zur Laufzeit entschieden ob die Anwendung innerhalb der BitBox gestartet wurde oder im Administratormodus.

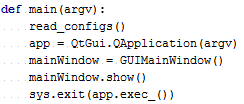


Abbildung 21: main() Funktion innerhalb des policytool.py Moduls

Die *main()* Funktion wurde außerhalb der Klasse *GUIMainWindow* implementiert. Innerhalb der *main()*, werden alle notwendigen Klassen instanziiert, die für die Ausführung der Anwendung benötigt werden. In Abbildung 21, beginnt die erste Zeile mit dem Aufruf der *read\_configs()* Funktion. Diese Funktion wurde von den Entwicklern der BitBox implementiert. Diese Funktion ist nicht während dieser Arbeit entstanden, sie wurde lediglich übernommen. Die allgemeinen Konfigurationen der BitBox, werden in der *BitBoxUser.ini* Konfigurationsdatei gespeichert. Zu diesen Konfigurationen, gehört auch die Sprache. Um die Lokalisierung des Security Policy Managers immer korrekt zu gewährleisten, muss diese BitBoxUser.ini ausgewertet werden. Genau diese Aufgabe übernimmt die *read\_configs()* Funktion. Wann immer der Security Policy Manager gestartet wird aus der BitBox heraus, wird die Konfigurationsdatei neu eingelesen, um zu prüfen welche Sprache eingestellt wurde. Deshalb ist diese Funktion auch die erste die ausgeführt wird innerhalb von *main()*.

Das Objekt *app* das aus der *QApplication* Klasse erzeugt wurde, ist für die Kontrolle der Abläufe innerhalb der GUI-Anwendung zuständig [6]. Als nächstes wird das Objekt *mainWindow* aus der *GUIMainWindow* Klasse erzeugt, in der die eigentlichen Funktionalitäten des Security Policy Managers programmiert wurden. Die Funktion *show()* muss aufgerufen werden, damit das Hauptfenster sichtbar wird, wenn die Anwendung ausgeführt wird [6].

Als letzter Schritt wird die eigentliche Anwendung in einem so genannten Event-Loop gestartet. Diese Schleife, wird so lange ausgeführt, bis der Benutzer das Hauptfenster schließt. Während dieser Schleife, werden alle Benutzereingaben des Anwenders, an das jeweilige Element im Hauptfenster übergeben. Der Funktionsname *exec\_(),* hat in Python zusätzlich einen Unterstrich bekommen. Das kommt daher, weil die Funktion in Qt eigentlich *exec()* lautet, allerdings ist dieser Funktionsname in Python bereits reserviert und konnte deshalb nicht direkt übernommen werden [6]. Sobald das Hauptfenster geschlossen wird, liefert die *exec\_()* Funktion einen Rückgabewert, welche der *sys.exit()* Funktion übergeben wird. Dieser Rückgabewert ist ein so genannter Exitcode, welches dem Betriebssystem mitteilt, ob die Anwendung sauber geschlossen wurde oder ein anderer Grund für das Beenden gesorgt hat.

### Das Signale und Slots Konzept

Dieses Unterkapitel hat das Ziel, das Konzept von Signalen und Slots nahezubringen. Andernfalls kann es zu Verwirrung führen, wie die Benutzereingaben überhaupt von der GUI-Anwendung behandelt werden. Das PySide Framework arbeitet mit diesem Konzept, um die Kommunikation zwischen den GUI-Objekten und anderen Funktion bzw. Objekten zu ermöglichen [6]. Ein Button zum Beispiel, kann vom Benutzer angeklickt werden. Sobald der Benutzer diesen Button anklickt, soll eine vom Entwickler implementierte Funktion ausgeführt werden. Und genau diese Kommunikation erfolgt über das Entwicklungsmuster von Signale und Slots. Die Funktionalität zum Laden der Policy Datei im Adminmodus, wird hier als praktisches Beispiel aufgeführt.



Abbildung 22: Signal und Slot Beispiel

Der Security Policy Manager soll sobald auf den Button Laden geklickt wurde, ein Dateidialog hervorrufen, in der der Benutzer eine Policy Datei auswählt und öffnet. Die Anforderung ist also, eine Funktion zu programmieren, welches das Laden der Policy Datei übernimmt. Allerdings nur wenn der Benutzer auf Laden geklickt hat. Das Button-Objekt in Abbildung 22 *pushButton\_load*, ist das Element auf das der Benutzer klicken kann. Sobald diese Benutzereingabe stattfindet, sendet das Objekt ein Signal aus, in diesem Falle *clicked*. Dieses Signal wird wiederum mit einem Slot verbunden, indem die Funktion *connect()* vom Signal aufgerufen wird. Diese Funktion erwartet als Argument Python-Callables [6], was in diesem Beispiel nichts weiter ist, als eine Funktion, die ausgeführt werden soll. Die Funktion *load\_configs()* kommt immer dann zum Einsatz, sobald das entsprechend Button ein *clicked* Signal entsendet.

### Funktionalitäten

Die *\_\_init\_\_* Funktion der Klasse *GUI\_MainWindow* beginnt mit der Überschreibung der *­\_\_init\_\_* Funktion der vererbten Klassen. Durch die Vererbung der *QMainWindow* Klasse, wird die *GUIMainWindow* Klasse als Hauptfenster der Anwendung festgelegt. Deshalb bekommt die *\_\_init\_\_* Funktion der *Ui\_MainWindow* Klasse als Argument, die *self* Variable übergeben, welches die Instanz der aktuellen Klasse beinhaltet. In Kapitel 4.5.3 wurde näher beschrieben, wieso dieses Argument erwartet wird von der *Ui\_MainWindow* Klasse, deshalb soll an dieser Stelle nicht weiter darauf eingegangen werden. Die Variable *mode* speichert die Argumente, die der *policytool.exe* übergeben wurden. Die Idee dahinter ist, der Anwendung mitzuteilen, in welchem Modus der Policy Manager arbeiten soll. Die Argumente werden in den nachfolgenden if-Abfragen untersucht. Die Mehrfachvererbung wird aus dem einfachen Grund genutzt, um GUI-Objekte aus der Klasse *Ui\_MainWindow* nicht jedes Mal in der Form *Ui\_MainWindow.comboBox\_upload* aufzurufen. Also das immer der Klassenname führen muss. Sondern das Objekt soll direkt über die *self* Variable aufrufbar sein. Dieser direkte Aufruf wird durch die Vererbung der Klasse *Ui\_MainWindow* realisiert. Allerdings muss auch die Hauptfenster Klasse *QMainWindow* geerbt werden, und dadurch muss eine Mehrfachvererbung stattfinden.

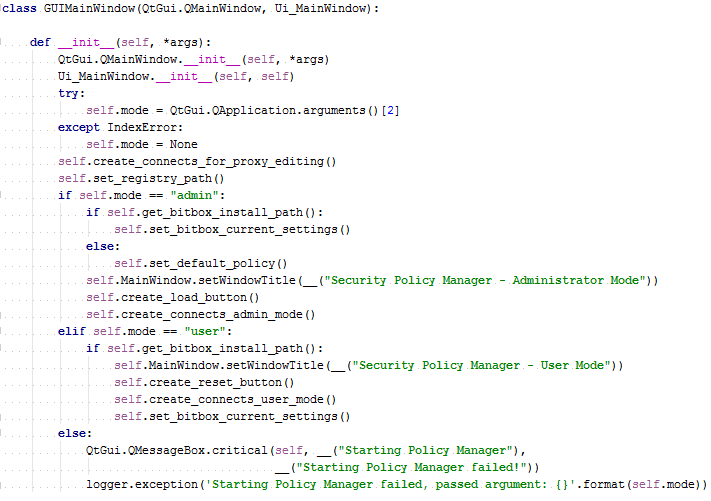


Abbildung 23: Ui\_MainWindow Klasse Initialisierung

Wenn die Anwendung im Adminmodus gestartet werden soll, wird anstelle des Zurücksetzen Buttons vom Benutzermodus, das Laden Button über die Funktion *create\_load\_button()* erstellt. Innerhalb dieser Funktion, wird dem Layout in der *Ui\_MainWindow* Klasse, einfach dass Button zum Laden hinzugefügt. Wird allerdings der Benutzermodus gestartet, bekommt dasselbe Layout einen Zurücksetzen Button, über die *create\_reset\_button()* Funktion.

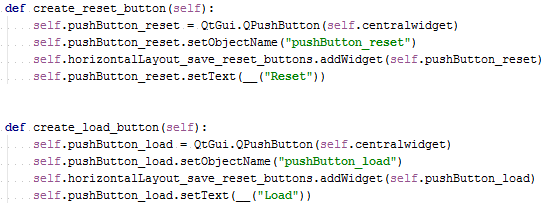


Abbildung 24: Dynamisches erzeugen von Buttons

Diese Lösung ermöglicht eine dynamische grafische Benutzeroberfläche, dessen Elemente vor dem Start der Anwendung bestimmt werden können. Die Funktion *set\_registry\_path()* wird ausgeführt ungeachtet in welchem Modus gestartet wird. Hier werden alle Security Policy relevanten Windows Registry Pfade definiert, die der Benutzer über die Anwendung neu konfigurieren kann.

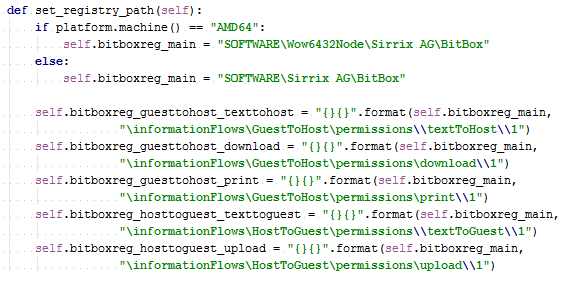


Abbildung 25: Registry Pfade der Security Policy Einstellungen

Diese Variablen, werden später in den Funktionen benötigt, welche die Registry Einträge manipulieren sollen.

Für den Adminmodus wird zudem geprüft, ob BitBox installiert wurde oder nicht. Die Prüfung erfolgt über die Methode *get\_bitbox\_install\_path()*.

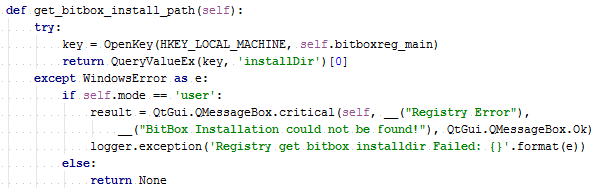


Abbildung 26: Prüfen ob BitBox installiert wurde

Die Prüfung erfolgt durch die Abfrage der Registry. Wenn eine BitBox Installation existiert, wird ein bestimmter Registry Eintrag vorgenommen. Sofern dieser Eintrag besteht, liefert die Funktion als Rückgabewert den Wert des Eintrages, ansonsten erfolgt die Rückgabe von *None*. Sollte BitBox installiert sein, werden die aktuellen Security Policies aus der Registry ausgelesen, sowie die Netzwerkkonfiguration aus der *BitBoxTom.ini*, die im Installationsverzeichnis liegt. So sollen auch im Adminmodus die Elemente der Benutzeroberfläche bereits einen bestimmten Zustand haben. Selbst wenn gar keine Installation von BitBox vorhanden ist, bekommt die Oberfläche im Adminmodus, einen vordefinierten Standardzustand, über die Funktion *set\_default\_policy()* zugewiesen.

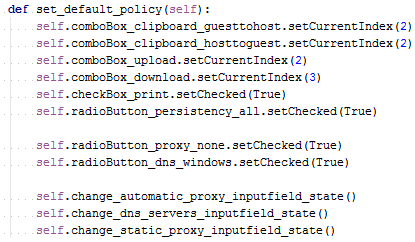


Abbildung 27: Standardzustand der GUI-Elemente im Adminmodus

Das Verhalten des Benutzermodus ist ähnlich, auch hier wird geprüft ob eine BitBox Installation vorhanden ist. Allerdings wird bei einer nicht vorhandenen BitBox Installation, keine Standard Konfiguration durchgeführt. Das starten des Security Policy Managers im Benutzermodus ist dann nicht möglich, da alle kritischen Funktion für den Benutzermodus nicht aufgerufen werden. Der Benutzermodus dient schließlich nur für die Echtzeitkonfiguration und darf außerhalb der BitBox nicht gestartet werden. Ist allerdings die BitBox im Benutzermodus installiert, werden die GUI-Elemente immer auf den Zustand der aktuellen BitBox Security Policy sowie den Einstellungen der BitBoxTom.ini gesetzt.

Die beiden Funktionen *create*\_*connects\_user\_mode()* und *create\_connects\_admin\_mode()*, haben die Aufgabe die Signale der Buttons mit den richtigen Slots zu verbinden.

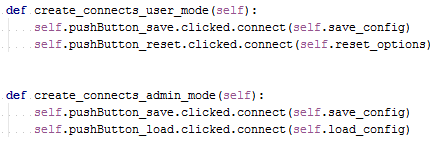


Abbildung 28: Signale und Slots für die unterschiedlichen Modis

Ein Klick auf das Button Speichern, ruft die Funktion *save\_configs()* auf. Im Benutzermodus existiert das Zurücksetzen Button, dessen Betätigung die Funktion *reset\_options()* ausführt. In dieser Funktion werden alle Elemente wieder auf den Zustand der aktuellen BitBox Security Policy, sowie der Netzwerkeinstellungen zurückgesetzt. Klickt der Anwender den Laden Button im Adminmodus an, wird die Funktion für das Laden der Policy Datei ausgeführt.

Nachdem alle Elemente der GUI, sowie die Variablen initialisiert wurden und die Buttons mit den gewünschten Slots verbunden sind. Wartet der Security Policy Manager zur Laufzeit auf die Benutzereingaben.

Sofern der Benutzer alle Einstellung vorgenommen hat und diese speichert, müssen alle Zustände der GUI-Elemente abgefragt und die Security Policy Einstellungen in der Windows Registry gesetzt, sowie die Netzwerkeinstellungen in der BitBoxTom.ini gespeichert werden.

Allgemein wird zum Speichern der im Security Policy Manager vorgenommen en Konfiguration, die Funktion *save\_config()* aufgerufen. Innerhalb dieser Funktion wird entschieden, für welchen Modus gespeichert werden soll. Im Benutzermodus sollen die Einstellungen in der Windows Registry sowie der *BitBoxTom.ini* gespeichert werden. Dies erfolgt in der *save\_bitbox\_config\_to\_registry\_and\_tomini()* Funktion.

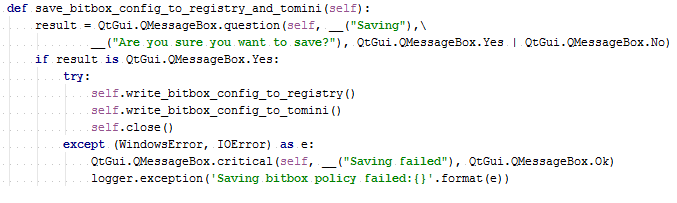


Abbildung 29: Speichern im Benutzermodus

Der Dialog fragt den Benutzer ob er sicher ist, dass er speichern möchte. Wenn der Benutzer Ja geantwortet hat, erfolgt der Aufruf der Funktion *write\_bitbox\_config\_to\_registry().* Dort werden die Windows Registry Einträge für die Security Policy, mit den entsprechenden Werten gesetzt, welche aus der GUI entnommen wurden. Die Werte der GUI-Objekte, stellen Ihre Funktionen zur Verfügung, über welche die aktuellen Werte abgefragt werden können. Zum Beispiel liefert der Aufruf *checkBox\_print.isChecked()* ein booleschen Wert zurück, welches Rückschluss darüber gibt, ob der Benutzer das Drucken innerhalb der BitBox erlauben möchte oder nicht.

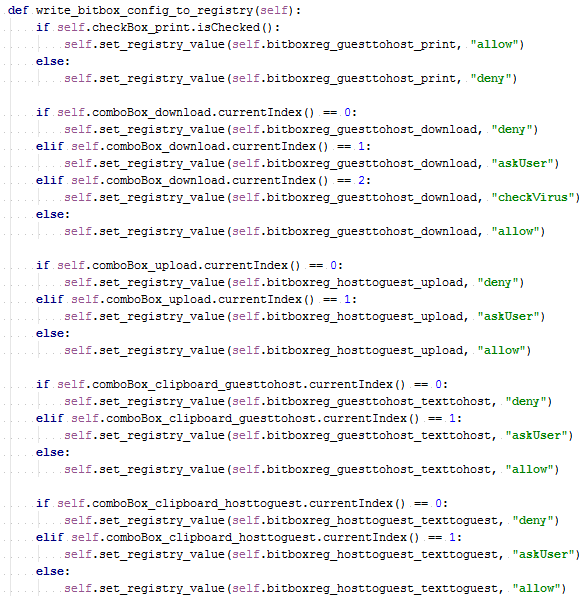


Abbildung 30: In der Registry speichern

Um den Registry Eintrag der Policy, mit dem neuen Wert zu setzen. Wird die Funktion *set\_registry\_value()* aufgerufen und als Argument der Registry Pfad übergeben, in der die Policy existiert.

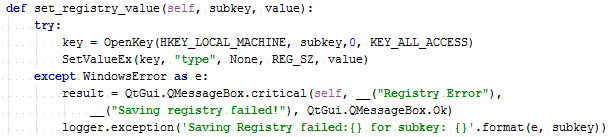


Abbildung 31: Auf Windows Registry zugreifen und speichern

Die Windows Registry lässt sich direkt von Python aus manipulieren. Python liefert dafür das Modul *\_winreg*, in der alle Klassen zur Verfügung gestellt werden, um mit der Windows Registry zu arbeiten. Die Funktion *SetValueEx*() ist die Funktion, welches das eigentliche Speichern in der Registry übernimmt. Als Argument, wird der Schlüsselpfad, Name, Typ und der eigentliche Wert übergeben. Nach dieser Methode, werden im Security Policy Manager alle Policy Einstellungen in der Windows Registry gesetzt.

Nachdem alle Security Policy Einstellung des Benutzer in der Windows Registry gespeichert wurden. Erfolgt der Aufruf der Funktion

*write\_bitbox\_config\_to\_tomini().* Hier wird das Speichern, der Netzwerkeinstellung und der Persistenten Daten in der *BitBoxTom.ini,* übernommen.



Abbildung 32: Speichern in der BitBoxTom.ini

Die Variable *parser* speichert die .ini-Konfigurationsdatei als Objekt. Das Objekt selber wird in der *get\_bitbox\_tomini()* Funktion erzeugt und als Rückgabewert zurückgegeben.

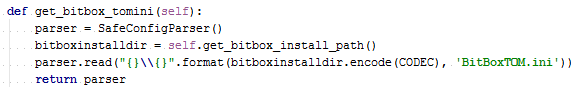


Abbildung 33: BitBoxTom.ini parsen

Für den Umgang mit .ini-Konfigurationsdateien, liefert Python das Modul *SafeConfigParser()*. Die gesamte .ini wird vom Parser ausgewertet und als Objekt für weitere Operationen zur Verfügung gestellt. In Abbildung 32 werden die neuen Einstellungen über die *set()* Funktion neu definiert. Als Argument wird die Section, Option und der Wert übergeben. In Abbildung 34 ist die Konfigurationsdatei *BitBoxTom.ini* zu sehen. Die eckige Klammer ist die Section, die Bezeichnungen links vom Gleichheitszeichen, die Option und rechts davon ist der dazugehörige Wert.

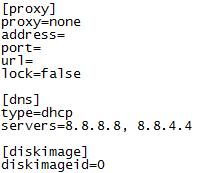


Abbildung 34: BitBoxTom.ini

Nach dem alle Konfigurationen der .ini-Datei vorgenommen wurden, muss diese gespeichert werden. Dies erfolgt im letzten Teil der Funktion. Dazu wird die .ini-Datei mit Schreibrechten geöffnet und als Objekt in der Variablen *tomini* gespeichert. Dieses Objekt wird wiederum der *write()* Funktion des *parsers* Objektes übergeben, wodurch die neue *BitBoxTom.ini* letztendlich gespeichert wird.

Läuft der Security Policy Manager im Adminmodus, so müssen die Einstellungen die vom Administrator vorgenommen wurden, in Form einer Policy Datei gespeichert werden. Die Struktur der Policy Datei, ist dieselbe wie die einer .ini-Konfigurationsdatei. Es muss während des Speicher Prozesses, eine .ini-Datei erzeugt werden.

Das Button Speichern im Adminmodus, ruft die Funktion *save\_bitbox\_config\_to\_ini()* auf. Innerhalb dieser Funktion wird zunächst ein *SafeConfigParser()* Objekt erzeugt. Mit diesem Objekt wird die Policy Datei, entsprechend der Einstellungen vom Administrator konstruiert. Da die Funktion sehr lang ist, wird die Konstruktion nur für den *Information* flow in der Abbildung 35 gezeigt. Das *SafeConfigParser()* Objekt ermöglicht die Konstruktion einer .ini-Konfiguration über die Funktion *add\_section()*. Mit dieser Funktion können die gewünschten Sections erstellt werden, denen dann wiederum die Option mit Ihren Werten zugewiesen werden können. Es wird nachdem selben Prinzip wie im Benutzermodus gearbeitet. Die Werte der GUI-Elemente werden ermittelt und anschließend der Funktion *set()* übergeben. So entsteht am Ende eine vollständige .ini-Konfigurationsdatei.



Abbildung 35: Policy Datei Konstruktion Information flow

Als letzter Schritt muss diese konstruierte .ini-Datei als *.policy-Datei* gespeichert werden. Deshalb erfolgt am Ende der Funktionsaufruf *save\_policy\_file().*

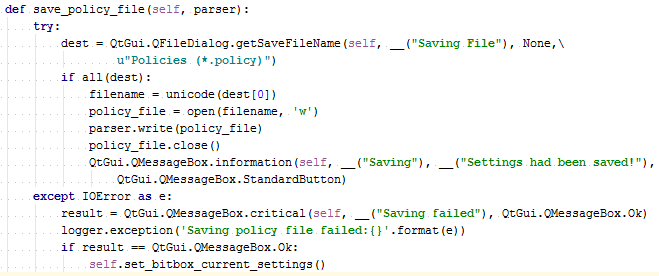


Abbildung 36: .policy-Datei speichern

Die Klasse *QFileDialog* ermöglicht, das Arbeiten mit Dialogen zum Speichern und Öffnen von Dateien. Die Funktion *getSaveFilneName()* öffnet ein Dialog zum Speichern einer Datei. Das letzte Argument, gibt dem Dialog vor, mit welcher Endung die Datei gespeichert werden soll. Da die Anforderung vorgibt eine Policy-Datei zu speichern, wird auch diese in der Form (*\*.policy)* der Funktion übergeben. Das PySide Framework legt automatisch die gewünschte Dateiendung fest. Der Anwender muss lediglich im Dialog zum gewünschten Pfad navigieren und anschließend speichern. Als Rückgabewert liefert *getSaveFileName()* den Speicherpfad zurück, welches der Anwender über das Dialog ausgewählt hat. Dieser Pfad wird der *open()* Funktion übergeben, welches als Rückgabewert das Dateiobjekt übergibt. Dieses Dateiobjekt, wird der *write()* Funktion vom *parser* überreicht, wodurch es letztendlich gespeichert wird. Die *close()* Funktion vom Parser sollte aufgerufen werden, um nicht unnötig System Ressourcen zu blockieren, welche zum Öffnen der Datei genutzt werden.

Das Laden einer Policy Datei im Administratormodus erfolgt auf ähnliche weise. Nur mit dem Unterschied, dass ein Dateidialog zum Öffnen einer Datei erzeugt wird. Der Lade Button, öffnet das Dialog über die Funktion *load\_config().*

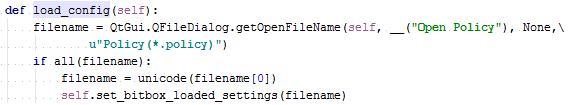


Abbildung 37: Policy-Datei öffnen

Der Hauptunterschied liegt im Aufruf der *getOpenFileName()* Funktion. Dieser Dialog, lässt den Benutzer eine Datei öffnen. Allerdings gibt diese Funktion nicht ein Dateiobjekt zurück, sondern den vollständigen Pfad zur Datei als String. Dieser Dateipfad wird der Funktion *set\_bitbox\_loaded\_settings()* übergeben.

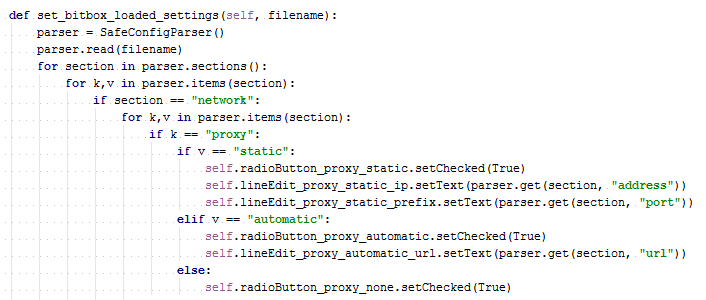


Abbildung 38: Policy Datei auswerten

Auch an dieser Stelle, wird nur ein Teil der Funktion vorgestellt, aufgrund der Länge. Der übergeben Dateipfad, wird der Funktion *read()* vom *parser* Objekt übergeben. Dieser öffnet und analysiert die Policy-Datei, mit den Funktionen *sections()* und *items()* kann man durch die Konfigurationsdatei navigieren und alle Optionen mit Ihren Werten entsprechend auswerten. In diesem Falle werden mit diesen Werten, die GUI-Elemente initialisiert.

Es wurde auf alle Hauptfunktionalitäten des Security Policy Managers eingegangen und Schrittweise durch den Quellcode geführt. Die Zusammenhänge wurden zwischen den Klassen wurde aufgezeigt und das Zusammenspiel aller Objekte wurde an praktischen Beispielen vorgeführt. Im nächsten Teil, widmen wir uns der Implementierung von besonderen Funktionalitäten. Allerdings sind diese notwendigen Lösungen für Probleme, die während der Entwicklungsphase aufgetaucht sind. Deshalb wurde diesem Abschnitt ein eigener Kapitel gewidmet.

## Komplikationen

Im Laufe dieser Arbeit, vor allem in der Entwicklungsphase, sind Komplikationen nicht ausgeblieben. In diesem Kapitel werden zwei Probleme behandelt, welche die Erfüllung der Softwareanforderungen zunächst behinderten. Zu den Problemen werden auch die Lösungen vorgestellt, mit denen die Anforderungen erfüllt werden konnten.

### Administratorrechte

Das erste Problem während der Entwicklung war, das der Security Policy Manager nicht die Administrator Zugriffsrechte bekam, um die Windows Registry sowie die *BitBoxTom.ini* zu konfigurieren. Es musste also eine Lösung erarbeitet werden, auf welchem Weg, der Anwendung Administratorrechte zugewiesen werden. Die Zuweisung der Rechte sollte dabei über die Windows Benutzerkontensteuerung (UAC) erfolgen. Mit der UAC kann sichergestellt werden, dass egal welcher Benutzer auf dem Windows System existiert, die Administratorrechte immer vergeben werden können. Andernfalls hätte man einen Benutzer mit Administratorrechten voraussetzen müssen, welcher die Anwendung mit diesen Rechten startet. Den Anwendern der BitBox vorzuschreiben, welche Rechte das Benutzerkonto haben muss, auf dem BitBox ausgeführt wird, würde der Benutzererfahrung sehr schaden. Dieses Problem bezieht sich in erster Linie auf den Benutzermodus, da hier nicht die Möglichkeit gegeben ist, den Security Policy Manager mit einem Rechtsklick als Administrator auszuführen. Die Anforderung ist, das über die BitBox TrayApp der Button konfigurieren betätigt wird, welcher wiederum den Policy Manager startet. Genau während diesem Startvorgang, soll das UAC Dialog erscheinen, und der Anwender selber soll über die UAC das Ausführen als Administrator zulassen.

Um Windows mitzuteilen, dass eine bestimmte Ausführbare Datei mit dem UAC gestartet werden soll, muss für diese Anwendung ein gesonderter Eintrag in der Windows Registry erfolgen.

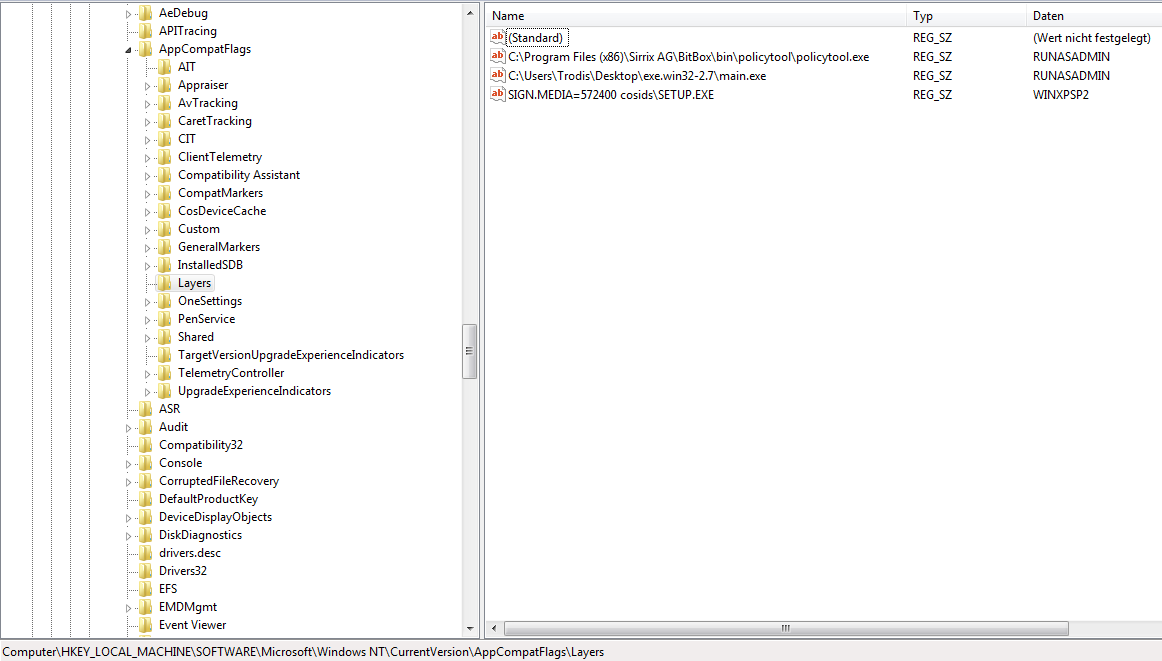


Abbildung 39: Windows Registry Benutzerkontensteuerung

Auf der Abbildung 39, rechte Seite, teilt der zweite Eintrag Windows mit, dass die Anwendung *policytool.exe,* mit der Benutzerkontensteuerung aufgerufen werden soll. Dieser Eintrag wird während der Installation der BitBox, vom Installer selber vorgenommen. Das Skript für den Installer wurde dafür entsprechend programmiert.

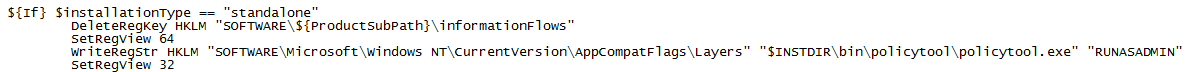


Abbildung 40: UAC Eintragung während der Installation

Das Problem war somit gelöst. Sobald der Security Policy Manager aus der BitBox heraus ausgeführt wird, wird der Endbenutzer aufgefordert, der Anwendung die Administratorrechte zu gewähren.

### TrayApp aktualisieren

Das zweite Problem ergab sich gegen Ende der Entwicklungsphase. Der Security Policy Manager war in der Lage alle Security Policy Einstellungen zu setzen und auch die *BitBoxTom.ini* zu aktualisieren, allerdings wurden die neuen Einstellungen nicht sofort von der TrayApp aktualisiert. Sobald der Benutzer die Konfiguration speichert, schließt sich der Security Policy Manager automatisch, damit der Benutzer wieder zur TrayApp gelangt. Allerdings musste auch das TrayApp Fenster geschlossen und neugestartet werden, ansonsten wurden die Einstellungen nicht aktualisiert. In Kapitel 3.4 Abbildung 6, sieht man im oberen Teil des Hauptfensters, dass die Konfigurationen der Policies aufgelistet sind. Diese galt es sofort nach dem Speichern der neuen Policy Einstellungen zu aktualisieren. Das Aktualisieren sollte in Echtzeit stattfinden, so dass der Benutzer sofort die neuen Einstellungen inspizieren kann. Andernfalls würde es zur Verwirrung beim Anwender führen, wenn noch die alten Einstellungen angezeigt werden, und die Aktualisierten erst nach einem Neustart der TrayApp.

Gelöst wurde das Problem, indem die TrayApp Anwendung um einen Watchdog erweitert wurde, welche die Security Policy Registry Einträge und die BitBoxTom.ini überwacht. Das win32api und win32event Modul erwiesen sich als äußerst nützlich für diese Aufgabe. Mit diesen Modulen, kann man auf Registry Events reagieren. Der Event sollte in diesem Falle genau dann ausgelöst werden, sobald die Policy Registry Einträge sich verändert hatten. Diese Überwachung sollte solange erfolgen, wie die TrayApp im Hintergrund ausgeführt wird.

Der erste Ansatz lag darin, diesen Watchdog in einem Python Thread laufen zu lassen. Nur ergab sich dadurch sofort das nächste Problem. Qt ist so konzipiert, das das Hauptfenster mit seinen GUI-Elementen in einem eigenen Thread läuft, wo auf Benutzereingaben gewartet wird. Versucht man nun die grafische Benutzeroberfläche, aus einem anderen Thread heraus neu zu laden, endet dieser Versuch mit einem Absturz der Anwendung.

Die eleganteste und vor allem auch sicherste Lösung war anstelle der Python Threads, den von Qt Angebotene Thread Klasse *QThread* zu nutzen. Qt schreibt nämlich vor, dass man um zur Laufzeit, die GUI manipulieren zu können, Qt seine eigene Threadklasse einsetzen muss. Andernfalls ist ein reibungsloses Zusammenspiel, nur mit sehr hohem Aufwand zu realisieren. Also wurde eine eigene QThread Klasse, in *der BitBoxTrayApp.py* implementiert, in der der Watchdog laufen soll. Hier soll noch erwähnt werden, dass die TrayApp nicht im Rahmen dieser Arbeit entwickelt wurde, sondern bereits vorhanden war im BitBox Projekt. Während der Entwicklungsphase dieser Arbeit, wurde die TrayApp nur um die hier genannte Klasse erweitert.

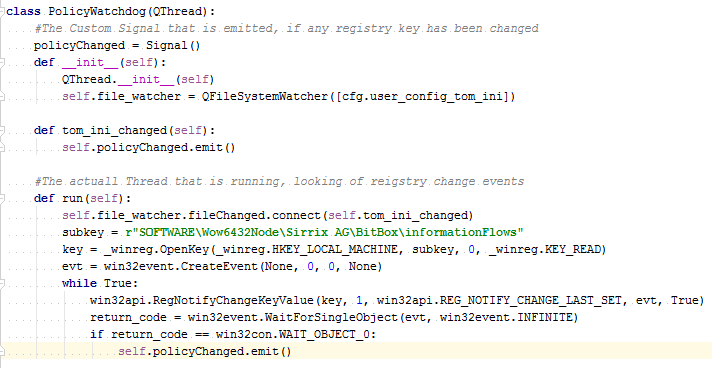


Abbildung 41: Watchdog

Die *BitBoxTom.ini* wird mit Hilfe der Klasse *QFileSystemWatcher* überwacht. Sobald die Datei verändert wurde, wird ein Signal *fileChanged* gesendet, welcher wiederum die Funktion *tom\_ini\_changed()* aufruft. Diese Funktion sendet selber ein Signal, welches selber definiert wurde am Anfang der Klasse.

Das Signal und Slot Konzept erwies sich auch hier als äußerst nützlich, um das Hauptfenster der TrayApp zu aktualisieren.



Abbildung 42: Signal und Slot für den Watchdog

Denn sobald das Signal *policyChanged* gesendet wird aus dem Thread heraus, wird die *update\_gui()* Funktion ausgeführt. Diese Funktion existierte bereits vor dieser Arbeit. Sie wurde allerdings genutzt um das Problem zu lösen, welches geschildert wurde.

Das waren die beiden Hauptprobleme während der Entwicklung des Security Policy Managers. Natürlich gab es auch andere kleinere Komplikationen. Diese betrafen allerdings nicht direkt die Anwendung, weshalb es keinen Sinn macht, auf diese einzugehen.

# Ergebnis

Das Ergebnis dieser Bachelorarbeit ist, das alle Anforderungen an den Security Policy Manager implementiert wurden und die Software auch die Erwartungen der Entwickler erfüllt. Komplikationen wurden durch professionelle Lösungen beseitig und die Anwendung konnte vollständig in die BitBox integriert werden. Was Sie nun zu einem Bestandteil der virtuellen Surfumgebung macht. Was von Anfang an auch das Ziel dieser Arbeit war.

## Benutzerfreundlichkeit

Die Benutzerfreundlichkeit hat sich mit Hilfe des Security Policy Managers immens gesteigert. Der Endanwender ist nicht mehr dazu gezwungen, die BitBox im Expertenmodus neu zu installieren. Stattdessen wird ihm nun die Möglichkeit geboten, komfortabel über den Policy Manager, die gewünschten Einstellungen vorzunehmen und diese zu speichern.

### Vorher

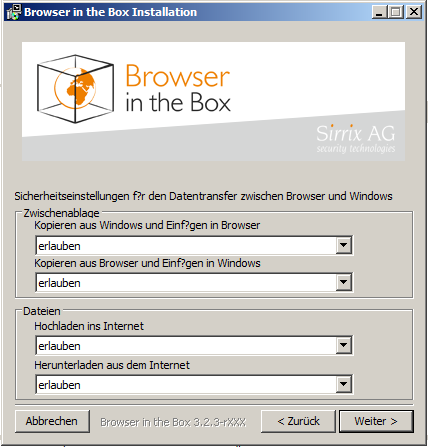
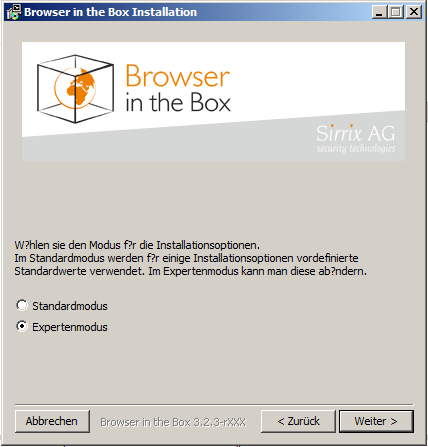
Der Benutzer musste vorher ständig die BitBox neu installieren und während der Installation den Expertenmodus auswählen, um die Security Policy Einstellungen sowie die Netzwerkeinstellungen nach seinen Vorstellungen vorzunehmen. Erfahrene Benutzer konnten die Konfigurationen der Security Policies auch über die Windows Registry manuell ändern. Dadurch kann es aber vorkommen, dass die Einstellungen fehlerhaft durchgeführt werden und am Ende die BitBox nicht mehr richtig funktioniert. Dieses Vorgehen kann unter Umständen soweit führen, dass eine Neu Installation der BitBox notwendig wird. 

Abbildung 43: Experten Modus

### Nachher

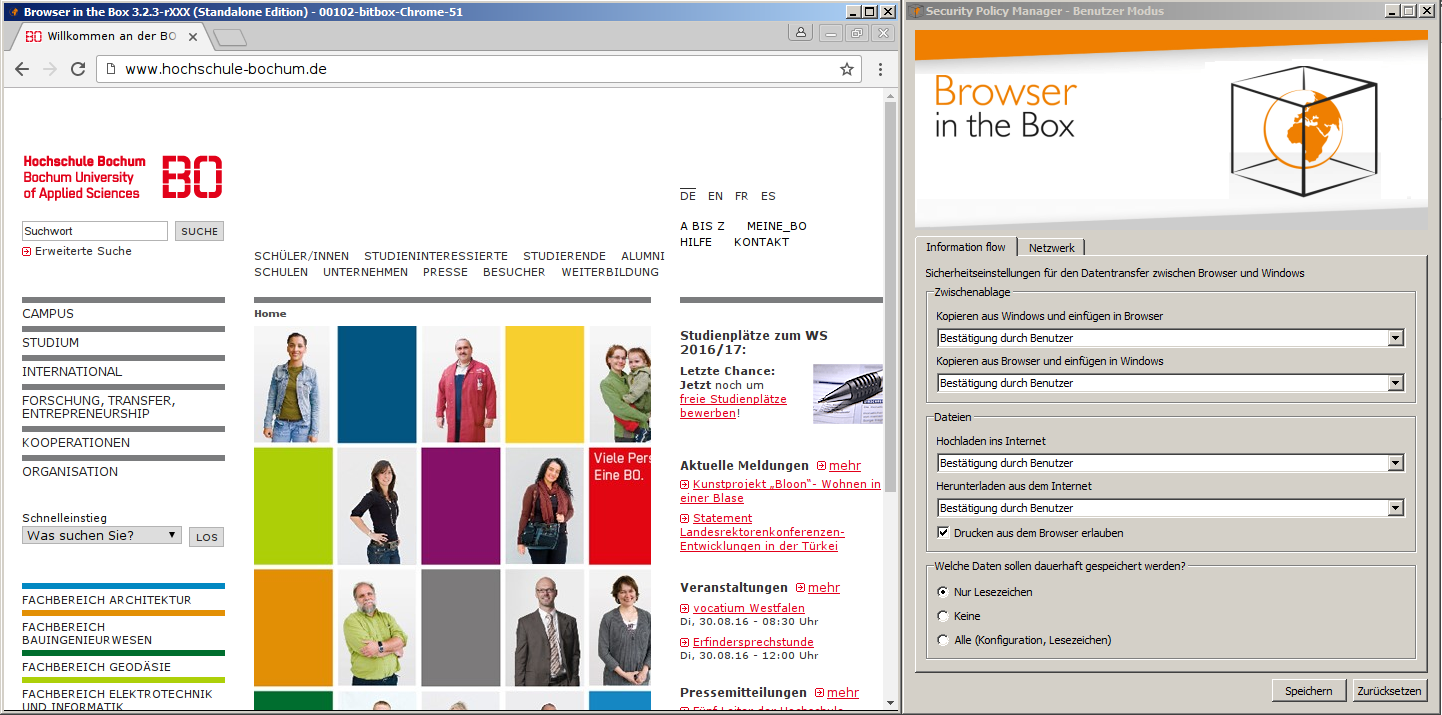


Abbildung 44: Browser in the Box und Security Policy Manager

Mit dem Abschluss diese Arbeit, hat der Anwender der BitBox, jederzeit die Möglichkeit über den Security Policy Manager, die gewünschten Einstellungen vorzunehmen. Er muss nicht mehr, manuelle Konfigurationsversuchen unternehmen und auch keine erneute Installation der BitBox durchführen, nur um neue Einstellungen zu setzen bzw. diese zu ändern.

Der Administrator hat auch die Möglichkeit über den Security Policy Manager unterschiedliche Policy Dateien zu erstellen, welche er der BitBox Installation beilegen kann. Somit erübrigt sich die Arbeit, für jede BitBox Installation auf unterschiedlichen Rechnern, immer wieder im Expertenmodus die gewollten Einstellungen zu setzen. Selbst in einem kleinen Umfeld mit fünf Rechnern, kann dies unter Umständen zu unerwünschten Konfigurationen führen. Obwohl fünf Rechner drucken dürfen, kann es nun passieren, dass der Administrator beim fünften Rechner vergisst diese Einstellung während der Installation zu setzen. Viel effizienter ist es, nur eine Policy-Datei zu erstellen und diese immer wieder für die Installationen zu benutzen.

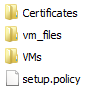


Abbildung 45: BitBoxInstallData

# Zusammenfassung

In dieser Bachelorarbeit wurde die Konzeption und Entwicklung einer Anwendung mit einer grafischen Benutzeroberfläche, für die Konfiguration der Browser in the Box Security Policies beschrieben. Beginnend mit Kapitel 1 wird die eigentlich Motivation für solch eine Anwendung beschrieben. In Kapitel 2 werden die Grundlagen sowie Begriffsdefinitionen erläutert, um den Sachverhalt in dieser Arbeit besser zu verstehen. Besonders der Begriff Security Policy wurde in diesem Kapitel behandelt, da die entwickelte Anwendung in Bezug zu diesem Begriff steht.

In Kapitel 3 wird das Endpoint Security Produkt Browser in the Box vorgestellt. Es galt aufzuzeigen, wie diese virtuelle Surfumgebung überhaupt möglich ist und welche Versionen es davon gibt. Da im Rahmen dieser Arbeit nur für die Standalone Variante entwickelt wurde, galt es aufzuzeigen, worin sich die Standalone und Managed unterscheiden und wieso ausgerechnet für die Standalone so eine Anwendung nötig war. Außerdem wurden die eingesetzten Technologien vorgestellt, mit denen das Projekt BitBox überhaupt möglich wurde. Im letzten Teil diesen Kapitels, wurde das Sicherheitskonzept vorgestellt, um aufzuzeigen welche Sicherheitsfunktionen die BitBox als Endpoint Security Produkt anbietet.

In Kapitel 4 wurde sich der eigentlichen Entwicklung gewidmet. Es galt zunächst aufzuzeigen, welche Anforderung an den Security Policy Manager gestellt wurden und welche Rahmenbedingungen erfüllt werden müssen. Die Akteure, welche die Software in erster Linie bedienen sollen wurden vorgestellt sowie die Anwendungsfälle.

Nach der Analyse der Anforderungen und Anwendungsfälle, widmeten sich die nächsten Unterkapitel der eigentlichen Implementierung. Die Kernbereiche des Quellcodes wurden dabei Schritt für Schritt erklärt. Die Strategie für das Design der Klassen war ebenfalls das Thema in diesem Kapitel. Die Komplikationen während der Entwicklung, ist der Abschluss dieses Kapitels. Um aufzuzeigen welche Probleme sich ergeben haben und mit welcher Strategie diese gelöst wurden.

Kapitel 5 dient dazu, um nochmal das Gesamtergebnis vorzustellen und welche Vorteile mit dem Security Policy Manager erreicht wurden.

# Literaturverzeichnis

[1] Mark Rhodes-Ousley: Information Security: The Complete Reference, Second Edition, Mcgraw-Hill Education Ltd; Auflage: 2. Auflage. 01. 04. 2013

[2] Mark S. Kadrich: Endpoint Security, Addison-Wesley Longman, Amsterdam; Auflage: 1, 30. 03. 2007

[3] Michael Waidner, Michael Backes, Jörn Müller-Quade, Eric Bodden, Markus Schneider, Michael Kreutzer, Mira Mezini, Christian Hammer, Andreas Zeller, Dirk Achenbach, Matthias Huber, Daniel Kraschewski: Entwicklung sicherer Software durch Security by Design, Frauenhofer Verlag, Mai 2013

[4] Norbert Schirmer, Marcel Selhorst, Christian Stüble : Eine Studie im Auftrag des Bundesamtes für Sicherheit in der Informationstechnik (BSI) Browser in the Box (BITB) Anforderungen an eine virtuelle Surfumgebung. Version 0.7. Rohde & Schwarz Cybersecurity, 15. 07. 2011

[5] Dr. Norbert Schirmer: Sicherheitskonzepte in „Browser in the Box“ (BitBox). Version 1.1 Rohde & Schwarz Cybersecurity, 13. 08. 2013

[6] Peter Bouda: PyQt und PySide. GUI- und Anwendungsentwicklung mit Python und Qt, Open Source Press, 30. 01. 2012

[7] Peter Kaiser, Johannes Ernesti: Python Das umfassende Handbuch, Galileo Computing; Auflage: 1, Dezember 2007

# Internetquellen-Verzeichnis

[1.1] <https://de.wikipedia.org/wiki/VirtualBox>

[Stand: August 2016]

[1.2] <https://de.wikipedia.org/wiki/Python_(Programmiersprache)>

[Stand: August 2016]

[1.3] PySide v.1.0.7 documentation QtGui.QMainWindow

<https://srinikom.github.io/pyside-docs/PySide/QtGui/QMainWindow.html>

[Stand: August 2016]

[1.4] System-specific parameters and functions (sys)

<https://docs.python.org/2/library/sys.html>

[Stand: August 2016]

[1.5]

# Eidesstattliche Versicherung

|  |  |  |  |
| --- | --- | --- | --- |
| Name:Ferhat |  | Vorname:Özmen |  |
| Matrikel-Nr.:008204835 |  | Studiengang:Informatik |  |

Hiermit versichere ich, Ferhat Özmen, an Eides statt, dass ich die vorliegende Bachelorarbeit mit dem Titel *Implementierung einer Grafischen Benutzeroberfläche für die Konfiguration der Browser in the Box Security Policy* selbständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinne nach anderen Werken ent­nommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Ich habe die Bedeutung der eidesstattlichen Versicherung und prüfungsrechtlichen Folgen (§ 26 Abs. 2 Bachelor-SPO bzw. § 19 Abs. 2 Master-SPO der Hoch­schule der Medien Stuttgart) sowie die strafrechtlichen Folgen (siehe unten) einer unrichtigen oder unvollständigen eidesstattlichen Versicherung zur Kenntnis genommen.

Ort, Datum Unterschrift