



Universität Bremen

Fachbereich 3: Mathematik und Informatik

Bachelorarbeit

**Erarbeitung und Umsetzung eines
Sicherheitskonzeptes für eine webbasierte
Robotik-Wissensbasis am Beispiel von openEASE**
English title: Development and implementation of a security concept for a
web-based robot knowledge service

Moritz Horstmann

Matrikel-Nr.259 007 4

3. Juni 2015

- 1. Gutachter:** Prof. Michael Beetz PhD
 - 2. Gutachter:** Dr. Karsten Sohr
- Betreuer:** Daniel Beßler, Dr. Moritz Tenorth

Moritz Horstmann

Erarbeitung und Umsetzung eines Sicherheitskonzeptes für eine webbasierte Robotik-Wissensbasis am Beispiel von openEASE

English title: Development and implementation of a security concept for a web-based robot knowledge service

Bachelorarbeit, Fachbereich 3: Mathematik und Informatik

Universität Bremen, Juni 2015

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Bremen, den 3. Juni 2015

Moritz Horstmann

Inhaltsverzeichnis

Inhaltsverzeichnis	i
1 Einleitung	1
1.1 Inhalt der Arbeit	2
1.2 Gliederung der Arbeit	2
2 Ziel und Motivation der Arbeit	3
2.1 Informationssicherheit	3
2.1.1 Vertraulichkeit	3
2.1.2 Integrität	4
2.1.3 Verfügbarkeit	5
2.1.4 Notwendigkeit von Informationssicherheit	6
2.2 openEASE	8
2.3 Ziel der Arbeit	11
3 Sicherheitsanalyse von openEASE	13
3.1 Hostsystem	13
3.1.1 Verwendetes Betriebssystem	14
3.1.2 Softwareaktualisierungen	15
3.1.3 Zugänge	15
3.1.4 Ausfall- und Datensicherung	15
3.2 Softwarekomponenten von openEASE	16
3.2.1 Docker	17
3.2.1.1 Technologie	19
3.2.1.2 Sicherheit	20
3.2.1.3 Nutzung in openEASE	21
3.2.2 Nginx-Webserver	22
3.2.3 Webrob	23
3.2.3.1 HTTP-Server	24
3.2.3.2 Flask	25
3.2.3.3 Docker-API	26

3.2.3.4	Editor	27
3.2.4	Datenbanken	28
3.2.4.1	PostgresSQL	28
3.2.4.2	MongoDB	28
3.2.5	Nutzercontainer/Knowrob	29
3.2.5.1	Datencontainer	30
3.2.5.2	Ressourcen	31
3.2.5.3	Authentifizierung	31
3.2.5.4	Prolog-Interpreter	32
3.3	Komponentendiagramm von openEASE	33
3.4	Zusammenfassung und Bewertung der Sicherheit	35
4	Konzept zur Absicherung von openEASE	37
4.1	Betriebssystem des Hosts	37
4.2	Sicherungsstrategie	38
4.3	Nginx-Webserver	39
4.4	Dockerbridge	40
4.5	Webrob	42
4.6	Nutzercontainer	44
4.7	Allgemeine Maßnahmen	47
4.8	Ausstehende Umsetzungen und Zusammenfassung	47
5	Diskussion der Arbeitsergebnisse	51
5.1	Prüfung der Transportverschlüsselung	51
5.2	Penetrationstests	52
5.2.1	Administrationsrechte auf dem Host	53
5.2.2	Zugriff auf fremde Nutzerdaten	54
5.2.3	Situation nach Behebung der Sicherheitsmängel	55
5.3	Fazit	55
5.4	Ausblick	56
A	Appendix	59
A.1	Abbildungsverzeichnis	59
A.2	Literatur	59
A.3	Liste der Abkürzungen	60
A.4	Glossar	61
A.5	Details zur Root-Sicherheitslücke	63
A.6	Inhalt des Datenträgers	64

Einleitung

Informationssicherheit gewinnt in Zeiten alles durchdringender Vernetzung und omnipräsenter Computertechnik einen immer größeren Stellenwert. Angriffe auf vernetzte Computersysteme sind allgegenwärtig und mittlerweile Teil des Alltags vieler Menschen: Sei es durch eigene Betroffenheit (infizierte Emails, versehentlich heruntergeladene Schadsoftware) oder Informationen aus Medienberichten, wie etwa der Angriff auf das Netzwerk des Deutschen Bundestages, bei dem die Rechner der Abgeordneten infiziert wurden [BRS15]. Die Motive der Angriffe sind vielfältig und häufig nicht direkt nachvollziehbar: Politische Motive, Wirtschaftsinteressen, Abneigung oder Spaß am Brechen von Sicherheitsmaßnahmen sind nur einige davon. Es ist nicht einmal erforderlich, IT-Systeme absichtlich anzugreifen, um ihren Betrieb zu stören oder gänzlich einzustellen: Durch falsche oder ungewöhnliche Bedienung von IT-Systemen, die von den Systementwicklern nicht vorhergesehen wurden, können normale Nutzer ohne böswillige Absichten ebenfalls Probleme hervorrufen, die die Verfügbarkeit und Sicherheit der Systeme beeinträchtigen. Daher ist es sehr wichtig, Computer und Software, die mit dem Internet verbunden sind, intensiv auf die Einhaltung von Grundprinzipien der Informationssicherheit zu überprüfen, unabhängig von Nutzeranzahl, Bekanntheit oder der zugemessenen Bedeutung.

Die Aufgabe dieser Arbeit besteht darin, eine bestehende Webanwendung hinsichtlich der Einhaltung dieser Prinzipien zu analysieren und ein Konzept zu entwickeln, mit welchen Maßnahmen die Sicherheit der Webanwendung verbessert werden kann. Es handelt sich bei der Webanwendung um die Robotik-Wissensbasis openEASE¹ der AG Künstliche Intelligenz (IAI) an der Universität Bremen. openEASE erlaubt die Einsicht und Befragung von Wissensdatenbanken, die z.B. während Robotikexperimenten aufgezeichnet wurden. Mit Visualisierungsfunktionen kann das Wissen des Roboters, etwa über seine Aufgabe, die Umgebung oder die Grundlage der autonom getroffenen Entscheidungen, im Internetbrowser dargestellt werden. Zur Realisierung dieser Funktionalität setzt openEASE auf Docker, eine neue Technologie zum Erstellen von Anwendungscontainern, was aus Sicht der Informationssicherheit aufgrund der nur in geringer Anzahl vorhandenen wissenschaftlichen Arbeiten und Literatur besonders interessant ist.

¹<http://www.open-ease.org>

1.1 Inhalt der Arbeit

Im Zuge der Analyse wurde openEASE auf die Einhaltung von Grundprinzipien der Informationssicherheit überprüft. Durch die Versetzung in die Rolle verschiedener Angreifer wurden Motive für verschiedene Angriffsarten gesucht und Gefährdungen durch Angriffe auf openEASE skizziert. Dabei wurden neben einigen kleinen und mittelschweren Sicherheitsmängeln, die den Betrieb und die Funktion von openEASE stören (z.B. Kapitel 3.2.4.2 auf Seite 28 und 3.2.5.4 auf Seite 32), auch mehrere sehr kritische Fehler gefunden, die etwa die vollständige Übernahme des openEASE-Servers möglich machen (Kapitel 3.2.3.3 auf Seite 26). Das Ergebnis der Sicherheitsanalyse ist eine Mängelliste (Tabelle 3.1 auf Seite 36), die als Grundlage für das folgende **Konzept zur Absicherung von openEASE** dient. In diesem Konzept ist nicht nur für jeden gefundenen Mangel ein möglicher Lösungsansatz beschrieben, um ihn zu beheben: Der überwiegende Teil der Maßnahmen wurde im Rahmen dieser Arbeit tatsächlich implementiert und in openEASE integriert. Erwähnenswerte Beispiele dafür sind die Dockerbridge (Kapitel 4.4 auf Seite 40), die Einführung von Nutzerauthentifizierung und Datenseparierung (Kapitel 4.6 auf Seite 44). Alle Beiträge des Autors zur Implementierung von openEASE können auf der Projektseite eingesehen werden².

Zum Ende der Arbeit wird der Erfolg der umgesetzten Maßnahmen durch Penetrationstests eines alten Versionsstandes von openEASE und eines Versionsstandes mit den implementierten Sicherheitsverbesserungen gemessen (Kapitel 5.2 auf Seite 52) und diskutiert, inwieweit die Maßnahmen die Sicherheit tatsächlich verbessert haben. Weitere Verbesserungsvorschläge, um die Sicherheit von openEASE auch nach Fertigstellung der Arbeit kontinuierlich zu verbessern, werden im Ausblick vorgestellt (Kapitel 5.4 auf Seite 56).

1.2 Gliederung der Arbeit

- Das Kapitel **”Ziel und Motivation der Arbeit”** auf Seite 3 gibt eine Einführung über die in dieser Arbeit behandelten Anwendungen und Technologien, sowie eine Erklärung zur Motivation, sich mit Informationssicherheit zu beschäftigen.
- **”Sicherheitsanalyse von openEASE”** auf Seite 13 enthält die Analyse von openEASE im Hinblick auf Verstöße gegen Prinzipien der Informationssicherheit.
- In **”Konzept zur Absicherung von openEASE”** auf Seite 37 erfolgt eine Vorstellung von erarbeiteten Lösungsmöglichkeiten der im vorherigen Kapitel ermittelten Sicherheitsprobleme, sowie eine Erläuterung der Umsetzungen dieser Lösungen.
- Zum Schluss werden in **”Diskussion der Arbeitsergebnisse”** auf Seite 51 die Ergebnisse dieser Arbeit diskutiert, sowie ein Ausblick auf mögliche Folgetätigkeiten gegeben.

²<https://github.com/knowrob/docker/pulls?utf8=%E2%9C%93&q=is%3Apr+author%3AAnh0rst>

Ziel und Motivation der Arbeit

In dieser Arbeit wird eine bestehende Webanwendung im Hinblick auf Informationssicherheit untersucht. Etwaige Mängel bei der Sicherheit sollen dokumentiert und durch die Implementierung von Gegenmaßnahmen behoben werden. Um die Relevanz dieser Aufgabe greifbar zu machen, wird in diesem Kapitel zunächst der Begriff der Informationssicherheit definiert und deren Wichtigkeit erläutert, sowie eine Vorstellung der zu sichernden Webanwendung gegeben.

2.1 Informationssicherheit

Die Begriffe Sicherheit oder Absicherung im Kontext dieser Arbeit beziehen sich immer auf Informationssicherheit, bzw. die Anwendung oder Implementierung selbiger. In der Informationssicherheit gibt es zwei obere Ziele, die für IT-Systeme angestrebt werden: Die Verhinderung von böswilligen oder beeinträchtigenden Aktionen gegen das IT-System und den Schutz des IT-Systems vor Ausfällen und Fehlfunktionen. Beide Ziele lassen sich durch die Berücksichtigung dreier Aspekte in IT-Systemen erreichen: Vertraulichkeit, Integrität und Verfügbarkeit (vgl. [Bis04, S.1]).

2.1.1 Vertraulichkeit

Vertraulichkeit beschreibt das Bedürfnis, Daten oder allgemein Informationen geheim zu halten. Es ergibt sich daraus, dass entweder gesetzliche Vorgaben wie das Bundesdatenschutzgesetz Vertraulichkeit erfordern, oder dass Organisationen bzw. Personen Informationen geheim halten, deren Bekanntwerden finanzielle, politische, strafrechtliche oder rufschädigende Konsequenzen hätte (vgl. [Bis04, S.2]). Im Kontext von Computersystemen ist Vertraulichkeit von besonderer

Wichtigkeit. Wenn viele Personen Zugang zum System haben, was bei einer Webanwendung potentiell allen Teilnehmern des Internets (bzw. unbestimmt vielen Nutzern) entspricht, kann die ungewollte Einsicht fremder Nutzerdaten dadurch verhindert werden. Die Internationalität des Internets ist auch ein Grund, Daten immer vertraulich zu behandeln und nur in begründeten Ausnahmefällen öffentlich zu machen, da Nutzer aus anderen Ländern mit anderen Gesetzen andere Gründe für Vertraulichkeit haben, die bei der Konzeptionierung des Computersystems nicht bekannt sind.

Um Vertraulichkeit in Computersystemen herzustellen, sind mehrere Maßnahmen denkbar. Kryptographie ist eine davon, sie sorgt bei korrekter Anwendung für eine Verschlüsselung von Daten, deren Entschlüsselung nur mit dem richtigen Schlüssel vorgenommen werden kann. Hierbei wird zwischen asymmetrischer und symmetrischer Kryptographie unterschieden: Bei symmetrischer Verschlüsselung erfolgt die Ver- und Entschlüsselung mit ein und demselben Schlüssel, bei der asymmetrischen Verschlüsselung erfolgt die Verschlüsselung mit einem anderen Schlüssel als die Entschlüsselung. Beide Verschlüsselungsarten haben eine große Bedeutung für die vertrauliche Kommunikation über das Internet. Wenn zwei Partner miteinander vertraulich kommunizieren wollen, jedoch kein gemeinsames Geheimnis besitzen, so erzeugen beide Partner einen öffentlichen und einen privaten Schlüssel und tauschen gegenseitig ihre öffentlichen Schlüssel aus. Verschlüsselt nun der erste Kommunikationspartner Daten mit dem öffentlichen Schlüssel des zweiten Partners, kann eine Entschlüsselung nur mit dem privaten Schlüssel des zweiten Partners erfolgen. Nach diesem Prinzip arbeitet SSL bzw. *Transport Layer Security (TLS)*, eine häufig im Internet genutzte Standardtechnologie zur vertraulichen Kommunikation. Weitere erwähnenswerte Beispiele für Vertraulichkeit durch Kryptographie sind Festplattenverschlüsselung (*TrueCrypt*¹, *Bitlocker*²), aber auch historische Verfahren wie die Cäsar-Chiffre (Verwendung für militärische Kommunikation im römischen Reich) oder die ENIGMA (Einsatz im Zweiten Weltkrieg durch das nationalsozialistische Deutschland).

Neben Kryptographie kann Vertraulichkeit auch durch eine Kontrollinstanz auf dem Computersystem mit den zu schützenden Informationen erreicht werden, indem die Kontrollinstanz Zugriffe auf die Informationen verhindert und nur an berechtigte Kommunikationspartner freigibt. Hier wird der Aspekt der Integrität benötigt, um berechtigte Nutzer zweifelsfrei zu identifizieren.

2.1.2 Integrität

Integrität beschreibt einerseits die Unversehrtheit/Vollständigkeit von Daten, andererseits auch die Herkunft von Daten (auch Authentizität genannt, vgl. [Bis04, S.3]). Im Kontext von Webanwendungen ist sie wichtig, um sicherzustellen, dass Daten während des Transports nicht geändert

¹<http://www.heise.de/download/truencrypt.html>

²<http://windows.microsoft.com/de-de/windows7/products/features/bitlocker>

wurden, wie es im Falle eines Übertragungsfehlers oder eines **Man-in-the-Middle (MitM)** Angriffes (Erläuterung siehe dazu Kapitel 2.1.4 auf Seite 6) geschehen kann. Es wird unterschieden zwischen Verfahren, die ungewollte Veränderungen an Daten erkennen und Verfahren, die ungewollte Veränderungen an Daten verhindern. Erstere werden z.B. zur Prüfung der Unversehrtheit von Datenträgern (CDs, Festplatten, Sicherungsbänder) oder zu Beweisführungszwecken (Buchprüfung, Beweiswerterhalt bei elektronisch signierten Dokumenten) eingesetzt, Letztere sorgen bei einer Datenmanipulation für das Zurückweisen der Daten (im Falle von **TLS** etwa ein sofortiger Verbindungsabbruch). Die Verhinderung von Datenmanipulation ist der Detektion von Manipulation vorzuziehen, da Anwender direkt darüber informiert werden und nicht erst veränderte Daten ggf. für längere Zeit unentdeckt gespeichert werden. Allerdings ist Detektion bei einer Datenmanipulation unter Umgehung der Manipulationssicherungsverfahren die einzige Gegenmaßnahme, die den Erfolg einer Datenmanipulation und damit den entstehenden Schaden durch Verwendung der manipulierten Daten dämpfen bzw. abwenden kann.

Durch Authentifizierung z.B. mittels eines gemeinsamen Geheimnisses (wie Benutzername/Passwort-Kombinationen) wird die Identität eines Nutzers und damit die Herkunft von Daten sichergestellt (sogenannte **Herkunftsintegrität**). Für Webanwendungen ist Authentifizierung ein wichtiges Werkzeug, um nutzerbezogene Dienste anzubieten, die auch nur den zum Zugriff berechtigten Nutzern zugänglich sein sollen. Auch hier gibt es unterschiedlich komplexe Ausführungen: Die einfache Authentifizierung, die nach Prüfung der Nutzerberechtigung Zugriff auf das gesamte geschützte Computersystem bietet, die rollenbasierte Authentifizierung mit Abstufungen bei den Berechtigungen (Lesen/Schreiben/Ändern, nur Zugriff auf bestimmte Ressourcen etc.), Zwei-Faktor Authentisierung mit zwei Geheimnissen auf getrennten Medien (Benutzername/Passwort und Transaktionsnummer per SMS).

Auch für die Integrität ist Kryptographie ein nützliches Hilfsmittel, erwähnenswerte Beispiele dafür sind elektronische Signaturen und Hashfunktionen (möglichst eindeutige Zuordnung eines Eingabewertes auf einen festen Ausgabewert, der für sich keine Rückschlüsse auf den Eingabewert zulässt).

2.1.3 Verfügbarkeit

Mit Verfügbarkeit von Computersystemen ist im Kontext der Informationssicherheit hauptsächlich der Schutz vor von Nutzern verursachten Systemstörungen oder -ausfällen gemeint (vgl. [Bis04, S.4]). Bei diesem Aspekt muss darauf geachtet werden, dass unvorhergesehene oder ungewöhnliche Interaktionen mit einem System keine falschen Ergebnisse produzieren, oder zur Blockade des Systems führen. Beispielsweise darf bei geteilter Nutzung eines Serversystems durch zwei Nutzer eine langlaufende, rechenintensive Anwendung des einen Nutzers nicht dazu führen, dass der andere Nutzer das System nicht mehr nutzen kann. Betrachtet man die Verfügbarkeit

von Webanwendungen, ist besonders auf **Denial of Service (DoS)** oder Distributed-DoS Angriffe zu achten, bei denen entweder durch gezieltes Provozieren von Fehlern in der Serveranwendung oder durch das Überlasten der Serveranwendung durch Absetzen vieler Anfragen von dutzenden bis hunderttausenden Computern gleichzeitig der Zugriff auf ein Computersystem absichtlich gestört wird.

Mögliche Maßnahmen zur Erhöhung der Verfügbarkeit sind die geringere Priorisierung von nutzerinitiierten Aktionen in einem Computersystem gegenüber den Prozessen, die für die Bereitstellung des Systems verantwortlich sind. Mit Programmen, die zeitliche Begrenzungen für die Ausführung eines Nutzerprozesses durchsetzen und für den Abbruch von zu lange laufenden Nutzerprozessen sorgen, sowie einer Überwachungsanwendung, die im Falle eines Absturzes die Systemsoftware wieder startet, kann sich ein System selbst von Fehlern erholen. Durch vor ein Computersystem vorgeschaltete Schutzsysteme (Firewall, **Intrusion Detection System (IDS)**) können die meisten Arten von DoS Angriffen abgewehrt werden. Großangelegte Distributed-DoS Angriffe, die über die Hardwaregrenzen eines Computersystems hinausgehen, sind allerdings deutlich schwieriger und nur mit erheblichem finanziellen Aufwand zu bekämpfen. Hier müssen sehr große Filtersysteme vorgeschaltet werden, die die Last eines Distributed-DoS Angriffes bewältigen können.

2.1.4 Notwendigkeit von Informationssicherheit

Werden Webanwendungen und Computersysteme an das Internet angeschlossen, sind sie einer steten Gefahr durch Angriffe ausgesetzt. Dieses Kapitel soll die Angriffsarten kurz erläutern, um Verständnis für die Motivation zur Absicherung einer Webanwendung zu schaffen.

Eine Angriffsart ist das Abhören von Daten im Internet. Sie erfolgt meistens passiv und kann daher nicht von den Kommunikationspartnern bemerkt werden. Innerhalb eines Netzwerkes kann ohne besondere Sicherungsmaßnahmen jeder mit physischem Zugang zu den Routern den Datenfluss aufzeichnen und mitlesen. Auf der Ebene des Internets kann die Kommunikation zwischen Netzwerken von allen beteiligten Netzwerkbetreibern und staatlichen Geheimdiensten³, in deren Einflussbereich die Netzbetreiber operieren, eingesehen werden. Aus den in Kapitel 2.1.1 auf Seite 3 genannten Gründen sollte daher die erste Überlegung bei der Sicherung einer Webanwendung sein, warum ein Kommunikationsweg explizit ohne vertraulichkeitsbildende Verfahren angeboten werden soll.

Die weiteren Angriffe zählen zu den aktiven Angriffsarten, von denen die folgenden zu den bekanntesten und häufigsten Arten gehören:

³http://www.gesetze-im-internet.de/g10_2001/___2.html, Abruf am 15. Mai 2015

- *Bruteforce*-Angriff: Ein Angreifer, der Zugriff auf ein passwortgeschütztes Nutzerkonto innerhalb einer Webanwendung erhalten will, probiert automatisiert alle möglichen Passwortkombinationen aus.
- *MitM*-Angriff: Ein Angreifer, der Kontrolle über einen Teil des Netzwerkes hat, über das kommuniziert wird, liest alle ausgetauschten Daten mit und verändert diese gezielt. Beispielsweise kann der Angreifer einem Nutzer, der ein Programm von einer beliebigen Webseite herunterladen will, das Programm auf dem Weg durch das Netzwerk durch eine Datei mit einem Computervirus bzw. einem trojanischen Pferd (Eine als nützliches Programm getarnte Schadsoftware) ersetzen. Eine Gegenmaßnahme stellt die Integritätssicherung dar: Sie gewährleistet, dass keine Daten unbemerkt verändert werden können und ermöglicht, den Benutzer zu warnen. In Kombination mit Vertraulichkeit kann der Angreifer dann nicht beobachten, was der Nutzer mit der Webanwendung kommuniziert.
- *SQL-Injektion*: Wenn die Webanwendung die Daten, welche an sie gesendet werden, nicht ausreichend validiert, kann ein Angreifer mit speziell manipulierten Daten Steuerbefehle an die Datenbank der Webanwendung absetzen. Dies ermöglicht ihm, Nutzerdaten daraus zu exportieren und zu verändern. Durch Validierung aller Daten, die in die Webanwendung aus dem Internet gelangen, kann dieser Angriff unterbunden werden, was einen übergreifenden Schutz in den Bereichen Vertraulichkeit, Integrität und Verfügbarkeit darstellt.
- *Cross-Site-Scripting (XSS)*-Angriff: Ähnlich wie bei der SQL-Injektion führt die unvalidierte Übernahme von Daten dazu, dass ein Angreifer, der einen Nutzer dazu bringt einen präparierten Link zur Webanwendung aufzurufen, im Kontext des Nutzers die Webanwendung beliebig kontrollieren kann. Die Gegenmaßnahme ist wie bei der SQL-Injektion eine vollständige Validierung der eingehenden Daten.
- *DoS*: Dieser Angriff wurde bereits im Kapitel 2.1.3 auf Seite 5 eingehend erläutert. Ein Angreifer führt ihn aus unterschiedlichen Gründen durch. Beispiele dafür sind Abneigung⁴ oder Erpressung⁵.
- *Exploits*: Hier nutzt der Angreifer einen bekannten Fehler in einem auf einem Computersystem installierten Betriebssystem oder einer Anwendung aus. Die Ausnutzung von Exploits kann zu unberechtigtem Zugang zu Daten des Computersystems, zu DoS oder sogar zur Komplettübernahme des Systems führen, bei der der Angreifer vollständige administrative Rechte auf dem System erlangt. Abwehren kann man diese Angriffe durch regelmäßiges, zeitnahes Einspielen von Programmaktualisierungen und durch Verfolgen von Mailinglisten und IT-Nachrichtenseiten, die sich mit Informationssicherheit beschäftigen.

Auch wenn ein Motiv für einen Angriff auf die eigene Webanwendung im ersten Moment nicht direkt ersichtlich ist, weil ein neutraler Inhalt angeboten wird und keine wirtschaftlichen Interessen an der Webanwendung bestehen, ist die Berücksichtigung der Informationssicherheit sehr wichtig. Ein Motiv für den Angriff beliebiger Webanwendungen besteht in der Übernahme

⁴<http://www.bbc.co.uk/news/technology-13848510>, Abruf am 15. Mai 2015

⁵<http://www.heise.de/security/meldung/Erpresser-drohen-zahlreichen-deutschen-Shops-mit-DDoS-Attacken-2650465.html>, Abruf am 16. Mai 2015

des Computersystems, auf dem die Webanwendung betrieben wird, um es für eigene Zwecke zu missbrauchen: Versand unerwünschter Emails, D-DoS-Angriffe, Anbieten urheberrechtlich geschützten Materials oder strafrechtlich relevanten Materials sind nur eine Auswahl der Missbrauchsmöglichkeiten. Es existieren zudem Programme, die alle genannten Angriffe automatisch auf beliebig vielen Computersystemen im Internet ausprobieren. Mit ferngesteuerten, infizierten Rechnern (bis zu 500 Mio. registrierte Rechnerinfektionen pro Jahr⁶) können Kriminelle große Teile des Internets automatisch nach angreifbaren Webanwendungen durchsuchen, so dass mit regelmäßigen Angriffsversuchen zu rechnen ist.

Die Absicherung von Webanwendungen dient folglich nicht nur dem eigenen Schutz und dem Schutz der Anwendungsnutzer, mit ihr wird man als Diensteanbieter der Verantwortung gegenüber den anderen Teilnehmern des Internets gerecht, nicht selber Mittäter von Angriffen oder anderen illegalen Aktivitäten zu werden.

2.2 openEASE

Die in dieser Arbeit hinsichtlich Informationssicherheit zu untersuchende und abzusichernde Webanwendung heißt openEASE⁷ und wurde innerhalb der IAI entwickelt. openEASE wurde dazu entwickelt, die während Robotikexperimenten aufgezeichneten Daten mit Nutzern zu teilen. Dabei geht es hauptsächlich um das Wissen bzw. die Wissensbasis des Roboters über seine Umgebung, die Aktionen die er während des Experimentes ausgeführt hat und welche Entscheidungen er aufgrund seines Wissens während des Experimentes getroffen hat. Durch den generischen Aufbau von openEASE sind auch andere Experimentdaten darstellbar, beispielsweise die Aufzeichnung der Bewegung eines Menschen mittels 3D-Motion-Tracking-Verfahren. Um diese Daten den Nutzern zugänglich zu machen, wurde die Möglichkeit geschaffen, mit im Browser lauffähigen Programmen die Daten zu interpretieren, zu analysieren und zu visualisieren. Bedingt durch das inhomogene Datenmaterial der verschiedenen Experimente ist eine Datenauswertung ohne Domänenwissen sehr schwierig. Zur Vereinfachung werden daher zu den Experimenten eine Sammlung von vorgefertigten Anfragen für die Auswertung zu einer einheitlich bedienbaren Oberfläche in englischer Sprache zusammengefasst.

Das Ziel von openEASE ist, die Wissensverarbeitung für autonome Robotiksysteme nachvollziehbar und verständlich zu machen, um den Einstieg in dieses Themengebiet zu erleichtern. Außerdem ist vorgesehen, dass Forscher, die an für Robotik relevanten Themen wie Perzeption oder maschinelles Lernen arbeiten, einfach die für sie relevanten Daten aus Robotikexperimenten auswerten können (vgl. [BTW15, S.1]). Menschliche Nutzer sind nicht die einzige Zielgruppe

⁶<http://www.fbi.gov/news/testimony/taking-down-botnets>, Abruf am 15. Mai 2015

⁷EASE steht für Everyday Activity Science and Engineering, vgl. [BTW15]

von openEASE: Es ist vorgesehen, dass autonome Robotersysteme über das Internet automatisiert eigene Anfragen an die Wissensbasis senden können, um Aufgaben ohne eigene Erfahrung auf Basis von Erfahrungen und Aufzeichnungen anderer autonomer Systeme lösen zu können. Denkbar ist etwa, dass in einem Experiment ein Roboter zum Backen von Pfannenkuchen programmiert wurde und dass ein anderer Roboter ohne diese Programmierung nur durch Abfragen in der Wissensbasis zu dem aufgezeichneten Experiment das Backen von Pfannenkuchen selbstständig lernt. Daher ist die Umsetzung eines programmatischen Zugriffs auf openEASE geplant [Ten+15].

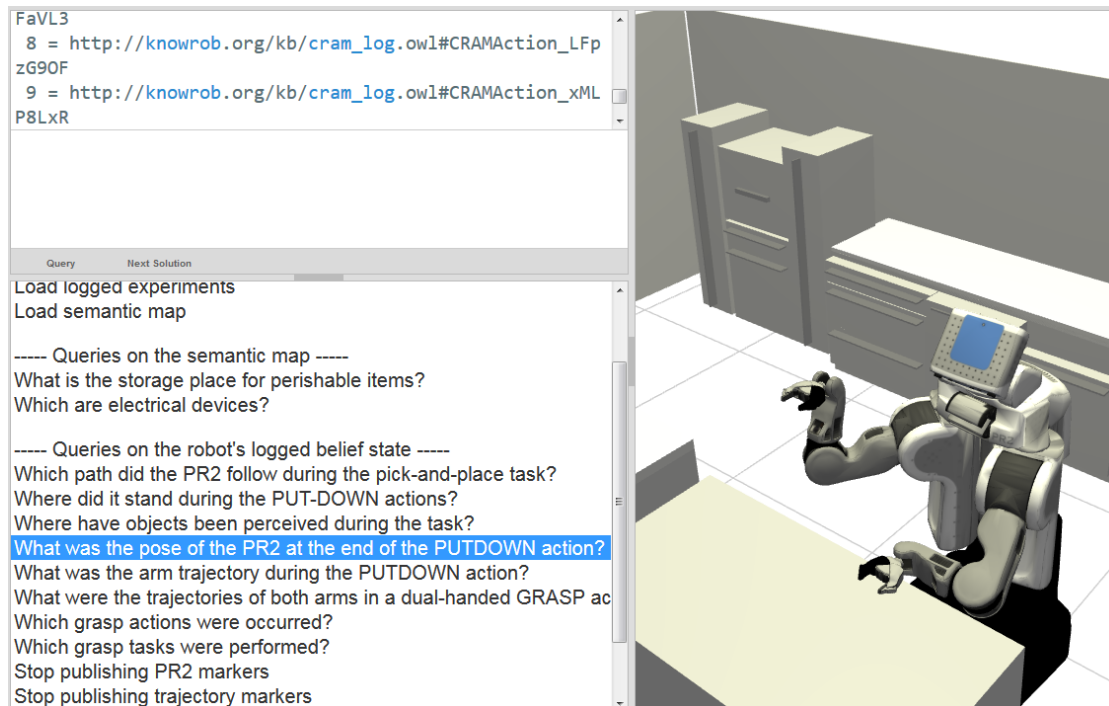


Abbildung 2.1 Die Oberfläche von openEASE

Die Auswertung der Wissensbasis in openEASE erfolgt mit der Wissensverarbeitungssoftware KNOWROB (siehe [TB13]). KNOWROB bietet für die Repräsentation und Interaktion mit Wissen aus verschiedenen Quellen eine gemeinsame Grundlage, indem es durch die Definition von Klassen und ihren Relationen eine Taxonomie vorgibt. Beispielsweise definiert KNOWROB in der Wissensbasis, dass es eine Objektklasse „Rechte Hand“ gibt, die als übergeordnete Klasse „Hand“ besitzt. „Hand“ ist wiederum die Unterklasse von „Extremität eines lebendigen Körperteils“ usw., bis als gemeinsame Oberklasse dann „Ding mit relativer Ortsangabe“ und schlussendlich „Ding“ als höchstmögliche Abstraktionsebene definiert wird. Durch Definition von Bedingungen, die für die Einordnung eines Objektes in eine Kategorie zutreffen müssen, wird es möglich, inhomogene Daten aus verschiedenen Quellen (z.B. Anleitungen im Internet, Online-Shops, Zustandsrepräsentation eines Roboters) semantisch einzuordnen und Aussagen darüber zu treffen.

Anfragen an die Wissensbasis werden als Prologausdrücke gestellt. Prolog ist eine logische Programmiersprache, die die Erfüllbarkeit von aussagenlogischen Formeln auf Grundlage einer Datenbasis prüft und bei Erfüllbarkeit die Wertzuweisungen für Variablen ausgibt, die zur Erfüllbarkeit geführt haben. Mithilfe von Fakten und Prädikaten wird die Datenbasis aufgebaut und deren Regeln definiert. Ein Beispiel für eine Datenbasis sieht so aus:

```
position(roboterhand, 1.5, 2, 0.3).
position(roboterhand2, 1.4, 1.9, 0.6).
rechte_hand(roboterhand).
linke_hand(roboterhand2).
position(ruehrbesen, 1.4, 1.9, 0.6).

% Bedingung, für ein in der Hand gehaltenes Objekt.
in_hand(Hand, Objekt) :-
    (rechte_hand(Hand); linke_hand(Hand)), % Hand muss vom Typ
    rechte_hand oder linke_hand sein.
    position(Hand, _X, _Y, _Z), % Die Position der Hand muss mit
    position(Objekt, _X, _Y, _Z), % der Position des Objektes
    übereinstimmen
    Hand \= Objekt. % Das Objekt darf nicht gleich der Hand sein.
```

Wird nun folgende Anfrage abgesetzt:

```
?- in_hand(roboterhand, ruehrbesen).
```

Ist der Rückgabewert `false`, da der Rührbesen nicht an der selben Position wie die Roboterhand ist. In der Datenbasis wurde dies nie explizit angegeben, sondern ein allgemeingültiges Prädikat `in_hand` definiert. Dies erlaubt, Belegungen für diese Bedingung zu inferieren:

```
?- in_hand(roboterhand2, Gegenstand).
Gegenstand = ruehrbesen.
?- in_hand(Hand, ruehrbesen).
Hand = roboterhand2.
```

Die unbelegte Variable `Gegenstand` bzw. `Hand` wurde durch Prolog automatisch durch Inferenz mit einem Wert belegt, durch den die Aussage wahr wird. KNOWROB liefert ähnlich zu dem Beispiel für die gemeinsame Wissensbasis vordefinierte Prädikate mit, die Anfragen daran vereinfachen und die Integration von Datenquellen in die Prolog-Datenbasis hinein übernehmen.

Diese Prologausdrücke mit KNOWROB-Prädikaten sind in openEASE für alle Auswertungen und Visualisierungen zuständig und sind für jedes Experiment durch bereits vordefinierte Fragen beispielhaft vorgegeben. Klickt ein Nutzer etwa auf „Where have objects been perceived during the task?“ im „Pick-and-Place“-Experiment (siehe Abbildung 2.1 auf Seite 9), wird ein diese Frage beantwortender Prologausdruck im Abfragefenster vorausgefüllt, welcher nur noch vom Nutzer

bestätigt werden muss. Nutzer haben dabei auch die Möglichkeit, eigene Prologausdrücke an KNOWROB zu senden. Prolog kann hierbei nicht nur für Einzelabfragen, sondern durch Laden von Prologdateien auch für komplexe Programme und eigene Auswertungen genutzt werden. Nutzern wird zu diesem Zweck ein Editor bereitgestellt, mit dem Prologdateien editiert und für den eigenen Gebrauch auf openEASE hochgeladen werden können.

Ansonsten richtet sich openEASE zwar primär an Wissenschaftler, ist jedoch für beliebige Nutzer im Internet zugänglich und soll auch Studenten zur Bearbeitung von Aufgaben in Vorlesungen bereitgestellt werden. Veröffentlichungen der IAI thematisieren openEASE mehrfach, so dass mittelfristig mit steigender Aufmerksamkeit gerechnet werden kann.

2.3 Ziel der Arbeit

Das primäre Ziel dieser Arbeit ist, die Webanwendung openEASE nach Gesichtspunkten der Informationssicherheit zu untersuchen und gefundene Schwachstellen abzusichern bzw. notwendige Schritte zur Absicherung zu dokumentieren. Auch wird in dieser Arbeit untersucht, ob die zur Entwicklung von openEASE eingesetzten Technologien die Beachtung von Prinzipien der Informationssicherheit gefördert haben, oder im Gegenteil zu zusätzlich zu berücksichtigendem Sicherheitsaufwand geführt hat.

Anhand der Ergebnisse kann für zukünftige Softwareprojekte abgeleitet werden, wo Programmierer bei der Verwendung von den untersuchten Technologien noch selbst auf Informationssicherheit achten müssen; weiterhin ergeben sich auch Verbesserungsvorschläge zur Erhöhung der Sicherheit in den Technologien selbst.

Sicherheitsanalyse von openEASE

Um eine Analyse des openEASE-Softwaresystems durchführen zu können, wurde zunächst ein Gesamtüberblick über die Komponenten der Software, sowie über die Systemumgebung gegeben. Anschließend werden einzelne Komponenten genauer untersucht, wobei das Hauptaugenmerk auf die Anwendung von Sicherheitsfunktionen der in openEASE verwendeten Software, Schnittstellen zu anderen Komponenten und die Quelltexte von Eigenentwicklungen gelegt wurde.

3.1 Hostsystem

Für diese Arbeit ist der Host (gemeint ist hier die physikalische Hardware und das Betriebssystem), auf dem openEASE installiert ist, von geringerem Interesse, da hauptsächlich die Softwarekomponenten von openEASE und deren Konfiguration untersucht werden sollen. Nichtsdestotrotz muss der Host ein Mindestmaß an Sicherheit erfüllen: Die Absicherung von openEASE ist sinn- und wirkungslos, wenn ein Angreifer sich über veraltete oder fehlerkonfigurierte Betriebssystemkomponenten Zugriff auf den Host verschaffen kann.

So ist es für die Sicherheit sehr wichtig, regelmäßig Aktualisierungen für das Betriebssystem einzuspielen. Neue Sicherheitslücken in Betriebssystemen werden üblicherweise nach Veröffentlichung einer Fehlerbehebung öffentlich bekanntgemacht. Angreifer können dann mit diesen Informationen die Sicherheitslücke bei noch nicht aktualisierten Computersystemen ausnutzen. Damit im Einklang ist auch die Vorgabe, so wenig Software wie möglich auf dem Host zu installieren und zu betreiben. Wird das Betriebssystem mit den Standardvorgaben des Herstellers installiert, werden meistens viele Programme installiert und gestartet, die für den eigentlichen Zweck des Hosts (wie den Betrieb von openEASE) nicht erforderlich sind. Jedes installierte, aber nicht benötigte Programm erhöht die Angriffsfläche und den Wartungsaufwand des Hosts. Es ist daher darauf zu achten, nur die für openEASE benötigte Software zu installieren und jede vorinstallierte Software hinsichtlich ihrer Sinnhaftigkeit zu hinterfragen.

Ein weiterer wichtiger Aspekt bei der Sicherheit des Hosts sind die Zugänge: Sowohl der physikalische Zugang zum Host, als auch Benutzeraccounts und Fernzugänge wie SSH müssen genau geprüft und beschränkt sein. Alle Sicherheitsmaßnahmen wären zum Beispiel wirkungslos, wenn beliebige Personen ohne Zugangskontrolle und Überwachung einfach die Festplatte aus dem Host entwenden oder USB-Sticks anschließen könnten¹. Benutzeraccounts sollten zum Schutz vor Brute-force-Angriffen mit einem ausreichend langen Passwort oder einem asymmetrischen Kryptographieverfahren gesichert sein. Administratorenrechte sollten niemals pauschal, sondern nur den Nutzern gewährt werden, die dies unbedingt benötigen und Erfahrung mit der Administration eines Servers haben.

Zur Sicherheit des Hosts gehört auch das Erarbeiten und Ausführen einer Datensicherungsstrategie. Datenverlust kann durch vielfältige Ursachen hervorgerufen werden und stört die Verfügbarkeit der Anwendung. Es sollte sichergestellt werden, dass in regelmäßigen Zeitabständen die Nutzerdaten auf räumlich getrennte Computersysteme gesichert werden. Umgekehrt muss auch eine Vorgehensweise dokumentiert sein, wie im Falle eines Datenverlustes die Datensicherung wiederbeschafft und eingespielt wird. Bei großen Ansprüchen an die Verfügbarkeit, etwa die Erreichbarkeit trotz Wartungsarbeiten bis hin zu Erreichbarkeit bei Ausfall durch Naturkatastrophen müssen deutlich aufwändigere Maßnahmen ergriffen werden, wie das Bereithalten von mehreren Datensicherungen oder gar betriebsbereiten Ersatzrechnern mit räumlicher Distanz (verschiedene Rechenzentren/Städte/Länder/Kontinente) zum Host.

3.1.1 Verwendetes Betriebssystem

openEASE benötigt zwingend die Funktionen eines GNU/Linux-basierten Betriebssystems und ist allgemein nur für die Verwendung damit entwickelt worden. Der Host von openEASE läuft daher auf einem GNU/Linux Ubuntu 12.04.5 LTS Desktop. Das Betriebssystem ist eine Desktopversion, was im Hinblick auf die empfohlene minimale Softwareausstattung kontraproduktiv ist. Desktop-Betriebssysteme, wie das auf dem Host installierte, haben standardmäßig sehr viele Programme für graphische Nutzeroberflächen, Internetnutzung (Browser, Email, Chat etc.) und Bürotätigkeiten vorinstalliert, die auf Serversystemen nicht benötigt werden.

Weiterhin sind Betriebssysteme für Desktops nicht für den Server- oder Dauerbetrieb ausgelegt. Sie haben Optimierungen für Energieeinsparung (z.B. Ruhezustand nach Inaktivität, Festplattenabschaltung), Multimediafunktionen (Video- und Tonausgabe) und dafür benötigte Treiber integriert, die für einen monatelangen Betrieb nicht getestet sind und zu Systeminstabilität führen können.

¹siehe dazu <https://srlabs.de/badusb/>

3.1.2 Softwareaktualisierungen

Bei einer Inspektion des Paketmanagers von Ubuntu und einer Prüfung auf verfügbare, noch nicht installierte Aktualisierungen ergab sich, dass für die auf dem System installierte Software über 250 Aktualisierungen verfügbar waren. 200 davon waren Sicherheitsaktualisierungen. Für eine unbestimmte Zeit waren also bis zu 200 Programmbibliotheken, Programme und Betriebssystembestandteile mit unterschiedlich kritischen Sicherheitslücken auf dem Host installiert und potentiell durch Angreifer nutzbar. Auf Nachfrage bei dem Administrator des Hosts wurde bestätigt, dass Aktualisierungen unregelmäßig und selten eingespielt werden, was für die Sicherheit des Hosts negative Folgen haben kann (siehe auch „Exploits“ im Abschnitt 2.1.4 auf Seite 6).

3.1.3 Zugänge

Der physikalische Zugang zum Host von openEASE ist durch eine elektronische Schließvorrichtung beschränkt. Der Server steht in einem Büro, zu dem nur Mitarbeiter der IAI Zugang haben. Dies ist als Schutzmaßnahme zunächst ausreichend, da der Autor dieser Arbeit den Mitarbeitern keine boshafte Absicht zur physikalischen Manipulation des Servers unterstellt. Hier sind eher unbeabsichtigte und unvorhersehbare Gefährdungen der Verfügbarkeit problematisch, sei es durch Stromausfall oder eine Reinigungskraft, die die Stromzufuhr zum Server für den Betrieb eines Staubsaugers unterbricht.

Laut Aussage des Hostbetreibers ist ein Umzug von openEASE auf einen in einem Serverraum betriebenen Host geplant, daher ist spätestens nach dem Umzug unter Berücksichtigung der Verhältnismäßigkeit kein Grund zur Kritik mehr vorhanden.

Zugang zum Betriebssystem in Form von Benutzerkonten ist auf die Personen Daniel Beßler und Asil Kaan Bozcuoglu als Hauptentwickler von openEASE, Daniel Nyga als Entwickler von PRAC [NB12] und Moritz Tenorth als ehemaliger Hauptentwickler von openEASE beschränkt. Alle Personen sind Mitglieder der IAI und können als vertrauenswürdig im Umgang mit administrativen Rechten betrachtet werden. Hier ist keine Gefährdung erkennbar, der Kreis der Nutzer ist hinreichend klein und ist bis auf Moritz Tenorth im selben Gebäude tätig, wodurch die Nachvollziehbarkeit von Änderungen am Server problemlos möglich ist.

3.1.4 Ausfall- und Datensicherung

Alle Verwaltungsskripte und der Quelltext der Webanwendung sind öffentlich bei der Kollaborationsplattform *github* hinterlegt², welche Sicherungen der Projektdateien und deren Versi-

²<https://github.com/knowrob/docker>

onsverwaltung pflegt. Zusätzlich haben alle Entwickler, die lokal an openEASE arbeiten, durch die dezentrale Funktionsweise der verwendeten Versionsverwaltung `git` selbst eine vollständige Kopie der Projekt-Versionsverwaltung und sind damit indirekt Teil der Datensicherung.

Die Experiment- und Nutzerdaten auf dem Host jedoch werden nicht oder nicht ausreichend gesichert. Der Administrator des Hosts hält auf seinem Entwicklungsrechner eine Kopie der Experimentdaten vor, die im Fall eines Hardwaredefektes zur Rücksicherung verwendet werden kann. Die Nutzerdaten, also die Datenbank mit Zugangsdaten und die von Nutzern erstellten bzw. generierten Daten, werden weder automatisch noch manuell gesichert und wären bei Hardwaredefekten, versehentlichem Löschen oder böswilligem Löschen bei einer Infektion mit Schadsoftware endgültig verloren.

Es fehlt außerdem eine schriftlich festgehaltene Vorgehensweise, wie bei Ausfällen vorzugehen ist und wie der Host nach einem Ausfall wieder in Betrieb genommen werden kann. Dazu müssten Hilfsmittel zur Sicherung und Wiederherstellung von Sicherungen bereitstehen und ebenfalls schriftlich dokumentiert sein. So könnte gewährleistet werden, dass der Betrieb von openEASE auch in Abwesenheit von eingeweihten Administratoren (z.B. im Krankheitsfall) aufrechterhalten wird.

3.2 Softwarekomponenten von openEASE

Die folgende Analyse der Softwarekomponenten von openEASE bezieht sich auf den Entwicklungsstand vom 16. Januar 2015³, welche damals mit genau diesem Stand öffentlich und produktiv unter <http://data.open-ease.org> betrieben wurde.

openEASE besteht im Wesentlichen aus zwei Teilen: Einerseits gibt es eine Webanwendung, welche die eigentliche Oberfläche für den Browser bereitstellt. Darin ist auch die aufbereitete Darstellung der Ergebnisse der Prolog-Anfragen enthalten. Weiterhin werden in der Webanwendung die Steuerung (also Erstellen, Starten, Stoppen und Löschen) von den KNOWROB-Wissensbasisinstanzen kontrolliert, sowie die Nutzer von openEASE verwaltet. Der andere Teil sind die KNOWROB-Wissensbasisinstanzen. Jeder openEASE-Nutzer erhält eine eigene Instanz, auf die er exklusiv Zugriff hat und an die er Prolog-Anfragen über die Webanwendungs Oberfläche senden kann.

Aufgrund der Beschaffenheit von KNOWROB ist es erforderlich, die Instanzen strikt voneinander zu trennen. KNOWROB baut intensiv auf [Robot Operating System \(ROS\)](#) auf, einem System zur Integration und Nutzung von heterogener Software in Robotern. Es ist nicht vorgesehen, mehrere ROS-Systeme parallel auf einem Host zu betreiben, so dass Virtualisierungs- oder Containerlösungen erforderlich sind, die die Instanzen voneinander abschirmen. openEASE verwendet hierzu

³<https://github.com/knowrob/docker/tree/49558dfaed562b602ff43ffcbc918b1c763910c4>

die Anwendungscontainersoftware Docker⁴. Wie im Komponentendiagramm (Abbildung 3.3 auf Seite 34) ersichtlich ist, werden sämtliche Komponenten von openEASE innerhalb von Docker betrieben, was daher auch direkten Einfluss auf die Sicherheit hat. Bevor die dargestellten Anwendungen auf nach den im Motivationskapitel (2 auf Seite 3) beschriebenen Gesichtspunkten analysiert werden, wird zunächst zur besseren Nachvollziehbarkeit Docker und dessen technische Funktionsweise erläutert.

3.2.1 Docker

Docker ist eine Software zur Erstellung, Verbreitung und Ausführung von Anwendungscontainern. Sie erlaubt es, Programme und deren Abhängigkeiten in sog. Images zu verpacken und isoliert in Containern auszuführen. Images sind Abbilder von Dateisystemen, welche aufeinander aufbauen können. So kann ein Programmierer beispielsweise ein Basis-Image für alle seine Anwendungen anlegen, in denen gemeinsam genutzte Programmbibliotheken enthalten sind. Die Images für die verschiedenen Anwendungen basieren dann auf dem Basis-Image und enthalten nur die neuen und geänderten Daten verglichen mit dem Basis-Image. Die Erstellung von Images erfolgt durch sog. „Dockerfiles“. Nach Art eines Rezeptes sind darin Informationen über den Autor und das Basis-Image enthalten, sowie eine Abfolge von auszuführenden Kommandozeilenbefehlen, die schrittweise abgearbeitet am Ende das vollständige Image ergeben. Alternativ können Images auch aus tar-Archiven oder aus bereits ausgeführten Docker-Containern erstellt werden. Erstellte Images können auf eine zentrale Plattform hochgeladen werden, von der sie wiederum heruntergeladen und aktualisiert werden können - die bekannteste Plattform ist Docker Hub⁵ von *Docker, Inc.*, der Entwicklerfirma von Docker.

Auf Basis eines Images kann dann ein Container erstellt und gestartet werden, in dem die Anwendung ausgeführt wird. Bemerkenswert ist, dass dabei die Images im laufenden Container nur lesend genutzt werden. Alle Daten, die während der Containerausführung erstellt, geändert oder gelöscht werden, werden als Differenzabbild zu den Images gespeichert. Weiterhin sind Container tendenziell kurzlebig ausgelegt, da bei Aktualisierungen der zugrundeliegenden Images (etwa bei neuen Programmversionen) der gesamte Container gelöscht und neu auf Basis des neuen Images erstellt werden muss. Die Daten des Differenzabbildes gehen dabei verloren; es muss daher ein besonderes Konzept zur permanenten Speicherung über die Containerlebensdauer hinweg angewandt werden, welches die Isolation von Containern kontrolliert aufhebt.

Die Isolation ist standardmäßig sehr umfassend, d.h. jede Containerinstanz hat ihr eigenes Dateisystem und ihr eigenes virtuelles Netzwerk. Prozesse von anderen Programmen außerhalb des Containers sind nicht sichtbar und es kann nicht mit ihnen interagiert werden. Weiterhin sind keine direkten Zugriffe auf Geräte des Hosts wie Festplatte und Grafikkarte möglich. Wird ein

⁴<https://www.docker.com/>

⁵<https://hub.docker.com/>

Container entfernt, bleiben keine Daten übrig, die während der Ausführung entstanden sind. Durch umfangreiche Konfigurationsmöglichkeiten der Docker-Container ist es jedoch möglich, Teile der Isolation zu deaktivieren. So können einzelne Dateien und Ordner des Hosts in einen Container eingebunden werden, wodurch sie bidirektional les- und schreibbar werden, sowie die Freigabe von Netzwerkports des Containers konfiguriert werden. Die Isolation kann nicht nur zwischen einem Container und dem Host, sondern auch zwischen zwei Containern aufgehoben werden, wodurch sich innerhalb von Docker Netzwerke aus Containern bilden lassen, deren Schnittstellen explizit konfiguriert werden können.

Dies ermöglicht das Konzept von Datencontainern: Um Daten aus einem Anwendungscontainer über die Lebensdauer des Containers hinweg zu behalten, wird ein zweiter Container erstellt, der nur einen geteilten Ordner enthält (im Docker-Kontext auch als „Volume“ bezeichnet). Jener zweite Container wird nun bei der Erstellung des Anwendungscontainers als eine Datenquelle angegeben. Dadurch wird der geteilte Ordner im Anwendungscontainer sicht- und beschreibbar. Wird der Anwendungscontainer später gelöscht, bleibt der Datencontainer bestehen und kann in einer neuen Instanz des Anwendungscontainers eingebunden werden, wodurch die alten Daten wieder zur Verfügung gestellt werden.

Im Vergleich zu Virtualisierungssoftware wie VMWare⁶, qemu⁷, oder VirtualBox⁸ werden die Programme in Anwendungscontainern auf dem Host-Betriebssystem ausgeführt und nicht auf Gast-Betriebssystemen, die unterhalb des Host-Betriebssystems laufen. Dabei wird viel Verwaltungsaufwand und damit Rechenleistung eingespart, da zwischen den Anwendungen und dem Host-Betriebssystem nur eine leichtgewichtige Vermittlungsschicht liegt und nicht ein virtueller Computer mit eigenständigem Betriebssystem, wie auf Abbildung 3.1 zu sehen ist.

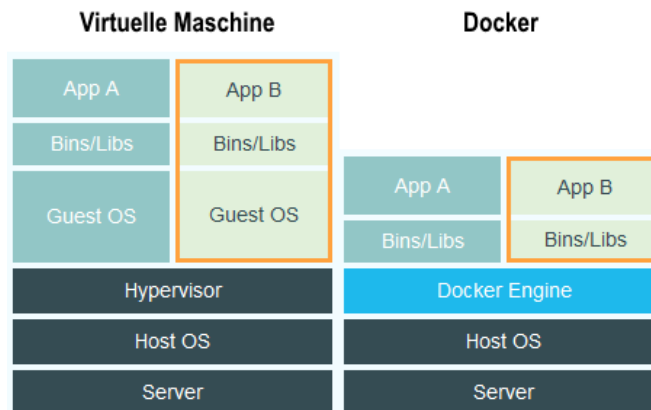


Abbildung 3.1 Unterschied zwischen klassischen virtuellen Maschinen und Docker. Quelle: <https://www.docker.com/whatisdocker/>

⁶ <https://www.vmware.com/de/virtualization/virtualization-basics/what-is-virtualization.html>

⁷ http://wiki.qemu.org/Main_Page

⁸ <https://www.virtualbox.org/>

Hier sind auch die Grenzen von Docker erkennbar: Programme, die ein bestimmtes Betriebssystem erfordern, welches nicht dem Betriebssystem entspricht, auf dem Docker betrieben wird, können nicht mit Docker genutzt werden, sondern müssen mit klassischer Virtualisierungssoftware betrieben werden.

Verwaltung und Betrieb von Docker wird vom Docker-Daemon übernommen, einer Anwendung, welche nach der Installation von Docker im Hintergrund ausgeführt wird, automatisch mit Systemstart startet und über eine Schnittstelle (Standard ist ein **UNIX-Socket**) mittels einer **JSON-API** gesteuert wird. Diese **API** wird etwa von der Kommandozeilenanwendung für Docker genutzt, kann aber auch programmatisch genutzt werden. Dazu existieren eine Reihe von Programmbibliotheken, die die Nutzung der **API** bei der Entwicklung unterstützen⁹.

Docker nutzt zur Isolation von Anwendungen Funktionen des GNU/Linux-Kernels, somit können nur unter Linux lauffähige Programme in Docker-Anwendungscontainern betrieben werden. Ebenfalls können als Host-Systeme nur Linuxsysteme verwendet werden; unter anderen Betriebssysteme wie Windows oder Mac OS kann Docker nur über eine virtuelle Maschine mit Linux als Gast-Betriebssystem verwendet werden¹⁰.

3.2.1.1 Technologie

Um Docker hinsichtlich der Bedeutung für die Informationssicherheit von openEASE beurteilen zu können, müssen zunächst noch zwei Technologien des Linux-Kernels vorgestellt werden, auf denen ein Großteil der Isolationsfunktionen von Docker basieren.

Der Linux-Kernel bietet mit *cgroups* die Möglichkeit, Gruppen von Prozessen zu bilden und diese Gruppen der Überwachung und Einschränkung des Zugriffs auf Systemressourcen zu unterwerfen. Welche Ressourcen davon betroffen sind, hängt davon ab, welche *cgroup*-Subsysteme an einer Prozessgruppe mit welcher Konfiguration zugeordnet sind. So kann etwa die Prozessornutzung, die Festplattennutzung, die Arbeitsspeichernutzung oder die Netzwerknutzung für jede Prozessgruppe anteilig festgelegt werden [Men04].

Eine weitere wichtige Technologie für Docker ist die Kernelfunktion *namespaces* (Namensräume). Ähnlich den *cgroups* nutzen sie Prozessgruppen, allerdings hierbei zum Trennen von Mounts, dem Hostnamen, der IPC-Schicht, der Prozessliste sowie dem Netzwerk (aus [Bui15, Kapitel 3.1]). Jede Prozessgruppe mit eigenem Namensraum sieht ausschließlich genannte Ressourcen, die demselben Namensraum entstammen. Daher ist es möglich, innerhalb eines Namensraums für laufende Anwendungen ein eigenes Netzwerk einzurichten, welches auch nur von diesen Anwendungen genutzt werden kann.

⁹siehe https://docs.docker.com/reference/api/remote_api_client_libraries/

¹⁰Microsoft gab im Oktober 2014 bekannt, für Windows ähnliche Isolationsfunktionen zu implementieren, so dass Docker zukünftig auch nativ unter Windows betrieben werden kann (siehe <https://msopentech.com/opentech-projects/docker/>). Damit wären auch Windows-Anwendungen in Docker lauffähig.

Ebenfalls von Relevanz ist der Einsatz der *capabilities* für Docker-Container. Standardmäßig laufen Anwendungen im Docker-Container als administrativer Nutzer (root). Dies ist notwendig, weil manche Funktionen in Linux nur durch den root-Nutzer ausgeführt werden dürfen, etwa einen Netzwerkport unterhalb von 1024 öffnen. Da in Docker-Containern diese Funktionen ebenfalls funktionieren sollen, können Anwendungen zwar als root-Nutzer laufen; sie sind jedoch mittels der *capabilities*-Technologie deutlich eingeschränkt. Capabilities erlauben es, Prozesse (genauer: einzelne Threads) mit feingranularen Rechten zu versehen, anstelle des binären Rechtes root oder nicht-root. So können Anwendungen als root-Nutzer ausgeführt werden, ohne dass sie aufgrund weitreichender Rechte aus dem Docker-Container „ausbrechen“ (Ändern/Löschen der Einschränkungen durch *cgroups/namespaces*) können.

3.2.1.2 Sicherheit

Docker als Containerlösung ist, verglichen mit den eingesetzten Technologien¹¹ noch sehr neu¹², was dazu führt, dass die Technologien durch die lange Lebensdauer deutlich länger getestet und eingesetzt wurden. Es ist auch anzunehmen, dass sie ob ihrer Eigenschaften als Werkzeuge zur Schaffung von Informationssicherheit bereits Gegenstand von gründlichen Untersuchungen und Sicherheitsprüfungen waren. Docker hingegen ist seit Veröffentlichung bis zum Zeitpunkt der Anfertigung dieser Arbeit in einer intensiven Entwicklungsphase¹³, weswegen es kaum Literatur oder Untersuchungen zur Sicherheit von Docker gibt, die den aktuellen Entwicklungsstand widerspiegeln.

Ein aktuelles Paper von Thanh Bui kommt zu dem Schluss, dass Docker durch den Einsatz oben beschriebener **Kernel**technologien ein hohes Maß an Sicherheit im Standardzustand bietet, welches zusätzlich durch den Einsatz von **Linux Security Module (LSM)** wie SELinux oder AppArmor erhöht werden könnte. Die Netzwerke in Docker-Containern sind durch nicht-standardmäßige Filterung anfällig für Flooding (DoS) oder ARP-Spoofing (Umleitung von Netzwerkverkehr über den Angreifer), was jedoch durch geeignete Firewallinstellungen relativiert werden kann. Schwieriger ist der Befund, dass Prozesse in Docker-Containern direkt mit dem **Host-Kernel** interagieren können und Systemaufrufe (syscalls) an diesen absetzen können. Gerade dieser Umstand ist mit ein Grund, warum Anwendungscontainer und damit Docker deutliche Geschwindigkeitsvorteile gegenüber anderen Virtualisierungsmethoden hat. Die Befürchtung des Autors jedoch ist, dass die Isolation der Container nicht ausreichend sein könnte und durch geschickte Kombination von Interaktionen mit nicht-isolierten **Kernelressourcen** ein Ausbruch aus dem Container möglich wäre (siehe [Bui15, Kapitel 5]).

¹¹ *capabilities* etwa wurden mit der Linux-Kernelversion 2.2 Anfang 1999 eingeführt

¹² Erstveröffentlichung im März 2013, siehe <https://www.docker.com/company/aboutus/>

¹³ Allein vom 13. bis zum 20.05.2015 haben 64 Autoren 185 Änderungen am Quelltext von Docker vorgenommen, was zu ca. 26.000 zusätzlichen und 11.000 gelöschten Textzeilen geführt hat (aus <https://github.com/docker/docker/pulse>).

Während der Erstellung dieser Arbeit wurden außerdem zwei Möglichkeiten entdeckt, wie mithilfe von Docker ein Angreifer, der Zugang zu einem Benutzerkonto auf dem Host hat, vollständige und unbeschränkte Administratorenrechte auf dem Hostsystem erlangen kann^{14,15}. Beide Angriffe erfordern, dass das Benutzerkonto Mitglied der „docker“-Nutzergruppe ist, also die Berechtigung hat, Docker-Container zu erstellen und zu kontrollieren. Im Verlauf beider Angriffe werden dann Ressourcen des Hosts in einen Container eingebunden und innerhalb des Containers mit root-Rechten gelesen/verändert. Tatsächlich birgt die Berechtigung, Docker zu kontrollieren, erhebliche Sicherheitsrisiken, da ohne Sicherheitsmechanismen wie `sudo` indirekt Zugriff auf alle administrativen Funktionen gewährt wird. Überraschend sind die Entdeckungen beider Möglichkeiten jedoch nicht, heißt es doch seit Erstellung der Docker-Dokumentation auf der Seite *Docker Security*¹⁶: „First of all, only trusted users should be allowed to control your Docker daemon. [...] you can start a container where the `/host` directory will be the `/` directory on your host; and the container will be able to alter your host filesystem without any restriction. This is similar to how virtualization systems allow filesystem resource sharing. Nothing prevents you from sharing your root filesystem (or even your root block device) with a virtual machine.“ Das Sicherheitsrisiko ist folglich bekannt und dokumentiert; es folgt daraus nur, dass das Recht zur Steuerung von Docker genau so restriktiv vergeben werden darf wie Administratorenrechte. Außerdem müssen alle Programme, die Gebrauch von der Docker-API machen, besonders aufmerksam geprüft und abgesichert werden, so dass sie nicht durch fehlerhaftes Verhalten direkten Zugriff auf die API von außen ermöglichen.

Zusammenfassend ist die Sicherheit von Docker durch die Exponiertheit des Host-Kernels in den Containern zwar geringer als bei klassischen Virtualisierungstechnologien, jedoch immer noch ausreichend für die Anforderungen von openEASE. Eine konkrete Möglichkeit zum Ausbruch aus einem Container ist derzeit nicht bekannt und würde durch die hohe Aufmerksamkeit, die Docker genießt, schnell bekannt und behoben werden. In Abwägung mit den Vorteilen von Docker (Leichtgewichtigkeit und einfache, konfigurationsfreie Auslieferung von Anwendungen) sind die Sicherheitsbedenken daher eher gering und vertretbar.

3.2.1.3 Nutzung in openEASE

Docker erfüllt bei openEASE hauptsächlich zwei Zwecke: Der erste, bereits in der Einleitung dieses Kapitels (3.2 auf Seite 16) erwähnte Zweck ist, mehrere Instanzen der Wissensverarbeitungsanwendung KNOWROB parallel betreiben zu können. Im Detail ist das von KNOWROB verwandte ROS (genauer: Der zentrale Verwaltungsprozess *roscore*) darauf ausgelegt, genau einmal auf einem Computer gestartet zu werden. Dabei wird das interne Netzwerk des Computers intensiv

¹⁴ <https://www.andreas-jung.com/contents/on-docker-security-docker-group-considered-harmful>, Abruf am 19.05.2015

¹⁵ <http://reventlov.com/advisories/using-the-docker-command-to-root-the-host>, Abruf am 19.05.2015

¹⁶ <https://docs.docker.com/articles/security/>, Abruf am 18. April 2015

zur Prozesskommunikation verwendet. Docker kann hier durch die Isolation der Netzwerkschicht mehrere Instanzen von KNOWROB jeweils in einem Container ausführen, wobei jede Instanz ihr eigenes internes Netzwerk besitzt. Der zweite Zweck von Docker ist die einfache Einrichtung von openEASE: Wie im Komponentendiagramm (Abbildung 3.3 auf Seite 34) sichtbar ist, besteht die Anwendung aus vier Anwendungscontainern plus einer variablen Anzahl an Containern mit KNOWROB-Instanzen. Zusätzlich haben alle Container Zugriffe auf verschiedene Datencontainer. Das gesamte openEASE-System wird in Docker betrieben und ist vollständig darin integriert. Diese gesamte, komplexe Integration des Systems sorgt dafür, dass der Aufwand zur initialen Einrichtung von openEASE sehr gering ist: Zur Einrichtung auf einem beliebigen System müssen nur Docker installiert, und zwei Skripte aus dem Projekt-Versionsverwaltungssystem ausgeführt werden. Diese enthalten einfache Anweisungen zum Herunterladen der benötigten Images und der Erstellung und Ausführung von Containern. Außer Docker und den Datenstrukturen, die Docker für die Speicherung von Images und Containern auf dem Host speichert, werden keine weiteren Programme benötigt. Eine gesonderte Konfiguration ist ebenfalls nicht weiter erforderlich.

3.2.2 Nginx-Webserver

Der *nginx*-Webserver als Komponente von openEASE dient der Verteilung von Nutzeranfragen an die Webanwendung und an die einzelnen KNOWROB-Instanzen. Zu diesem Zweck wird auf dem Netzwerkport 80 (durch Portfreigabe in Docker auch auf dem Host an allen Netzwerkgeräten) auf HTTP-Anfragen gewartet und diese je nach URL an *Webrob* (siehe auch das folgende Kapitel 3.2.3 auf Seite 23) oder eine KNOWROB-Instanz weitergeleitet. Da jeder Nutzer von openEASE seine eigene KNOWROB-Instanz besitzt, wird die korrekte Instanz, an die eine Nutzeranfrage weitergeleitet werden soll, anhand des Nutzernamens in der URL erkannt. In der Praxis wird dies so realisiert, dass alle Anfragen, die auf das Schema `/ws/<Nutzername>` passen, an die Schnittstelle der KNOWROB-Instanz des Nutzers `<Nutzername>` umgeleitet wird. Alle anderen Anfragen, etwa `/user/sign-in` für die Anmeldeseite von openEASE werden an *Webrob* weitergeleitet.

Die Umleitung von Anfragen zu den KNOWROB-Instanzen ist keine triviale Aufgabe, da sie dynamisch mit jedem An- und Abmelden von Nutzern in Form von neuen Docker-Containern erstellt und gestartet bzw. gestoppt und entfernt werden. Die Lösung dieses Problems ist *docker-gen*: *docker-gen* nutzt die Eigenschaft, dass das Starten und Stoppen neuer Container über die Docker-API verfolgt werden kann. *Docker-gen* erzeugt so bei jedem Starten/Stoppen von KNOWROB-Nutzercontainern einen entsprechenden Umleitungseintrag in der Konfiguration von *nginx*, was durch die Konvention, die Namen der Docker-Container mit den KNOWROB-Instanzen den jeweils zugehörigen Benutzernamen entsprechen zu lassen, nach dem oben genannten Schema möglich ist. Speziell für diese Aufgabe wurde das Projekt *nginx-proxy* von Jason Wilder¹⁷ ko-

¹⁷<https://github.com/jwilder/nginx-proxy>

piert und zur gesonderten Behandlung der KNOWROB-Nutzercontainer umgeschrieben¹⁸. *Nginx* und *docker-gen* werden dabei selber in ein Docker-Image eingebettet und können von Docker Hub heruntergeladen werden¹⁹.

Ein offensichtlicher Mangel des Webservers ist die inaktive Verschlüsselung. Dadurch, dass nur unverschlüsseltes HTTP auf Port 80 akzeptiert wird, ist es Nutzern nicht möglich, eine vertrauliche Verbindung zu openEASE aufzubauen und die Identität des Servers zu prüfen. Unscheinbar aber sehr schwerwiegend ist weiterhin die Kombination zweier Schwachstellen: Da der *nginx*-Webserver dynamisch von *docker-gen* konfiguriert wird und *docker-gen* dazu die Docker-API nutzt, muss der UNIX-Socket des Docker-Daemon in den Container des *nginx*-Webservers eingebunden werden. Wie in Kapitel 3.2.1.2 auf Seite 20 beschrieben, ist der Zugriff auf die Docker-API bedenklich, da sie indirekt Administratorenrechte auf dem Host bietet. Dazu kommt noch, dass durch das Kopieren des Originalprojektes *nginx-proxy* die Pflicht zum Aktualisieren des Docker-Images bei Erscheinen neuer (Sicherheits-)Aktualisierungen für *nginx* auf die Entwickler von openEASE übergegangen ist. Dies wurde unterlassen²⁰, was dazu führen kann, dass Angreifer durch derzeit existierende oder zukünftige Sicherheitslücken in *nginx* Kontrolle über den Container erlangen können. Die Kontrolle würde sich dabei auch auf *docker-gen* und den Zugang zur Docker-API erschließen, was eine vollständige Kompromittierung des Hostsystems möglich macht.

3.2.3 Webrob

Die Komponente *Webrob* stellt die graphische Oberfläche von openEASE bereit, über welche die Anfragen zu Robotikexperimenten abgesetzt werden. Die eigentliche Kommunikation mit KNOWROB-Nutzercontainern und die Visualisierung von Anfragen findet dabei direkt im Browser statt und nicht auf dem openEASE-Host - etwa 45 JavaScript-Bibliotheken werden hierzu eingesetzt. Außerdem enthält *Webrob* eine integrierte Benutzerverwaltung, die Steuerlogik zur Kontrolle von KNOWROB-Nutzercontainern sowie Hilfsfunktionen für die Oberfläche - etwa eine Historie von Prolog-Anfragen, eine Experimentverwaltung und einen Editor zum Bearbeiten von Prologmodulen.

Die Programmlogik von *Webrob* ist in der Programmiersprache Python unter Verwendung des Web-frameworks Flask²¹ geschrieben, die graphische Oberfläche zur Verwendung im Browser entsprechend in HTML, CSS und JavaScript. Dabei wird das in Flask unterstützte Vorlagensystem *Jinja2* genutzt, um dynamische Inhalte aus Python in die Oberfläche einzubetten. *Webrob* wird in einem eigenen Docker-Container ausgeführt und hat Zugriff auf die Docker-Datencontainer

¹⁸ <https://github.com/knowrob/nginx-proxy>

¹⁹ <https://registry.hub.docker.com/u/knowrob/nginx-proxy/>

²⁰ Letztes Image vom 24. November 2014, siehe https://registry.hub.docker.com/u/knowrob/nginx-proxy/builds_history/63597/ (Abruf am 20. Mai 2015)

²¹ <http://flask.pocoo.org/>

„user_data“ und „knowrob_data“, welche die in den KNOWROB-Nutzercontainern nutzbaren Experiment- und Nutzerdaten enthalten (Erläuterung im folgenden Kapitel 3.2.5.1 auf Seite 30). Um die Steuerung von KNOWROB-Nutzercontainern zu ermöglichen, erhält der Container von *Webrob* Zugriff auf den Docker-API-UNIX-Socket. Über das Netzwerk kann auf die Postgres-Datenbank (siehe Kapitel 3.2.4.1 auf Seite 28) zugegriffen werden, *Webrob* selbst stellt über den Port 5000 einen in Flask integrierten HTTP-Server bereit, welcher für die Auslieferung aller Webseiten (also graphische Oberfläche und Verwaltungsseiten) von openEASE zuständig ist.

Die Benutzerverwaltung wird durch das Flask-Modul *flask-user* bereitgestellt und sorgt durch einen einfach zu nutzenden Mechanismus (Annotation von Methoden) dafür, dass alle schützenswerten Seiten von openEASE erst nach erfolgreicher Anmeldung eines Nutzers zugänglich sind. Der Registrierungsprozess sowie der An- und Abmeldeprozess werden vollständig durch *flask-user* übernommen, ebenso wie die Kommunikation mit der Postgres-Datenbank. Passwörter werden durch das Modul als Hashwert gespeichert.

Bei der Analyse von *Webrob* sind mehrere, teilweise sehr ernste Schwachstellen erkannt worden, die in den folgenden Abschnitten beschrieben werden.

3.2.3.1 HTTP-Server

Die Verwendung des in Flask integrierten HTTP-Servers zum Betrieb in einer Produktivumgebung wird von den Entwicklern von Flask ausdrücklich nicht empfohlen. Er ist laut Dokumentation von Flask nur zu Testzwecken während der Entwicklung (development server) vorgesehen²². Grund dafür ist die einfache, serielle Abarbeitung von Anfragen, was während der Entwicklung Seiteneffekte durch Parallelisierung eliminiert, im Produktiveinsatz bei mehreren gleichzeitigen Nutzern von openEASE jedoch zu Geschwindigkeitseinbußen führt. Tatsächlich konnte in einem im Rahmen dieser Arbeit durchgeführten, simulierten Lasttest durch Herunterladen einer ca. 600 Megabyte großen Datei der Zugriff auf den Server vollständig blockiert werden. Außerdem wurde der integrierte Server nicht für den dauerhaften Betrieb ausgelegt und getestet, das Verhalten während hoher Last oder während eines DoS-Angriffes wurde auch nicht getestet. Es ist möglich, dass der Server während oder nach einer solchen Überlastsituation durch einen Fehler einfach abschaltet. Damit ist der Einsatz des integrierten Servers ein Mangel hinsichtlich der Verfügbarkeit des Systems, was mindestens die Ladezeit für die Nutzer und schlimmstenfalls die gesamte Erreichbarkeit beeinträchtigen kann.

Der Umstand, dass der Port 5000 des HTTP-Servers durch Docker auf dem Host auf allen Netzwerkgeräten geöffnet und weitergeleitet wird, sorgt zudem dafür, dass auf dem öffentlichen Netzwerk die Oberfläche von openEASE einmal über den *nginx*-Webserver auf Port 80 und einmal direkt auf Port 5000 erreichbar ist. Dies ist zwar nicht direkt ein Sicherheitsproblem, sorgt aber

²²<https://github.com/mitsuhiko/flask/blob/3d3b809347c98ec827687ed3fd6f9f935261536f/docs/deploying/index.rst>

für die vermeidbare Öffnung einer zweiten Schnittstelle zum System. Nach dem von Saltzer und Schroeder beschriebenen „Economy of mechanism“-Prinzip müssen Sicherheitsmechanismen in Software so klein und einfach wie möglich gestaltet sein, um durch bessere Nachvollziehbarkeit die Prüfung selbiger zu vereinfachen (siehe [SS75, 1.A.3.a.]).

3.2.3.2 Flask

Da Flask als **Mikroframework** beworben wird, soll der Aufwand für den Betrieb von Flask so gering wie möglich sein. Dies scheint die Ursache für die Entscheidung gewesen zu sein, Sitzungsdaten auf dem Client zu speichern: Sitzungsdaten sind für Webanwendungen notwendig, um während der Nutzersitzung (also der Gesamtdauer eines Webseitenbesuchs) den Zustand zu sichern. Beispielsweise werden in Sitzungsdaten teilweise ausgefüllte Formulare oder der Anmeldestatus eines Nutzers gesichert. Üblicherweise werden diese Sitzungsdaten auf dem Server gespeichert und einem eindeutigen Bezeichner zugeordnet, welcher dem Nutzer als Cookie oder Anfrageparameter während der Sitzung bei jeder Anfrage übermittelt und dem Server die Rückzuordnung zu den zugehörigen Daten ermöglicht. Die eigentlichen, in der Sitzung gespeicherten Daten werden nie an den Nutzer übermittelt und bleiben auf dem Server. Flask verwendet eine andere Methode: Hier werden alle Sitzungsdaten als Cookie an den Nutzer gesendet, um den Aufwand für die serverseitige Sitzungsverwaltung zu vermeiden. Um zu verhindern, dass Nutzer die Sitzungsdaten in den Cookies verändern, werden sie mit einem geheimen Schlüssel kryptographisch mit einem symmetrischen Verfahren (HMAC mit SHA-1) signiert. Ändert der Nutzer nun die Daten, stimmt die Signatur bei der Prüfung nicht mehr mit den Daten überein; die Sitzung wird verworfen.

In zweierlei Hinsicht ist diese Praxis problematisch: Ein Nutzer kann den Inhalt der Sitzungsdaten einsehen, da sie nur signiert, aber nicht verschlüsselt sind. Kennt ein Nutzer außerdem den geheimen Schlüssel, kann er die Sitzungsdaten manipulieren und die Signatur selber korrekt für diese Daten erzeugen. Der Server ist dann nicht in der Lage, die Manipulation zu erkennen und akzeptiert sie. Letzteres ist bei openEASE aufgetreten, der geheime Schlüssel wurde fest in eine der öffentlich zugänglichen Konfigurationsdateien im Projekt-Versionskontrollsystem eingetragen²³ und auf dem Produktivsystem nicht geändert. Dies stellt einen erheblichen Mangel für die Integrität der Sitzungsdaten dar. Angreifer können selber ein Sitzungs-Cookie konstruieren, welches sie als beliebigen angemeldeten Nutzer identifiziert. Das Konzept der clientseitigen Speicherung von Sitzungsdaten, wie es durch Flask praktiziert wird, stellt sich angesichts der gefundenen Sicherheitsmängel als höchst fragwürdig dar. Wenn ein Entwickler zur korrekten und sicheren Speicherung von Sitzungsdaten zunächst die genaue Implementierung der Verwaltung von Sitzungsdaten im **Webframework** nachvollziehen muss, dabei grundlegendes Verständnis

²³<https://github.com/knowrob/docker/blob/49558dfaed562b602ff43ffbc918b1c763910c4/webrob/webrob/config/settings.py#L4>, Abruf am 21. Mai 2015

vom kryptographischen Signieren von Daten haben sollte, um dann selber aufgrund der fehlenden Funktion zum einfachen Tauschen des geheimen Schlüssels in der Produktionsumgebung eine solche Funktion nachimplementieren muss, trifft bei einem Versagen der Sicherheit den Entwickler höchstens eine Teilschuld.

Eine weitere Schwachstelle ist die in Flask integrierte Debug-Konsole. Ist sie aktiviert, wird bei jedem Programmfehler statt einer Fehlermeldung im Browser eine vollwertige Python-Konsole im Kontext der Flask-Webanwendung geöffnet. In ihr können beliebige Python-Befehle eingegeben werden, die in der laufenden Webanwendung ausgeführt werden. Während der Entwicklung ist diese Funktion sehr nützlich, um Fehlerursachen direkt bei Auftreten des Fehlers zu finden. In der Produktion ist diese Funktion hochgefährlich, da jeder Angreifer, der einen Programmfehler provozieren kann, Vollzugriff auf den Python-Prozess bekommt und mit seinen Rechten agieren kann. In openEASE ist die Debug-Konsole gleich an zwei Stellen aktiviert worden^{24,25} und wurde im Produktivbetrieb auch nicht ausgeschaltet.

Eigentlich würde der Umstand, dass *Webrob* in einem eigenen Docker-Container ausgeführt wird, dafür sorgen, dass ein Angreifer durch Nutzung der Debug-Konsole nur innerhalb des Containers Kontrolle ausüben kann. Kombiniert mit der nächsten Schwachstelle ist diese Lücke jedoch deutlich weitreichender und der erste Teil der schwersten Sicherheitslücke, die im Rahmen dieser Arbeit gefunden wurde.

3.2.3.3 Docker-API

Wie zuvor mehrfach erwähnt, nutzt *Webrob* zur Steuerung der KNOWROB-Nutzercontainer die Docker-API. Mittels der Programmbibliothek *docker-py* wird bei jedem Anmeldevorgang eines Nutzers bei openEASE ein neuer Nutzercontainer erstellt und gestartet und bei jedem Abmeldevorgang wieder gestoppt und entfernt. Der direkte Zugang zur API hat zusammen mit der letzten Schwachstelle zur Folge, dass ein Angreifer über die Debug-Konsole direkten und ungefilterten Zugang zur Docker-API erhält. Im Kapitel 3.2.1.2 auf Seite 20 wurden die Folgen einer Möglichkeit für einen Angreifer, auf die Docker-API zuzugreifen, bereits erörtert. Eine Demonstration, wie ein Angreifer vollständige Administrationsrechte auf dem Host durch diese beiden Schwachstellen erlangt, ist im Kapitel 5.2.1 auf Seite 53 beschrieben, was eine weitere Erklärung der Ernsthaftigkeit des Problems erübrigt.

Neben der kritischen Schwachstelle birgt die Steuerung der Nutzercontainer selbst noch eine Gefahr für die Verfügbarkeit von openEASE. Melden sich viele Benutzer gleichzeitig bei openEASE an, wird für jeden Benutzer ein Nutzercontainer gestartet, ohne dass in der Steuerung eine Grenze

²⁴ <https://github.com/knowrob/docker/blob/49558dfaed562b602ff43ffcbc918b1c763910c4/webrob/webrob/config/settings.py#L3>

²⁵ <https://github.com/knowrob/docker/blob/49558dfaed562b602ff43ffcbc918b1c763910c4/webrob/runserver.py#L13>

für die maximale Anzahl gleichzeitig laufender Container vorgegeben ist (siehe dazu ergänzend Kapitel 3.2.5.2 auf Seite 31). Die Container werden beendet, wenn sich die jeweils dazugehörigen Nutzer von openEASE über den in der Benutzeroberfläche vorhandenen Link „Logout“ abmelden. Viele Nutzer versäumen dies jedoch, ohne dabei böse Absichten zu verfolgen, was zu entsprechenden Mengen an ewig laufenden Nutzercontainern führt. Es fehlt eine automatische Überwachung der Nutzeraktivität in dem Container, die auch ohne das explizite Abmelden eines Nutzers verwaiste Container stoppen und entfernen kann. Laut Aussage des Hostbetreibers müssen aus diesem Grund regelmäßig händisch alle übriggebliebenen Nutzercontainer aufgeräumt werden.

3.2.3.4 Editor

Der Editor bietet den Nutzern von openEASE die Möglichkeit, eigene Module (in Form von ROS-Packages) zu schreiben und damit die Visualisierungs- und Analysefunktionen von openEASE selbst zu erweitern. Die Funktionen des Editors umfassen das Anlegen von neuen Modulen, das Erstellen, Hochladen, Bearbeiten und Löschen von Dateien, sowie das Herunterladen und Löschen ganzer Module. Um den Anfang der Modulentwicklung und -integration zu erleichtern, werden bei Neuerstellung eines Moduls Beispieldaten hinzugefügt. Die Module der Nutzer werden hierbei im Datencontainer `user_data` in dem Ordner `/home/ros/user_data/<Benutzername>` gespeichert, ein Modul mit dem Namen „pfannkuchenexperiment“ des Nutzers „mhorst“ würde also im Ordner `/home/ros/user_data/mhorst/pfannkuchenexperiment` gespeichert werden.

Alle Funktionen, die der Editor bereitstellt, werden über asynchron ablaufende HTTP-Anfragen mit JavaScript über die Oberfläche von openEASE an entsprechende Methoden im Python-Code weitergegeben. Klickt etwa ein Nutzer auf eine Prologdatei innerhalb eines Moduls, wird eine HTTP-Anfrage zur Beschaffung des Inhalts der Prologdatei an *Webrob* geschickt. Bei allen derartigen Anfragen findet keine Überprüfung oder sonstige Validierung der Eingabeparameter statt. Ein Angreifer kann so etwa eine Anfrage zur Auswahl des Moduls `../` abschicken und anschließend die Funktion zum Herunterladen des Moduls aufrufen. Der Doppelpunkt mit Schrägstrich sorgt für das Wechseln in das übergeordnete Verzeichnis, die Abfrage zum Herunterladen von `/home/ros/user_data/mhorst/..` führt zum Herunterladen des Ordners `/home/ros/user_data/`, welcher die Nutzerdaten von allen Benutzern enthält. Im Kontext der angedachten Verwendung von openEASE im Übungsbetrieb von Vorlesungen könnte ein Student die Lösungsdaten aller anderer Studenten einsehen; potentiell auch eine Musterlösung, wenn der Dozent sie in seinem Benutzerkonto hinterlegt hat. Durch fehlende Eingabeparametervalidierung ermöglicht der Editor daher einen Angriff auf die Vertraulichkeit der Nutzerdaten von openEASE.

3.2.4 Datenbanken

In openEASE kommen zwei Datenbanken zum Einsatz: PostgreSQL für die Verwaltung der Nutzerdatenbank und Nutzerrollen und MongoDB als Quelle für große Experimentdaten in KNOW-ROB-Nutzercontainern.

3.2.4.1 PostgreSQL

Die PostgreSQL-Datenbank läuft eigenständig in einem separaten Docker-Container und ist nur aus dem Docker-Container von *Webrob* heraus erreichbar. Analysen der Datenbankkonfiguration ergaben keine Auffälligkeiten und auch die Verwendung der Datenbank in *Webrob* ist durch konsequente Verwendung der Persistenzschicht „SQLAlchemy“ vor SQL-Injektion-Angriffen geschützt. Die Persistenzschicht sorgt für eine Kapselung der Datenbankstruktur in Python-Objekte, generiert selbstständig SQL-Anweisungen aus den Interaktionen mit den Objekten und separiert dabei automatisch die Daten von den Anweisungen, so dass hier keine Schwachstellen gefunden werden konnten.

3.2.4.2 MongoDB

Die MongoDB ist ebenfalls in einem eigenständigen Docker-Container eingerichtet, wird jedoch in allen Nutzercontainern und durch eine Docker-Portfreigabe auch direkt auf dem Host an allen Netzwerkgeräten auf Port 27017 verfügbar gemacht und ist dadurch öffentlich im Internet sichtbar. Das öffentliche Verfügbarmachen ist an sich bereits eher kritisch zu betrachten, da außer den intern in Docker laufenden Nutzercontainern und der Administrator des openEASE-Systems kein direkter Zugang zur Datenbank vorgesehen und notwendig ist. Kombiniert mit der Standardeinstellung von MongoDB, keinerlei Benutzerauthentifizierung durchzuführen und damit vollen Lese- und Schreibzugriff ohne Passwort für jeden zu ermöglichen, ist das öffentliche Betreiben katastrophal. Der „Angriff“ auf die MongoDB-Instanz kann so von jedem Teilnehmer des Internets durchgeführt werden, der den offiziellen MongoDB-Client installiert und `mongo data.open-ease.org` in die Konsole eingeben kann. Anschließend können alle Datenbanken der MongoDB vom Angreifer ausgelesen, modifiziert, vollständig gelöscht oder um neue Datenbanken erweitert werden, was ein sehr hohes Sicherheitsrisiko für die Verfügbarkeit und Vertraulichkeit der Experimentdaten darstellt. Jeder Nutzer von openEASE hat über den eigenen Nutzercontainer ebenfalls eine Verbindung zur MongoDB, darüber ist ein Zugriff im gleichen Ausmaß möglich.

Auch wenn ein Betreiber eines Serversystems im Allgemeinen selber dafür verantwortlich ist, alle öffentlich verfügbaren Dienste so zu sichern, dass kein unautorisierter Zugriff möglich ist oder unsichere Dienste nicht erst aus dem Internet erreichbar sind, muss die Standardeinstellung

der MongoDB-Serversoftware stark kritisiert werden: In der Standardeinstellung ohne Konfiguration öffnet MongoDB auf allen Netzwerkgeräten den Standardport 27017 ohne jegliche Form der Authentifizierung. Die in der Dokumentation stehende Begründung „Exists for future compatibility and clarity.“ für die fehlende Authentifizierung²⁶ zeigt, dass die Entwickler der Datenbank das von Glaser bereits im Jahr 1965 vorgeschlagene Prinzip der sicheren und restriktiven Voreinstellungen (aus [SS75, 1.A.3.b.]) nicht kennen oder absichtlich zu Gunsten von Bequemlichkeit missachtet haben. Jeder Zugriff sollte bei Standardeinstellungen zunächst explizit gestattet werden müssen, anstatt ihn umgekehrt erst durch das Setzen einer Konfigurationseinstellung zu verwehren. Wie sich während der Analyse herausstellte, war nicht nur die MongoDB-Instanz von openEASE durch unsichere Voreinstellungen gefährdet: Studenten der Universität Saarland fanden im Januar 2015 ca. 40.000 ähnlich ungesicherte MongoDB Instanzen im Internet, wobei die tatsächliche Zahl höher (ausgelöst durch Blockaden des Scannens durch Internetanbieter) oder niedriger (durch einen Anteil von Honeypots an dem Ergebnis) sein kann (aus [HGP15, S.2]). Für zukünftige Veröffentlichungen von MongoDB sollte standardmäßig Authentisierung notwendig werden oder der Port von MongoDB zunächst nur lokal auf dem Computer geöffnet werden, auch wenn das Datenbanksystem dann weniger „kompatibel und übersichtlich“ ist.

3.2.5 Nutzercontainer/Knowrob

In openEASE sind die Nutzercontainer der wichtigste Bestandteil des Gesamtsystems. Pro Nutzercontainer läuft zur Verarbeitung der Anfragen an die Wissensbasis eine Instanz von KNOWROB darin. Da die Programmlogik von KNOWROB hauptsächlich in Prolog geschrieben wurde und auch die Anfragen in Form von Prologausdrücken zur Wissensdatenbank geschickt werden sollen, ist ein Prolog-Interpreter zur Nutzung von KNOWROB notwendig.

Der verwendete Interpreter heißt SWI-Prolog²⁷, hauptsächlich entwickelt durch Jan Wielemaker von der Universität von Amsterdam. Besonders interessant an SWI-Prolog ist die Möglichkeit, Java und Prolog bidirektional zu benutzen und die mitgelieferten Bibliotheken zur Verarbeitung von Ontologien wie RDF-S, also Dateien mit Definitionen von Klassen, den Beziehungen zwischen ihnen und Regeln zur Inferenz über sie. Durch die Ausrichtung von KNOWROB auf die Verwendung in Robotiksystemen mit ROS ist KNOWROB vollständig darin integriert. So gibt es etwa das Skript `rosprolog`, das eine Integration von ROS und SWI-Prolog vornimmt, Funktionen zum Laden von Prologmodulen aus ROS-Paketen innerhalb von SWI-Prolog bereitstellt und für Javaklassen in ROS-Paketen einen Klassenpfad erzeugt, der dann an SWI-Prolog zur Nutzung der Klassen innerhalb von Prolog weitergegeben wird. Da SWI-Prolog eine Konsolenanwendung ist, die nicht zur Fernsteuerung ausgelegt ist, bietet KNOWROB die Java-Anwendung

²⁶ <http://docs.mongodb.org/manual/reference/program/mongod/#cmdoption--noauth>, Abruf am 22. Mai 2015

²⁷ <http://www.swi-prolog.org/>

json_prolog, die als ROS-Node eine Instanz von SWI-Prolog startet und analog zur Konsoleneingabe Prolog-Ausdrücke in JSON-Form entgegennimmt und die Ergebnisse JSON-codiert wieder zurückgibt.

Innerhalb eines ROS-Systems gibt es üblicherweise viele ROS-Nodes, die über sog. Topics und Services miteinander kommunizieren. Topics sind Nachrichtenkanäle, auf denen eine beliebige Anzahl von ROS-Nodes Nachrichten in einem vorher spezifizierten Format veröffentlichen können. Andere ROS-Nodes können Topics dann abonnieren, worauf sie alle auf dem Nachrichtenkanal veröffentlichten Nachrichten erhalten. Während Topics unidirektional sind, sind Services bidirektional und können als Remote Procedure Call (RPC)-Mechanismus verstanden werden. ROS-Nodes können Services bereitstellen, die dann von anderen ROS-Nodes aufgerufen werden können, wobei wieder ein vorher spezifiziertes Nachrichtenformat verwendet wird. *json_prolog* etwa stellt die Services `query` zum Absetzen eines Prolog-Ausdrucks, `next_solution` für das Abrufen der nächsten Lösung eines vorher abgesetzten Ausdrucks und `finish` für das Beenden der Verarbeitung eines Prolog-Ausdrucks an. Jegliche Kommunikation in ROS läuft über interne Netzwerkkommunikation ab, was eine vollständige gegenseitige Sichtbarkeit aller Ports von allen ROS-Nodes erfordert. Dies ist für entfernte Zugriffe, etwa über das Internet, nicht hinnehmbar, da Ports im Internet nicht pauschal freigegeben werden sollten (Gefahr durch Angriffe auf andere Betriebssystemkomponenten) und Nutzer häufig gar nicht die Möglichkeit haben, selber Ports freizugeben (durch Firewall oder NAT). Die Lösung des Problems liegt in *rosbridge_server*, eine Anwendung zum Tunneln von ROS-Kommunikation über eine einzige WebSocket-Verbindung und einen einzigen Port. Dieser Port wird in openEASE dann über eine Docker-Portfreigabe von außen zugänglich gemacht, sowie durch die dynamische Umleitung des *nginx*-Webserver (siehe Kapitel 3.2.2 auf Seite 22) unter der URL `/ws/<Nutzername>` veröffentlicht. Wie schon bei *Webrob* (Kapitel 3.2.3.1 auf Seite 24) sind durch die direkte Öffnung des Ports auf dem Host und die Umleitung in *nginx* zwei Verbindungsmöglichkeiten geschaffen worden, von denen eine nicht benötigt wird.

Die Kombination aus *json_prolog* und *rosbridge_server* ermöglicht es den JavaScript-Bibliotheken (speziell *roslibjs*) in der openEASE-Oberfläche, über das Internet eine Verbindung zum ROS-System im Nutzercontainer aufzubauen, Prolog-Anfragen als ROS-Serviceaufruf an *json_prolog* zu senden und ROS-Topics zu abonnieren, die benötigte Daten zur Visualisierung bereitstellen. Auch für diesen Teil von openEASE ergab die Sicherheitsanalyse einige Schwachstellen, die nachfolgend erläutert werden.

3.2.5.1 Datencontainer

Innerhalb von openEASE existieren zwei Datencontainer, die sowohl von *Webrob* als auch von den Nutzercontainern eingebunden und genutzt werden. Der eine Container heißt *knowrob_data* und enthält im Ordner `/home/ros/knowrob_data` die vorbereiteten Experimentdaten, die in

openEASE genutzt werden können. Der andere Container heißt *user_data* und enthält einerseits die Historie von Prolog-Abfragen jedes Nutzers, als auch die im Editor durch die Nutzer erstellten eigenen Prologmodule. Jeder Nutzer hat in dem Container einen eigenen Ordner nach dem Schema `/home/ros/user_data/<Nutzername>`.

Eine Begutachtung der Art und Weise, wie die Datencontainer in *Webrob* und die Nutzercontainer eingebunden werden ergab, dass alle Nutzer vollständige Schreib- und Leserechte für alle Daten beider Container besitzen. Dies ist nicht nur eine Bedrohung für die Verfügbarkeit, wenn ein Nutzer von openEASE etwa versehentlich über Dateien-schreibende Prolog-Ausdrücke die Experimentdaten für alle anderen Nutzer gleichermaßen verändert, sondern auch für die Vertraulichkeit. Kein Sicherheitsmechanismus hindert einen böswilligen Nutzer daran, alle Daten im *user_data*-Container beliebig zu lesen, zu verändern oder zu löschen. Es ist daher dringend notwendig, Experimentdaten nur lesend in Nutzercontainer einzubinden und durch geeignete Trennungsvorgänge den Zugriff auf fremde Nutzerdaten zu verhindern.

3.2.5.2 Ressourcen

Wird ein Nutzercontainer gestartet, so weist er keinerlei Einschränkungen hinsichtlich der ihm zur Verfügung stehenden Hardwareressourcen auf. Theoretisch kann so ein einziger Nutzercontainer durch Ausführung besonders rechen- und speicherintensiver Prozesse alle Ressourcen des Hosts beanspruchen und so die Verfügbarkeit von openEASE einschränken. Begünstigt wird das Ressourcenproblem durch die Eigenschaft der Java-Anwendung *json_prolog*, nicht mehr verwendeten Speicher erst dann freizugeben, wenn die Menge an insgesamt beanspruchten Speicher gegen ein Gigabyte tendiert. Laufen dann noch Nutzercontainer nach Beendigung der Nutzeraktivität auf openEASE weiter (siehe Kapitel 3.2.3.3 auf Seite 26) oder werden massenhaft in Form eines DoS-Angriffes gestartet, ist es eine Frage der Zeit, bis alle Ressourcen des Hosts aufgebraucht sind und openEASE nicht mehr erreichbar ist.

Nutzercontainer müssen folglich für einen stabilen Betrieb von openEASE unbedingt Beschränkungen für insgesamt nutzbare Hardwareressourcen erhalten. Auch die insgesamte Menge an Nutzercontainern muss auf einen Wert begrenzt werden, bei dem der für den Betrieb von openEASE verwendete Host noch ein Mindestmaß an Ressourcen übrig hat (z.B. so viel, dass bei maximaler Anzahl gleichzeitig laufender Container noch ein Gigabyte Arbeitsspeicher frei bleibt).

3.2.5.3 Authentifizierung

Dass im ROS-System Belange der Informationssicherheit kaum berücksichtigt werden, geht aus dem Einsatzgebiet bereits hervor. Der Fokus liegt eher auf verlässlicher Kommunikation und bestmöglicher Unterstützung bei der Integration von stark inhomogenen Softwaresystemen zur

Robotersteuerung. Es wird außerdem eher lokal bzw. in kleinen, vertrauenswürdigen und geschlossenen Netzwerken betrieben. Das in den Nutzercontainern eingesetzte *rosbridge_server* zum Transport von ROS-Kommunikation über das Internet ist standardmäßig ebenfalls ungeschützt: Die Kommunikation über die *WebSocket*-Verbindung ist weder verschlüsselt, noch muss sich ein Nutzer zum Verbindungsaufbau authentisieren. Zum Zeitpunkt der Sicherheitsanalyse war es möglich, sich mit beliebigen Nutzercontainern zu verbinden und so fremde Container ohne Eingabe des Passwortes für openEASE zu steuern.

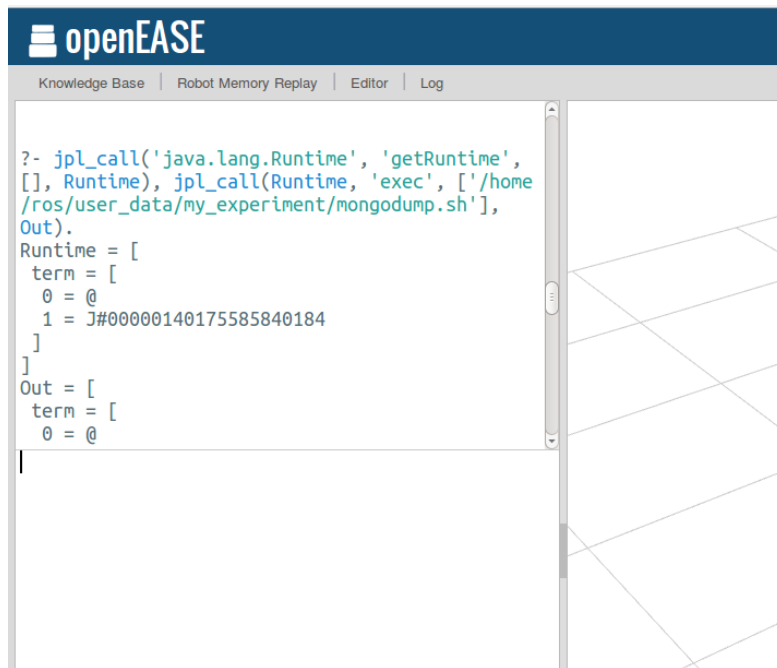
Zur Beschränkung des Zugriffes auf Nutzercontainer muss ein integritätssicherndes Verfahren eingeführt werden, das eine Authentifizierung des Nutzers zulässt. Die Vertraulichkeit kann über den *nginx*-Webserver sichergestellt werden, wenn dieser zur Verwendung von Transportverschlüsselung konfiguriert wird.

3.2.5.4 Prolog-Interpreter

Als letzter und wichtigster Teil des Nutzercontainers bleibt noch die Untersuchung des Prolog-Interpreters. An ihn werden die Prolog-Ausdrücke, die der Nutzer auf der openEASE-Oberfläche zu den Experimenten abschickt, übermittelt und anschließend ausgeführt.

Prolog ist eine universelle Programmiersprache, die durch entsprechende Module im Interpreter Vollzugriff auf Dateisystem, Netzwerk und Prozesse gewährt. Die SWI-Prolog spezifische Erweiterung *JPL* erlaubt zudem den Zugriff auf Javaklassen und die Ausführung von Methoden daraus innerhalb von Prolog. Diese Möglichkeiten sorgen dafür, dass ein Nutzer von openEASE praktisch beliebige Programme im Kontext des Nutzercontainers ausführen kann. Eine Demonstration der Möglichkeiten zeigt Abbildung 3.2, in der mithilfe von *JPL* und der Funktion *exec* aus dem Standard-Javapaket *java.lang.Runtime* ein eigenes Kommandozeilenskript im Nutzercontainer gestartet wird. In Kombination mit anderen Sicherheitsmängeln etwa bei der MongoDB (3.2.4.2 auf Seite 28) ist diese Schwachstelle sehr kritisch, da sie das böswillige Ausnutzen der Sicherheitsmängel deutlich erleichtert. Ein Angreifer wird nicht erst die Syntax der Prologfunktionen zum Löschen aller Nutzerdaten lernen, wenn er stattdessen ein eigenes Skript dazu schreiben kann, welches dann in einer Kommandozeilenanwendung wie *bash* ausgeführt wird. Er wäre auch in der Lage, im Nutzercontainer nicht vorhandene Anwendungen herunterzuladen und auszuführen. Wird jedoch angenommen, dass die genannten Sicherheitsmängel nicht vorhanden wären, so würde sich der potentielle Schaden durch Nutzung von Funktionen des Prolog-Interpreters für die Ausführung von unerwünschten Fremdprogrammen in den meisten Fällen nur auf den eigentlichen Nutzercontainer beschränken.

Es ist daher zu prüfen, wie Prolog-Ausdrücke mit unerwünschten Funktionsaufrufen gefiltert oder anderweitig an der Ausführung im Interpreter gehindert werden können; alternativ müssten alle anderen mit dieser Schwachstelle kombinierbaren Sicherheitsmängel abgestellt werden.



```
?- jpl_call('java.lang.Runtime', 'getRuntime',
[], Runtime), jpl_call(Runtime, 'exec', ['/home
/ros/user_data/my_experiment/mongodump.sh'],
Out).
Runtime = [
  term = [
    0 = @
    1 = J#00000140175585840184
  ]
]
Out = [
  term = [
    0 = @
  ]
]
```

Abbildung 3.2 Ausführung eines eigenen Kommandozeilenskriptes über die openEASE-Oberfläche

3.3 Komponentendiagramm von openEASE

Um eine Übersicht über die Architektur von openEASE zu erhalten, wurde ein Komponentendiagramm erstellt (siehe Abbildung 3.3 auf Seite 34). Es umfasst alle Docker-Container und die Berechtigungsverhältnisse zueinander. Das Ziel des Komponentendiagrammes ist, die Wege eines Angriffs nachvollziehbar zu machen und anzuzeigen, ob verwundbare Komponenten durch eine Berechtigung zum Zugriff auf andere Komponenten diese ebenfalls gefährden.

Das Diagramm zeigt außen das Hostsystem, in das eingehende Netzwerkverbindungen (als blaue Pfeile gekennzeichnet) führen. Innerhalb des Hosts sind Ressourcen, auf die von Docker aus zugegriffen werden, abgebildet, sowie die Docker-Containerebene selbst. Allen Anwendungs- und Datencontainern von openEASE sind darin aufgeführt, links die Anwendungs- und rechts die Datencontainer. Zwischen den Containern sind Pfeile eingezeichnet: Blaue Pfeile bedeuten den Zugriff auf das Netzwerk des Containers, auf den der Pfeil zeigt, rote Pfeile bedeuten den Zugriff auf bestimmte Teile des Dateisystems des Containers, auf den der Pfeil zeigt. An den Pfeilen stehen zusätzlich relevante Informationen, ob etwa der Dateisystemzugriff nur lesend oder auch schreibend möglich ist, oder welcher Netzwerkport freigegeben ist.

Die Farbe der Container bzw. Host-Ressourcen gibt an, ob im Rahmen der Sicherheitsanalyse ein Mangel festgestellt wurde und wie bedrohlich er für den Betrieb von openEASE ist: Weiß ist unbedenklich, gelb bedeutet eine Gefährdung einzelner Nutzer bzw. eine Gefährdung der Verfügbarkeit mit geringem Schadenspotential und rot bedeutet eine starke Gefährdung für den Gesamtbetrieb mit hohem Schadenspotential.

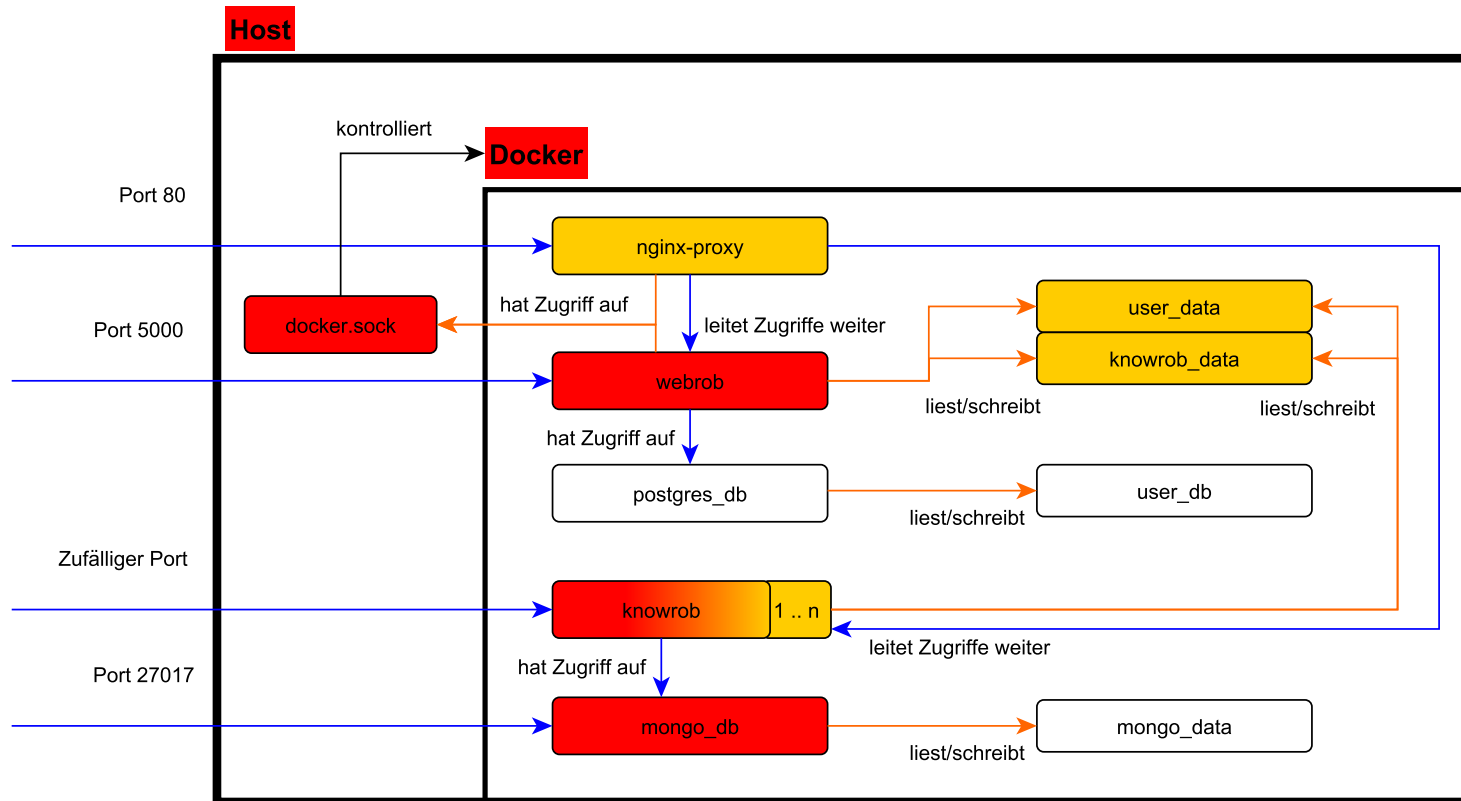


Abbildung 3.3 Komponentendiagramm von openEASE

3.4 Zusammenfassung und Bewertung der Sicherheit

Die Sicherheitsanalyse von openEASE hat in fast allen Komponenten Schwachstellen aufgezeigt. Die Schwere der Schwachstellen reichte dabei von potentiellen Bedrohungen ohne konkret benannte Möglichkeiten der Penetration, bis hin zur Erlangung totaler Kontrolle über das Hostsystem. Sie zeigt die Wichtigkeit, bereits vor und während des Entwickelns von Anwendungen die Sicherheit zu berücksichtigen. Hervorzuheben ist, dass die Entwickler von openEASE sich allgemein der mangelhaften Sicherheit bewusst waren und schon im Vorfeld der Arbeit etwa die unverschlüsselte Kommunikation und die unauthentifizierte `WebSocket`-Verbindung als verbesserungswürdig dargestellt haben. Vermutlich führten im Fall von openEASE knappe Zeitvorgaben und eine hohe Entwicklungsgeschwindigkeit zu einer Situation, in der durch unvollständige Auseinandersetzung mit den eingesetzten Technologien und Anwendungen zum Zeitpunkt der Entwicklung eine Berücksichtigung von Sicherheit nicht möglich war. Zudem ist es aus eigener Erfahrung häufig schwierig, selbst verfasste Anwendungen, Komponenten oder Quelltexte distanziert und unvoreingenommen zu betrachten. Letzteres Problem ist mit einer Überprüfung der Anwendung durch einen unbeteiligten Dritten einfach zu lösen, was durch die Vergabe dieses Bachelorarbeitsthemas auch geschehen ist. Ein Vorwurf der fahrlässigen Vernachlässigung der Anwendungssicherheit wäre damit entkräftet.

Alle gefundenen Schwachstellen müssen mit größter Sorgfalt untersucht und durch Erarbeitung von Lösungen behoben werden. Aufgrund der besonderen Schwere einiger Schwachstellen ist sonst der Erfolg und die Akzeptanz von openEASE bedroht, insbesondere wenn mit steigender Bekanntheit die Nutzung der Anwendung zunehmen wird. Zur besseren Übersicht sind in Tabelle 3.1 auf Seite 36 nochmals alle ermittelten Sicherheitsmängel von openEASE aufgelistet, inklusive Angabe des Kapitelabschnitts, in dem der jeweilige Mangel detailliert beschrieben ist. Es sei jedoch nachdrücklich darauf hingewiesen, dass die Liste keineswegs Anspruch auf Vollständigkeit hat.

Genau wie ein Entwickler eine voreingenommene Sicht auf die eigenen Anwendungen hat, ist ein Prüfer bei der Untersuchung voreingenommen auf Basis der eigenen Erfahrung mit Informationssicherheit. So ist es wahrscheinlich, dass auch in dieser Arbeit Sicherheitsmängel nicht gefunden wurden, etwa weil der Autor sie nicht in Betracht gezogen hat, oder sie für den Autor unbedeutend erschienen. Es ist daher sehr zu empfehlen, nach Abschluss der Arbeit die Sicherheit von openEASE durch weitere Personen prüfen zu lassen, um sich einer vollständigen Sicherheitsanalyse annähern zu können.

Nr.	Kurzbeschreibung	Kapitel
1.	Installierte, nicht benötigte Anwendungen auf dem openEASE-Host	3.1.1
2.	Unregelmäßiges Einspielen von Software- und Betriebssystemaktualisierungen	3.1.2
3.	Fehlen von automatischen Sicherungen der Nutzerdaten	3.1.4
4.	Keine Transportverschlüsselung in nginx konfiguriert	3.2.2
5.	Zugang zur Docker-API bei Kompromittierung von nginx	3.2.2
6.	Nutzung des internen Entwicklungsservers von Flask in Webrob	3.2.3.1
7.	Veröffentlichung des geheimen Schlüssels für Sitzungsdaten in Webrob	3.2.3.2
8.	Aktiviert Debug-Konsole in Webrob	3.2.3.2
9.	Zugang zur Docker-API in Webrob	3.2.3.3
10.	Unbestimmt lange laufende Nutzercontainer durch Nichterkennung von Abmeldevorgängen in Webrob	3.2.3.3
11.	Zugriff auf Daten anderer Nutzer im Editor von Webrob	3.2.3.4
12.	Öffentlicher Zugriff auf den MongoDB-Server ohne Authentisierung	3.2.4.2
13.	Zugriff auf Daten anderer Nutzer im Nutzercontainer	3.2.5.1
14.	Unbeschränkte Hardwareressourcen für Nutzercontainer	3.2.5.2
15.	Unauthentisierter Zugriff auf WebSocket-Verbindungen der Nutzercontainer	3.2.5.3
16.	Ausführung unerwünschter Prolog- und Javafunktionen im Prolog-Interpreter	3.2.5.4

Tabelle 3.1 Liste der gefundenen Sicherheitsmängel in openEASE

Konzept zur Absicherung von openEASE

Dieses Kapitel beschreibt die erarbeiteten Lösungsvorschläge für die im vorherigen Kapitel ermittelten Sicherheitsmängel von openEASE. Zur Identifizierung der Sicherheitsmängel werden sie mit den Nummern referenziert, die in Tabelle 3.1 auf Seite 36 definiert wurden.

4.1 Betriebssystem des Hosts

Das Betriebssystem, welches auf dem Host von openEASE betrieben wird, ist durch die Menge an nicht benötigten, vorinstallierten Anwendungen ungeeignet (Sicherheitsmangel Nr. 1). Ein Betriebssystem für den dauerhaften Serverbetrieb sollte nur eine minimale Menge an Anwendungen mitliefern (etwa zur Fernsteuerung, Softwarepaketverwaltung und Überwachung), keine nicht der Servertätigkeit dienenden Anwendungen und Treiber enthalten (z.B. graphische Benutzeroberflächen oder Multimedia) und so langfristig und vor allem so schnell wie möglich mit Sicherheitsaktualisierungen versorgt werden. Ein derartiges Betriebssystem ist z.B. Ubuntu Server 14.04 LTS. Es enthält aktuelle und getestete Softwarepakete und wird bis April 2019 mit Sicherheitsaktualisierungen versorgt¹. Es wird empfohlen, openEASE auf einen Host mit Ubuntu Server 14.04 LTS zu migrieren. Wie in Kapitel 3.1.3 auf Seite 15 erwähnt, ist bereits eine Migration von openEASE auf ein anderes Computersystem geplant, so dass im Zuge dessen auch gleich ein Wechsel zu einem Serverbetriebssystem vorgenommen werden kann. Auf eine Umsetzung in dieser Arbeit wurde angesichts der Migrationspläne verzichtet.

Ein weiterer, das Betriebssystem betreffender Mangel (Nr. 2) ist die unregelmäßige Aktualisierung der genutzten Anwendungen auf dem Host, sowie des Betriebssystems. Gängige GNU/Linux-basierte Betriebssysteme bieten hier Aktualisierungsroutinen, meist in Form von Paketverwaltungssoftware wie *dpkg* oder *rpm*, die zum Aktualisieren von installierten Anwendungen und dem

¹<http://www.ubuntu.com/download/server>, Abruf am 25. Mai 2015

Betriebssystem eingesetzt werden können. Zunächst wurde als Lösungsvorschlag erwogen, Aktualisierungen regelmäßig automatisch einspielen zu lassen, was etwa unter Ubuntu mit der Anwendung *unattended-upgrades* möglich wäre. Dies hätte den Vorteil, dass menschliche Interaktion in den wenigsten Fällen notwendig wird und dass sicherheitskritische Updates zeitnah eingespielt werden. Allerdings bergen Aktualisierungen auch immer die Gefahr, dass sie eine vormals funktionierende Integration von verschiedenen Anwendungen durch unvorhersehbare Wechselwirkungen stören. Eigentlich sollte dies bei Sicherheitsaktualisierungen nicht vorkommen, die Möglichkeit besteht jedoch, so dass sich letztendlich gegen automatische Aktualisierungen entschieden wurde, um Systemadministratoren vorher zu ermöglichen, die Funktionsfähigkeit zu prüfen. Stattdessen wird empfohlen, regelmäßig mindestens einmal in der Woche die Paketverwaltung auf Aktualisierungen zu prüfen und ggf. nach Tests auch einzuspielen. Weiterhin sollten die Systemadministratoren eine Mailingliste für Sicherheitsaktualisierungen abonnieren, um auf kritische Sicherheitslücken in verwendeten Anwendungen zeitnah reagieren zu können. Für Ubuntu wäre dies *ubuntu-security-announce*: <https://lists.ubuntu.com/mailman/listinfo/ubuntu-security-announce>.

4.2 Sicherungsstrategie

In Nr. 3 der Mängelliste wurde festgestellt, dass Datensicherungen entweder unregelmäßig und ohne Dokumentation (Experimentdaten) oder überhaupt nicht (Nutzerdaten und Nutzerdatenbank) erfolgen, was die Verfügbarkeit gefährdet. Bei näherer Betrachtung stellt sich die Aufgabe der Sicherung als nicht-trivial heraus: Die Nutzerdaten sind komplett in Docker-Datencontainern gespeichert. Physikalisch liegen diese in der Standardeinstellungen in `/var/lib/docker`, wobei sie dort nicht mit ihrem Namen gespeichert sind, sondern mit der internen Identifikationsnummer von Docker. Das Verzeichnis enthält außerdem viele irrelevante Metadaten und die Daten aus den openEASE-Anwendungscontainern, sowie alle weiteren eventuell auf dem Computer existenten Docker-Container. Zudem ist das Verzeichnis nur von dem Systembenutzer `root` lesbar, benötigt also Administratorenrechte.

Daher wurden nachfolgend beschriebene Maßnahmen für Datensicherungen ergriffen. Zunächst wurden die zu sichernden Dateien und Ordner auf dem Hostsystem, sowie die zu sichernden Docker-Container identifiziert und ausführlich beschrieben - das daraus entstandene Konzept wurde in die Versionsverwaltung von openEASE aufgenommen und ist im Projektordner von openEASE einsehbar (`openease_storage_concept.txt`²). Des Weiteren wurden die Skripte `backup` und `restore` im Ordner `scripts` des openEASE-Projekts erstellt, die betroffene Docker-Container in ein `tar`-Archiv sichern bzw. aus selbigem wiederherstellen können.

Es wird empfohlen, regelmäßig jede Woche eine Sicherung mithilfe des Skriptes automatisch

²https://github.com/knowrob/docker/blob/master/openease_storage_concept.txt

durchführen zu lassen. Die daraus resultierenden `tar`-Archive sollten ebenfalls automatisch auf einen anderen Rechner als dem, auf dem openEASE betrieben wird, kopiert werden. Im Falle eines Datenverlustes kann so die Verfügbarkeit wiederhergestellt werden.

4.3 Nginx-Webserver

Der *nginx*-Webserver in openEASE wird dazu genutzt, Anfragen von Nutzern an die Webanwendung *Webrob* und die Instanzen der *knowrob*-Nutzercontainer weiterzuleiten. Da so sämtlicher Datenverkehr von Nutzern zu openEASE durch *nginx* geleitet wird, eignet er sich ideal zur Sicherung mit der Transportverschlüsselung TLS und damit zur Schaffung von Vertraulichkeit (durch Verschlüsselung) und Integrität (durch den Einsatz von HMAC und die Möglichkeit der Identitätsverifikation durch das Server-Zertifikat). Daher wurde die Konfiguration von *nginx* um Einträge für die Unterstützung von TLS auf Port 443 erweitert. Besonderer Wert wurde darauf gelegt, dass unsichere Kryptographieverfahren wie SSL2, SSL3, RC4, MD5 oder US-Exportalgorithmen nicht angeboten werden. Das notwendige SSL-Zertifikat wurde dankenswerterweise durch den Systemadministrator der AGKI, Patrick Mania, bereitgestellt. Durch TLS konnte der Sicherheitsmangel Nr. 6 ausgeräumt werden.

Der andere, *nginx* betreffende Mangel (Nr. 5) umfasste das Bereitstellen der Docker-API im Docker-Container von *nginx*. Obwohl der Mangel nur eine theoretische Bedrohung darstellt, wäre ein erfolgreicher Angriff durch das Bekanntwerden einer Sicherheitslücke in *nginx* für die Sicherheit des Hosts verheerend. Daher wurden wie in der Dokumentation des Programms *docker-gen* beschrieben, zwei separate Docker-Container erstellt, einer auf Basis des offiziellen *nginx* Docker-Images und einer auf Basis des *docker-gen* Images. In dieser Konstellation kann *docker-gen* zwar die Konfiguration im *nginx*-Container beim Starten und Beenden von *knowrob*-Nutzercontainern beschreiben und *nginx* zum Neu-Einlesen der Konfiguration anweisen, hat aber selber keinen Zugang von außen durch eine Portfreigabe. Dadurch kann die Docker-API problemlos nur im *docker-gen*-Container bereitgestellt werden. Kontrolliert nun ein Angreifer durch eine Sicherheitslücke in *nginx* selbigen Container, so kann er nicht auf die Docker-API zugreifen, da sie nur im *docker-gen*-Container eingebunden ist.

Zusätzlich konnte durch Nutzung der offiziellen *nginx* und *docker-gen* Docker-Images statt der veralteten eigenen Version erreicht werden, dass immer aktuelle Versionen von *nginx* und *docker-gen* verwendet werden.

4.4 Dockerbridge

Dockerbridge ist eine im Rahmen dieser Arbeit entstandene Python-Anwendung zur Kapselung der Docker-API. Motiviert durch den frühzeitigen Fund der Sicherheitsmängel 8. und 9. konnte schnell erkannt werden, dass der Zugriff auf die Docker-API gefährlich und daher von den Bereichen, in die ein Angreifer potentiell vordringen kann, vollständig isoliert werden muss. So wurde mit dem Entschluss, die *Dockerbridge* zu entwickeln, sämtlicher für die Steuerung von Docker-Containern notwendiger Programmcode aus *Webrob* in *Dockerbridge* verschoben und eine davon abstrahierende Ebene programmiert. Über das Kommunikationsprotokoll JSONRPC wird die abstrakte Ebene intern im Docker-Netzwerk zur Verfügung gestellt, wozu in *Dockerbridge* als einzige externe Schnittstelle intern ein HTTP-Server auf Port 5001 gestartet wird. Über den HTTP-Server werden alle speziell mit dem Annotator `@pyjsonrpc.rpcmethod` gekennzeichneten Funktionen anderen Programmen mit JSONRPC-Client zum Aufruf angeboten. Ein Client kann dann durch Aufbau einer Verbindung und Angabe von Funktion und deren Parameter über das Netzwerk diese Funktion in *Dockerbridge* aufrufen.

Höchstes Ziel bei der Gestaltung der über JSONRPC verfügbaren Funktionen war die Validierung der Parameter, die ein Client an *Dockerbridge* übermittelt. So gibt es beispielsweise eine Funktion `files_exists`, die einen Dateinamen als Parameter entgegennimmt und prüft, ob sich eine Datei am angegebenen Pfad innerhalb eines Nutzercontainers befindet. *Dockerbridge* prüft hier, ob durch nicht-maskierte Zeichen, Kommandozeilensteuerzeichen oder sich durch Beziehung auf außerhalb des Funktionskontext liegende Dateien versucht wird, die Abstraktion zu umgehen. Wird eine Verletzung der zulässigen Eingabewerte festgestellt, bricht die Ausführung der Funktion sofort ab, noch bevor die zur Existenzprüfung notwendigen Befehle an die Docker-API abgesetzt wurden. Mit dieser Kapselung der Docker-API konnte *Webrob* der direkte Zugang zur API entzogen werden und durch Aufrufe der von *Dockerbridge* angebotenen JSONRPC-Schnittstellen ersetzt werden. Ein Angreifer, der etwa durch Sicherheitsmangel Nr. 8 oder eine andere, noch unbekannt Methode Vollzugriff auf *Webrob* erhält, kann nun nicht mehr durch Nutzung der Docker-API Administrationszugang zum Hostsystem erhalten, da die benötigten Funktionen der API von *Dockerbridge* nicht exponiert werden.

Nachdem die erste Fassung der *Dockerbridge* bereits Mitte Februar und damit recht weit am Anfang dieser Arbeit in openEASE integriert wurde, bot es sich an, die *Dockerbridge* um weitere Funktionen zur Docker-Containerverwaltung zu erweitern, die der Behebung weiterer Sicherheitsmängel dienen würden.

Die Nichterkennung von Abmeldevorgängen in openEASE (Mangel Nr. 10) etwa wurde durch eine ständig laufende Kontrollfunktion in *Dockerbridge* behoben, in der zu jedem gerade in Ausführung befindlichen Nutzercontainer festgehalten wird, wie lange bereits keine Aktivität des Nutzers vorgelegen hat. Überschreitet die Zeit der Inaktivität für einen Nutzer zehn Minuten, so wird der Nutzercontainer durch die Kontrollinstanz automatisch beendet. Jede Aktivität des

Nutzers führt zum Zurücksetzen seiner Inaktivitätsmessung. Der Begriff der Aktivität ist hier absichtlich sehr weit gefasst - das alleinige Offenhalten der openEASE-Seite im Browser zählt hier schon als Aktivität. Dadurch soll verhindert werden, dass in langen Arbeitssitzungen von Nutzern, in denen über längere Zeit die Oberfläche nur betrachtet wird, nicht ungewollt der Nutzercontainer terminiert wird. Die Aktivitätsmessung findet daher im Browser des Nutzers durch eine periodisch ausgeführte JavaScript-Funktion statt, die eine Funktion in *Webrob* aufruft, in der wiederum über einen JSONRPC-Aufruf die *Dockerbridge* über die Aktivität informiert wird. Schließt nun ein Nutzer die openEASE-Seite, ohne sich vorher abzumelden, wird die JavaScript-Funktion nicht mehr ausgeführt und der Nutzercontainer wird beendet.

Weitere Funktionen in *Dockerbridge* wurden zum Zugriff auf Nutzer-Datencontainer implementiert, welche in Abschnitt 4.6 auf Seite 44 beschrieben sind, sowie notwendige Hilfsmethoden zur Realisierung der Authentifizierung der WebSocket-Verbindung zu Nutzercontainern (ebenfalls Abschnitt 4.6 auf Seite 44).

Zur Übersicht und Beurteilung der Abstraktion ist nachfolgend eine Auflistung der Funktionen, die über JSONRPC bereitstehen und vom Autor dieser Arbeit implementiert wurden³, gegeben:

- `create_user_data_container(container_name)`: Erzeugt einen neuen Nutzerdatencontainer für den mit `container_name` benannten *knowrob*-Nutzercontainer.
- `start_user_container(application_image, user_container_name)`: Erzeugt und startet einen *knowrob*-Nutzercontainer. `user_container_name` gibt dabei den Namen des Nutzercontainers an, `application_image` den Namen des Docker-Images, auf welchem der Nutzercontainer basieren soll.
- `stop_container(user_container_name)`: Stoppt und entfernt den mit `user_container_name` benannten *knowrob*-Nutzercontainer.
- `container_started(user_container_name, base_image_name)`: Prüft, ob ein Docker-Container mit dem Namen `user_container_name` und optional auch dem Basis-Image `base_image_name` ausgeführt wird.
- `get_container_ip(user_container_name)`: Liefert die interne, im Docker-Netzwerk dem mit `user_container_name` benannten Container zugewiesene IP-Adresse.
- `refresh(user_container_name)`: Setzt die angesprochene Rücksetzung des Inaktivitätszählers für den Nutzercontainer `user_container_name` um.
- `get_container_log(user_container_name)`: Gibt das Protokoll für einen Nutzercontainer zurück.
- `files_fromcontainer(user_container_name, sourcefile)`: Liefert als Rückgabewert den Inhalt der Datei `sourcefile` aus dem Datencontainer für den Nutzercontainer `user_container_name` zurück.
- `files_tocontainer(user_container_name, data, targetfile)`: Überschreibt die Datei

³*Dockerbridge* wurde nach Einführung auch zur Realisierung anderer Funktionen durch einen Entwickler von openEASE genutzt, diese sind hier aufgrund des fehlenden Bezuges nicht mit aufgeführt, wurden jedoch mit Unterstützung des Autors unter Berücksichtigung der Validierungsvorgabe implementiert

`targetfile` im Datencontainer für den Nutzercontainer `user_container_name` mit dem Wert aus `data`.

- `files_lft_set_writeable()`: Sorgt für das Setzen der Schreibberechtigung für das Kopieren großer Dateien von und zu Datencontainern.
- `files_largefromcontainer(user_container_name, sourcefile, targetfile)`: Kopiert die Datei `sourcefile` aus dem Datencontainer Datencontainer für den Nutzercontainer `user_container_name` in den Datencontainer für große Dateitransfers mit dem Namen `targetfile`.
- `files_largetocontainer(user_container_name, sourcefile, targetfile)`: Kopiert die Datei `sourcefile` aus dem Datencontainer für große Dateitransfers in den Datencontainer für den Nutzercontainer `user_container_name` mit dem Namen `targetfile`.
- `files_readsecret(user_container_name)`: Liest den geheimen Schlüssel für die Authentisierung am Nutzercontainer `user_container_name` aus.
- `files_writesecret(user_container_name, secret)`: Schreibt den geheimen Schlüssel `secret` in den Datencontainer für den Nutzercontainer `user_container_name`.
- `files_exists(user_container_name, file)`: Prüft, ob die Datei `file` im Datencontainer für den Nutzercontainer `user_container_name` existiert.
- `files_mkdir(user_container_name, dir)`: Erstellt ein neues Verzeichnis `dir` im Datencontainer für den Nutzercontainer `user_container_name`.
- `files_rm(user_container_name, file, recursive)`: Entfernt die Datei `file` aus dem Datencontainer für den Nutzercontainer `user_container_name`. Die Datei wird rekursiv gelöscht (notwendig, wenn die Datei ein Ordner ist), wenn der Parameter `recursive` auf `True` gesetzt wird.
- `files_ls(user_container_name, dir, recursive)`: Gibt eine Auflistung der Inhalte des Ordners `dir` aus dem Datencontainer für den Nutzercontainer `user_container_name` zurück. Wird der Parameter `recursive` auf `True` gesetzt, werden auch die Inhalte aus Unterordnern des angegebenen Ordners zurückgegeben.

4.5 Webrob

Die Auslieferung der Bedienoberfläche von openEASE, die Benutzerverwaltung und die Verwaltung von KNOWROB-Nutzercontainern und den Daten darin obliegt der Komponente *Webrob*.

Sie macht dafür Gebrauch von dem Web-framework Flask, das in Python geschrieben ist und sich selbst das Attribut „Mikroframework“ verleiht, also leichtgewichtig im Betrieb und einfach zu entwickeln sei. Module für Flask erweitern es um oft in Webanwendungen benötigte Komponenten; die Benutzerverwaltung von openEASE wird z.B. von dem Modul `flask-user` übernommen. Flask ist kompatibel zum Python Web Server Gateway Interface (WSGI), zur Ausführung und

Bereitstellung muss eine Flask-Anwendung also von einem WSGI-kompatiblen Webserver ausgeliefert werden.

Hier ist der erste Mangel in *Webrob* (Nr. 6) zu finden. Zur Bereitstellung der *Webrob*-Webanwendung wird in openEASE der bei Flask mitgelieferte, für Tests gedachte Entwicklungsserver benutzt, was in Stabilitäts- und Geschwindigkeitsprobleme resultieren kann. Die Behebung dieses Mangels stellte sich jedoch als sehr einfach heraus: Anstelle des internen Servers wurde der Tornado-Server⁴ für die Seitenauslieferung gewählt, so wie in der Dokumentation von Flask beschrieben⁵. Zusätzlich wurde eine Möglichkeit integriert, den internen Testserver wieder zu aktivieren, wenn für Entwicklungszwecke eine bestimmte Umgebungsvariable (`EASE_DEBUG`) aktiviert ist.

Weitere, Flask betreffende Sicherheitsmängel sind Nr. 7 und Nr. 8. Flask speichert Sitzungsdaten der Nutzer nicht auf dem Server, sondern übermittelt sie als Cookie an die Nutzer selbst, so dass sie ihre eigenen Sitzungsdaten vorhalten. Um die Sitzungscookies vor Manipulation durch die Nutzer zu schützen, werden sie mit einem kryptographischen Verfahren signiert. Der Schlüssel dazu steht bei openEASE im Klartext in der `settings.py`-Datei im Ordner `webrob/webrob/config` des openEASE-Projektordners und wurde auch nicht auf dem produktiv auf <http://data.open-ease.org> laufenden openEASE-System geändert. Dadurch ist es möglich, die Sitzungscookies beliebig zu verändern, da die kryptographische Signatur mit Kenntnis des Schlüssels problemlos gefälscht werden kann (Mangel Nr. 7). Zur Lösung dieses Problems wurde im Rahmen dieser Arbeit *Webrob* so umgeschrieben, dass der Schlüssel aus einem neuen Docker-Datencontainer (`ease_secret`) bezogen wird und nicht mehr aus der öffentlich einsehbaren `settings.py`-Datei ausgelesen wird. Der Datencontainer wird, sofern noch nicht vorhanden, beim Start des openEASE-Systems mittels des `start-webrob`-Skriptes neu erstellt und ausgeführt, was im Container die von *Webrob* ausgelesene Schlüsseldatei erstellt. Bei Erstellung der Schlüsseldatei werden 64 zufällige Zeichen geschrieben, die somit auf jedem System, welches openEASE betreibt, unterschiedlich sind. Zu Fehlerbehebungszwecken ist es aber auch hier möglich, mit der `EASE_DEBUG`-Umgebungsvariable das alte Verhalten herbeizuführen.

Mangel Nr. 8 betraf die Einstellung, bei in Python auftretenden Fehlern im Browser eine Konsole zu öffnen, die eine Fehlerbehebung direkt im Browser ermöglicht. Dabei wurden Pythonanweisungen, die im Browser eingegeben wurden, im Kontext des WSGI-Servers und damit der gesamten Webanwendung ausgeführt, was zunächst die vollständige Kontrolle des *Webrob*-Containers zur Folge hätte. Ohne die *Dockerbridge* hätte auch das gesamte Hostsystem durch missbräuchliche Docker-API Benutzung kontrolliert werden können. Die einfachste Lösung, die auch für openEASE umgesetzt wurde, ist die standardmäßige Deaktivierung der Debug-Konsole. Wie in den beiden vorangegangenen Mängelbeseitigungen lässt sich die Debug-Konsole durch Setzen der `EASE_DEBUG`-Umgebungsvariable wieder aktivieren.

Die Möglichkeit, im Editor von *Webrob* durch Verwendung spezieller Dateinamen auf Daten an-

⁴<http://www.tornadoweb.org/>

⁵<http://flask.pocoo.org/docs/0.10/deploying/wsgi-standalone/#tornado>

derer Nutzer zuzugreifen (Sicherheitsmangel Nr. 11), ist durch die Einführung einer vollständigen Datentrennung zwischen Nutzern mithilfe von Nutzer-Datencontainer bereits nicht mehr existent (siehe das nachfolgende Kapitel 4.6 auf Seite 44). Allenfalls ist noch erwähnenswert, dass die Umstellung des Editors auf die Nutzung der Nutzer-Datencontainer ebenfalls im Rahmen dieser Arbeit vollzogen wurde.

Aus der Beschreibung von *Dockerbridge* in diesem Sicherheitskonzept geht bereits hervor, dass durch eine JavaScript-Funktion in *Webrob* die Aktivitätsfeststellung von Nutzern implementiert wurde (behebt Sicherheitsmangel Nr. 10). Hervorzuheben ist jedoch, dass diese durch eine im Rahmen dieser Arbeit entworfene und implementierte API realisiert wurde, die zum vereinfachten Aufrufen einiger Funktionen in *Webrob* aus JavaScript heraus erstellt wurde. Insbesondere die Funktion zur Authentisierung ist hervorzuheben: Für die Authentisierung mit *rosauth*, welche im folgenden Kapitel beschrieben ist, wird über die API eine Methode bereitgestellt, die anhand des derzeitigen Anmeldestatus des Nutzers prüft, ob dieser zum Zugriff berechtigt ist. Wenn dies der Fall ist, wird aus Nutzernamen, Zeit, Gültigkeitsdauer, einem Zufallswert und einem beim Start des KNOWROB-Nutzercontainers neu erzeugten geheimen Schlüssels ein mit *rosauth* prüfbarer Hashwert erzeugt, der eine Autorisierung des Nutzers am Container ermöglicht.

Die API enthält zudem Funktionen, welche einen automatischen Zugriff auf openEASE ermöglichen. Um von externen autonomen Robotersystemen auf die Wissensbasis von openEASE zugreifen zu können, ist es möglich die zur Autorisierung am Container notwendigen Werte unter Angabe eines API-Schlüssels zu beziehen. Ein derartiger Schlüssel besteht aus einer 64 Zeichen langen Buchstaben- und Zahlenkombination, die von menschlichen Nutzern in der Benutzeroberfläche von openEASE erzeugt werden können. Die Nutzer können ihren persönlichen API-Schlüssel in der Software ihrer Roboter zur Autorisierung einsetzen und müssen nicht ihr Passwort für openEASE hinterlegen. Derzeit bietet die API zum automatischen Zugriff mit API-Schlüssel Funktionen zum Starten und Beenden des zugehörigen Nutzercontainers sowie zur Autorisierung und zum Signalisieren von Aktivität.

4.6 Nutzercontainer

Im Zusammenspiel mit der Oberfläche von *Webrob* dienen Nutzercontainer der Verarbeitung von Anfragen zu protokollierten Experimentdaten in openEASE. Über die bereitgestellte *WebSocket*-Verbindung werden Prolog-Ausdrücke von JavaScript-Bibliotheken in der Nutzeroberfläche bzw. autonome Robotersystemen an die *KNOWROB-json_prolog*-Instanz im Nutzercontainer abgesetzt. In Rückrichtung werden die Ergebnisse der Ausdruckauswertung, sowie gegebenenfalls zusätzliche Daten für die Visualisierung (3D-Objekte, Parameter des Roboterplans bzw. Designatoren oder Statistiken) gesendet. Durch die Erstellung eigener Prolog-Module im Editor von *Webrob* ist es Nutzern freigestellt, den Funktionsumfang von openEASE beliebig zu erweitern.

Dazu muss für jeden Nutzer die Möglichkeit geschaffen werden, Daten zur Nutzung in openEASE speichern zu können.

Vor Umsetzung der Maßnahmen dieser Arbeit gab es zu diesem Zweck zwei Datencontainer (*user_data* und *knowrob_data*), von denen ersterer zur Speicherung aller Nutzerdaten und zweiterer zur Speicherung aller Experimentdaten vorgesehen war. Der Zugriff auf beide Container war sowohl in *Webrob* als auch in allen Nutzercontainern lesend und schreibend möglich, so dass jeder Nutzer Experimentdaten und die Daten anderer Nutzer lesen, schreiben und löschen konnte (Sicherheitsmangel Nr. 11 und 13). Daher wird als Lösungsmaßnahme eine vollständige Trennung der Nutzerdaten auf der Ebene von Datencontainern, sowie eine nur-lesende Einbindung der Experimentdaten in die Nutzercontainer vorgeschlagen und wie folgt umgesetzt: Für jeden Nutzer wird ein eigener Datencontainer angelegt. Diese Datencontainer werden ausschließlich in die Nutzercontainer der zugehörigen Nutzer eingebunden, was in Nutzercontainern den Zugriff auf ausschließlich eigene Nutzerdaten beschränkt. Die nur-lesende Einbindung der Experimentdaten konnte durch das Ändern eines einzigen Docker-Konfigurationswertes (*knowrob_data:ro* statt *knowrob_data*) vollzogen werden. Als problematisch stellte sich bei der Datentrennung der Umstand heraus, dass in den Docker-Container von *Webrob* nicht nachträglich die Datencontainer der einzelnen Nutzer eingebunden werden können, was zur Funktionslosigkeit des Editors geführt hätte. Abhilfe schafft das Konzept der intermediären Container: *Webrob* bindet den Datencontainer *lft_data* ein. Sollen nun Daten von oder in einem/einen Nutzer-Datencontainer kopiert werden, wird über die *Dockerbridge* ein temporärer Container gestartet, der sowohl *lft_data*, als auch den gewünschten Nutzer-Datencontainer einbindet und die gewünschten Daten kopiert. Nach Abschluss des Kopiervorgangs wird der temporäre Container terminiert und entfernt. Da das Erstellen und Terminieren von Dockercontainern sehr schnell ist, kann dies bei jeder im Editor möglichen Aktion ohne bemerkbare Zeitverzögerung durchgeführt werden.

Um Sicherheitsmangel Nr. 15 zu beheben, musste eine Authentisierung für die *WebSocket*-Verbindungen der Nutzercontainer eingeführt werden. Andernfalls wären beliebige Zugriffe auf laufende Nutzercontainer möglich, was auch die Datentrennung ad absurdum geführt hätte. Die Anwendung *rosbridge_server* zum Umsetzen der *WebSocket*-Verbindung zum ROS-System integriert bereits das Verfahren *rosauth*, welches den Mangel beheben kann. *rosauth* ist ein im ROS-System laufender ROS-Node, der als Service eine Authentifizierungsmethode anbietet. Sie nimmt als Eingabeparameter zwei IP-Adressen, den Gültigkeitsbeginn, das Gültigkeitsende, eine Berechtigungsstufe, ein Zufallswert, sowie einen *Message Authentication Code (MAC)* an. Der *MAC* besteht aus dem Hashwert (kryptographische Prüfsumme) der Konkatenation aller übrigen Eingabeparameter sowie eines gemeinsamen Geheimnisses. *rosauth* kennt das gemeinsame Geheimnis und berechnet den *MAC* aus den übermittelten Eingabeparametern neu. Stimmen der errechnete *MAC* sowie der übermittelte *MAC* nicht überein, so stimmt entweder das zur Erstellung verwendete Geheimnis, oder ein/mehrere Eingabeparameter nicht überein. *rosauth* kann so prüfen, ob ein Nutzer einen *MAC* aus einer vertrauenswürdigen Quelle, die im Besitz des gemeinsamen Geheimnis ist, bezogen hat. Diese vertrauenswürdige Quelle ist bei openEASE die *API* in

Webrob, die nur bei erfolgreicher Anmeldung oder unter Angabe eines gültigen API-Schlüssels die von *rosauth* geforderten Eingabeparameter inklusive MAC ausgibt. *rosbridge_server* kann nun so konfiguriert werden, dass nach Aufbau einer *WebSocket*-Verbindung die erste Nachricht eine *rosauth*-Nachricht sein muss. Bleibt diese aus, wird die Verbindung sofort getrennt. Wird eine *rosauth*-Nachricht übermittelt, wird sie an die *rosauth*-Node geschickt, welche dann den MAC validiert. Nur bei erfolgreicher Validierung bleibt die *WebSocket*-Verbindung geöffnet, womit eine Authentifizierung stattfindet und der Sicherheitsmangel effektiv behoben ist. Um beim Anmelden an openEASE eine Autorisierung durchführen zu können, wurde in *Webrob* in der für die *WebSocket*-Kommunikation zuständigen JavaScript-Bibliothek eine Änderung vorgenommen, die direkt nach Verbindungsaufbau eine Anfrage an die *Webrob-API* vornimmt und die notwendigen *rosauth*-Daten beschafft. Diese werden sogleich an die *WebSocket*-Verbindung weitergesendet, wodurch die Authentifizierung des Nutzers abgeschlossen ist.

Als letzte Schwachstelle der Nutzercontainer wurde die unbeschränkte Verfügbarkeit von Hardwareressourcen des Hosts ermittelt. Dadurch ist es Nutzern möglich, theoretisch sämtlichen Arbeitsspeicher oder einen großen Anteil der Prozessorleistung im Nutzercontainer zu beanspruchen, was die Verfügbarkeit anderer Nutzercontainer - im schlimmsten Fall sogar aller Container von openEASE - bedroht hätte. Die Behebung der Schwachstelle war jedoch trivial, da Docker die Möglichkeit zur Begrenzung der Ressourcen eines Containers bietet. Es mussten lediglich der Wert für die Anteile am Prozessor sowie der Wert für die maximal zulässige Arbeitsspeicherauslastung in der *Dockerbridge* beim Erstellen der Nutzercontainer mit übergeben werden. Experimentell wurde ermittelt, dass ein Nutzercontainer auch mit 20 Megabyte Arbeitsspeicher noch mit akzeptabler Geschwindigkeit läuft; da jedoch die Nutzungsgewohnheit und Anwendungsfälle der Nutzer in openEASE nicht verlässlich vorhergesehen werden kann, wurde die Grenze mit einem hohen Sicherheitsabstand auf 256 Megabyte festgelegt. Sollte selbst diese Grenze überschritten werden, so stehen jedem Nutzer zusätzlich 1 Gigabyte Auslagerungsspeicher (SWAP) auf der Festplatte zur Verfügung. Die Prozessoranteile können in Docker nur relativ vergeben werden: Ein Container hat in der Standardeinstellung 1024 Anteile, d.h. zwei gleichzeitig laufende Standardcontainer würden bei voller Auslastung jeweils 50% der Prozessorleistung erhalten. Ein Paar von Containern mit 512 und 1024 Anteilen würde folglich $\frac{512}{1536} \times 100\% = 33,3\%$ bzw. $\frac{1024}{1536} \times 100\% = 66,6\%$ Prozessorleistung erhalten. Für Nutzercontainer in openEASE wurde der Wert zunächst auf 256 gesetzt, was sie vier Mal so niedrig priorisiert wie die für den Betrieb von openEASE notwendigen Container (*Webrob*, *dockerbridge*, *nginx*, *mongo_db* und *postgres_db*). Es bleibt zu beobachten, ob die Limitierungen in einem hier nicht bedachten Anwendungsfall Probleme bereiten, oder ob sie noch weiter ausgebaut werden können. Davon hängt auch ab, wie viele Nutzer openEASE gleichzeitig nutzen können; hier ist als zusätzliche, noch nicht umgesetzte Maßnahme eine Beschränkung der maximal gleichzeitig aktiven Nutzer zu prüfen.

4.7 Allgemeine Maßnahmen

Während der Analyse wurde festgestellt, dass die meisten Dienste von openEASE mit öffentlichen Portfreigaben versehen waren. Im Fall des MongoDB-Containers führte dies zu der Schwachstelle Nr. 12, wo ein lesender/schreibender Zugriff auf alle Datenbanken aus dem Internet heraus möglich war. Um die Anzahl der Zugänge zu openEASE auf die minimal notwendige Menge zu reduzieren, wurden alle Portfreigaben entweder vollständig entfernt, oder auf das lokale interne Netz beschränkt. Lediglich die Ports des *nginx*-Container werden für öffentliche Netze freigegeben, was den *nginx*-Container zum alleinigen Zugangspunkt zu openEASE macht.

Um den korrekten Zugang zu openEASE unter Benutzung von TLS und der WebSocket-Verbindung mit Autorisierung zu demonstrieren, wurde ein Javaprojekt innerhalb des openEASE-Projektverzeichnis (Ordner `oeclient`) erstellt, welches eine rudimentäre API für die Steuerung des eigenen Containers in openEASE bereitstellt. Außerdem wurde ein kurzes Beispiel zur Demonstration der korrekten Nutzung der API mitgeliefert.

Zur Erleichterung der Entwicklung und des Betriebs von openEASE sind während der Arbeit einige Skripte erstellt und verbessert worden. Dazu zählen das *start-webrob* Skript, welches von Redundanzen befreit, neu strukturiert und um einen Kommandozeilenparameter für den Fehlerbehebungsmodus (`EASE_DEBUG`) erweitert wurde. Neu erstellt wurde das *build*-Skript zum Bauen ausgewählter/aller Docker-Images für openEASE aus den im Projektordner befindlichen Dockerfile-Dateien, sowie ein *stop-webrob* Skript zum vollständigen Anhalten aller Container von openEASE. Außerdem wurde ein *dockerclean* Skript zum Entfernen verwaister Docker-Images geschrieben, die auf einem Entwicklungsrechner nach längerer Zeit 500 Gigabyte und mehr belegen können.

4.8 Ausstehende Umsetzungen und Zusammenfassung

Insgesamt blieben von den 16 gefundenen Sicherheitsmängeln nur zwei ohne eine implementierte Lösung übrig: Nr. 12 (unauthentisierter MongoDB-Zugang) und Nr. 16 (Ausführung unerwünschter Prolog- und Javafunktionen im Prolog-Interpreter).

Zur Behebung des Problems der unauthentisierten MongoDB-Verbindungen ist zunächst zu klären, wie die MongoDB-Instanz im Docker-Container so konfiguriert werden kann, dass eine Authentifizierung erforderlich wird. Weiterhin müssen Benutzer festgelegt werden, die Schreibrechte für die MongoDB besitzen dürfen, sowie einen Benutzer, der nur Leserechte für alle Datenbanken besitzt. Letzterer Benutzer muss dann für alle Abfragen an die MongoDB innerhalb von Prolog-Ausdrücken genutzt werden. Dazu ist es erforderlich, die von Prolog genutzte Javaklasse⁶ so

⁶https://github.com/knowrob/knowrob/blob/indigo-devel/knowrob_mongo/knowrob_mongo/src/main/java/

umzuschreiben, dass sie eine Autorisierung für MongoDB unterstützt. Bevor diese Maßnahmen nicht umgesetzt sind, ist es Nutzern weiterhin möglich, aus dem Nutzercontainer heraus beliebig auf die MongoDB-Datenbank mit den Experimentdaten zuzugreifen, Änderungen durchzuführen und (un)absichtlich die Funktionalität von openEASE zu beeinträchtigen.

Als letzter offener Punkt bleibt das uneingeschränkte Interpretieren von Prolog-Ausdrücken. Prolog ist eine universelle Programmiersprache, die zusammen mit den Erweiterungen des Interpreters SWI-Prolog Funktionen zum Zugriff auf Systemressourcen und zur Ausführung von Javamethoden genutzt werden kann. Dadurch ist es Benutzern nicht nur möglich, die Wissensdatenbank von openEASE zu befragen, sondern über den vollen Umfang der Programmiersprachen Prolog und Java hinaus im Nutzercontainer zu agieren. Nutzt man beispielsweise unter Verwendung von JPL Funktionen aus Java zum Herunterladen einer Datei aus dem Internet und führt diese anschließend aus, kann so ein natives Programme im Nutzercontainer gestartet werden, welches Zugriff auf alle im Nutzercontainer verfügbaren Ressourcen hat und Aufrufe von Kernel-Funktionen tätigen kann. Das Unterbinden der Ausführung derartiger Prolog-Anfragen ist schwierig, da die gesamte Funktionalität von openEASE auf der Auswertung von Prolog-Ausdrücken basiert. Die folgenden Möglichkeiten wurden zur Umsetzung einer Ausführungsverhinderung von unzulässigen Prolog-Ausdrücken betrachtet.

Ein einfacher Ansatz, um missbräuchliche Prolog-Ausdrücke zu unterbinden, ist das Deaktivieren der freien Ausdruckeingabe und das ausschließliche Zulassen von Ausdrücken aus einer vorgegebenen Liste. Damit werden jedoch legitime eigene oder angepasste vorgegebene Anfragen an openEASE unterbunden, was eine nicht hinnehmbare Einschränkung der Funktionalität von openEASE darstellt. Daneben gibt es noch den Ansatz des Black- bzw. Whitelisting von Prolog-Ausdrücken: Beim Blacklisting wird eine Liste mit unzulässigen Prolog-Ausdrücken geführt und Nutzereingaben auf Übereinstimmung mit der Liste überprüft. Eine Übereinstimmung führt zur Ablehnung der Anfrage. Genau umgekehrt funktioniert das Whitelisting: Dort werden zulässige Ausdrücke in einer Liste geführt und Nutzereingaben dann abgelehnt, wenn sie nicht ausschließlich aus auf der Whitelist befindlichen Ausdrücken bestehen. Der Vorteil des Blacklistings ist, dass initial eine Sammlung unzulässiger Ausdrücke blockiert werden kann, was einen Angriff erschwert. Die Pflege der Blacklist ist ebenfalls hinsichtlich der Funktionalität von openEASE einfacher, da neue, zulässige Ausdrücke bei Einführung direkt genutzt werden können. Trotzdem sind Blacklisting-Verfahren ungeeignet zum zuverlässigen Verhindern von unzulässigen Ausdrücken: Das Verfahren widerspricht dem bereits bei der MongoDB vermissten Grundsatz der standardmäßigen Verweigerung von Berechtigungen (siehe 3.2.4.2 auf Seite 28). Bei ausreichender Geduld des Angreifers kann ein Angriff möglicherweise durch einen Ausdruck durchgeführt werden, der von den Urhebern der Blacklist als ungefährlich eingestuft wurde, oder ihnen bei der Verfassung der Liste unbekannt war. Außerdem können neue unzulässige Ausdrücke etwa bei Aktualisierungen verwendeter Prolog-Module oder des Prolog-Interpreters eingeführt werden. Whitelisting ist hinsichtlich der Sicherheit besser geeignet, da ihre Urheber jeden Aus-

org/knowrob/interfaces/mongo/MongoDBInterface.java

druck explizit angeben müssen, der ausgeführt werden darf. Aber auch hier ist die Gefahr des irrtümlichen Whitelisting von Ausdrücken, deren Angriffspotential unterschätzt wurde, vorhanden. Die Pflege der Whitelist ist zusätzlich deutlich aufwändiger, da zunächst alle Ausdrücke, die von Nutzern legitim gebraucht werden, ermittelt und festgehalten werden müssen. Dies macht eine Einführung auch sehr aufwändig, da sie nicht schrittweise erfolgen kann, ohne Funktionseinschränkungen von openEASE hinnehmen zu müssen. Erweiterungen von openEASE mit neuen Ausdrücken erfordern dann auch jedes Mal eine Anpassung der Whitelist. Neben dem hohen Aufwand der Pflege dieser Listen ist auch die Durchsetzung davon schwierig. Nutzer können Prolog-Ausdrücke über den Editor von *Webrob* oder über die Prolog-Konsole in der Oberfläche von openEASE angeben. Es muss also entweder eine Modifikation des Interpreters SWI-Prolog zur Forcierung der Black- bzw. Whitelist vorgenommen werden, oder ein eigener Prolog-Parser zur Prüfung der Nutzereingaben in den Editor und in *json_prolog* integriert werden.

Die ermittelten Lösungsvorschläge für den Sicherheitsmangel Nr. 16 sind entweder nicht ausreichend für eine abschließende Behebung des Mangels (Blacklist), unzumutbar für die Nutzer von openEASE (keine eigenen/angepassten Prolog-Ausdrücke) oder unzumutbar für die Entwickler von openEASE (Whitelist). Daher sollte erwogen werden, sich mit dem Sicherheitsmangel zu arrangieren, indem die Auswirkungen eines Angriffs reduziert werden. Dies ist im Rahmen dieser Arbeit schon größtenteils geschehen, übrig bleibt nur noch der Mangel der MongoDB (Nr. 12). Eventuell ist es auch möglich, durch Beschränkung des Netzwerkverkehrs für Nutzercontainer den Zugriff auf das Internet aus dem Container heraus zu blockieren bzw. nur auf Netzwerke der IAI zu erlauben.

Nach Umsetzung aller in diesem Kapitel beschriebenen Maßnahmen hat sich das Komponentendiagramm von openEASE geändert (siehe Abbildung 4.1 auf Seite 50). Es ist erkennbar, dass nur noch ein öffentlicher Zugang in das System existiert und dass der Zugriff auf unsichere Ressourcen wie die Docker-API von außerhalb des Hosts erreichbaren Komponenten abgeschirmt ist. Weiterhin ist der Zugriff auf die Datencontainer eingeschränkt, so ist etwa der Zugriff auf *knowrob_data* nur noch lesend möglich. Entsprechend der beschriebenen offenen Sicherheitsmängel sind nur noch die KNOWROB-Nutzercontainer und die MongoDB gefährdet, die Nutzercontainer dabei auch nur schwach.

Für eine Erklärung des Diagrammaufbaus und der verwendeten Farben siehe Kapitel 3.3 auf Seite 33.

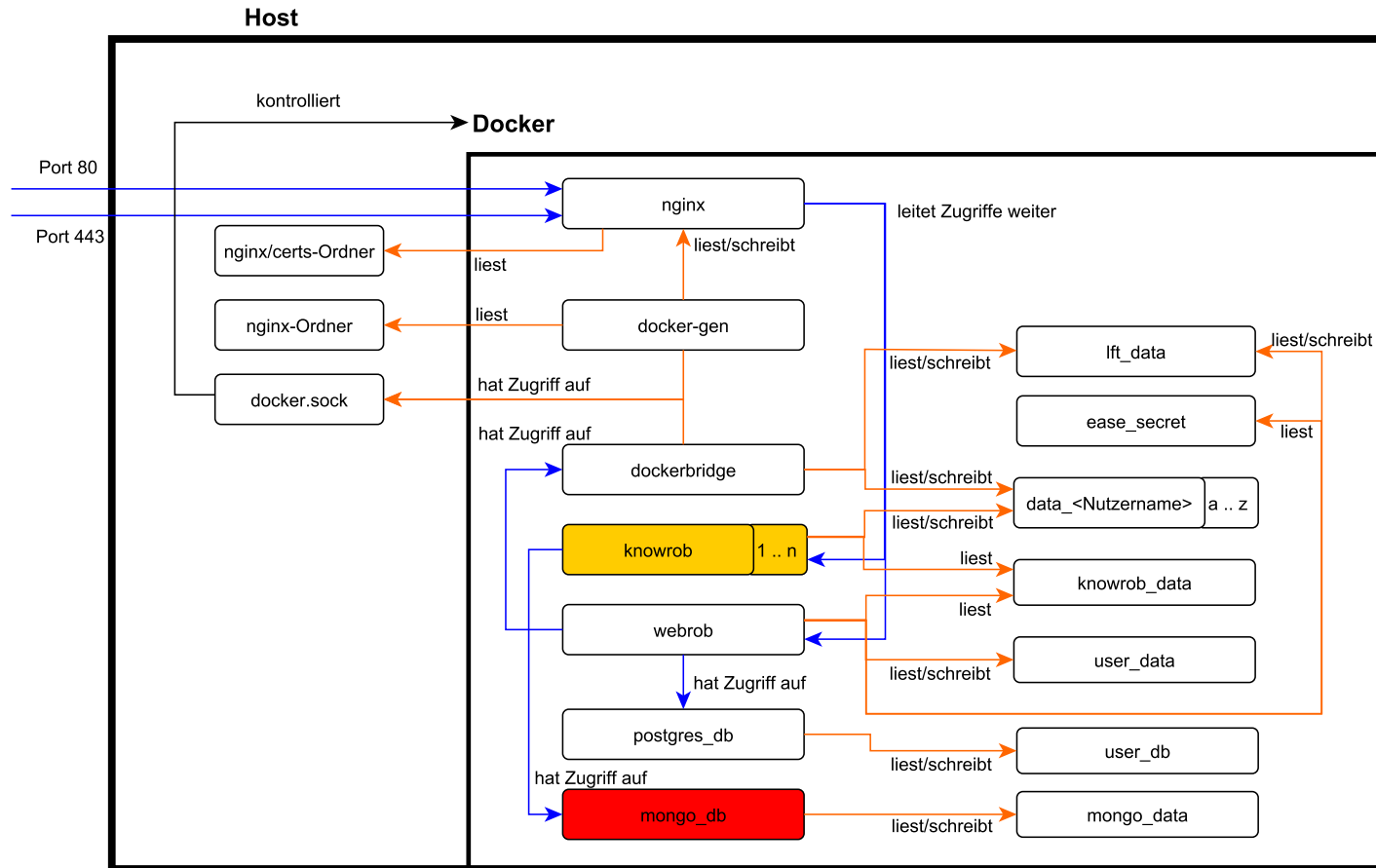


Abbildung 4.1 Komponentendiagramm nach Umsetzung der Verbesserungen

Diskussion der Arbeitsergebnisse

In diesem Kapitel wird eine abschließende Diskussion zu der in dieser Arbeit durchgeführten Sicherheitsanalyse und die im Anschluss ermittelten und implementierten Absicherungsmaßnahmen geführt.

5.1 Prüfung der Transportverschlüsselung

Um die Wirksamkeit und die Qualität der in Kapitel 4.3 auf Seite 39 beschriebenen Einführung von Transportverschlüsselung zu testen, wurde das webbasierte Prüfprogramm „SSL Server Test“ der Firma „Qualys, Inc.“ verwendet¹. Es analysiert die Konfiguration des Webservers auf die Verwendung von unsicheren Verschlüsselungs- und Prüfsummenverfahren, prüft die Gültigkeit und die Vertrauenskette der eingesetzten Zertifikate und simuliert den Verbindungsaufbau von gängigen Clients (Browser, Programmbibliotheken, Suchmaschinenroboter etc.). Auch werden Schwachstellen in der Implementierung der TLS-Bibliothek des Webservers geprüft (Beispielhaft genannt seien BEAST, POODLE und Heartbleed), die die Sicherheit der Verbindung gefährden können. Es vergibt Noten für die Güte der TLS-Implementierung des getesteten Webservers nach dem US-amerikanischen System (A+ bis F, A+ ist die bestmögliche Note), wobei A+ nur bei Verwendung und Unterstützung der neuesten Versionen und Funktionen vergeben wird. Ein Test des openEASE-Servers nach Ausrollen der TLS-Einführung in *nginx* ergab ein A+ (siehe Abbildung 5.1), was ein deutlicher Hinweis auf die Qualität der Umsetzung ist.

¹<https://www.ssllabs.com/ssltest/>

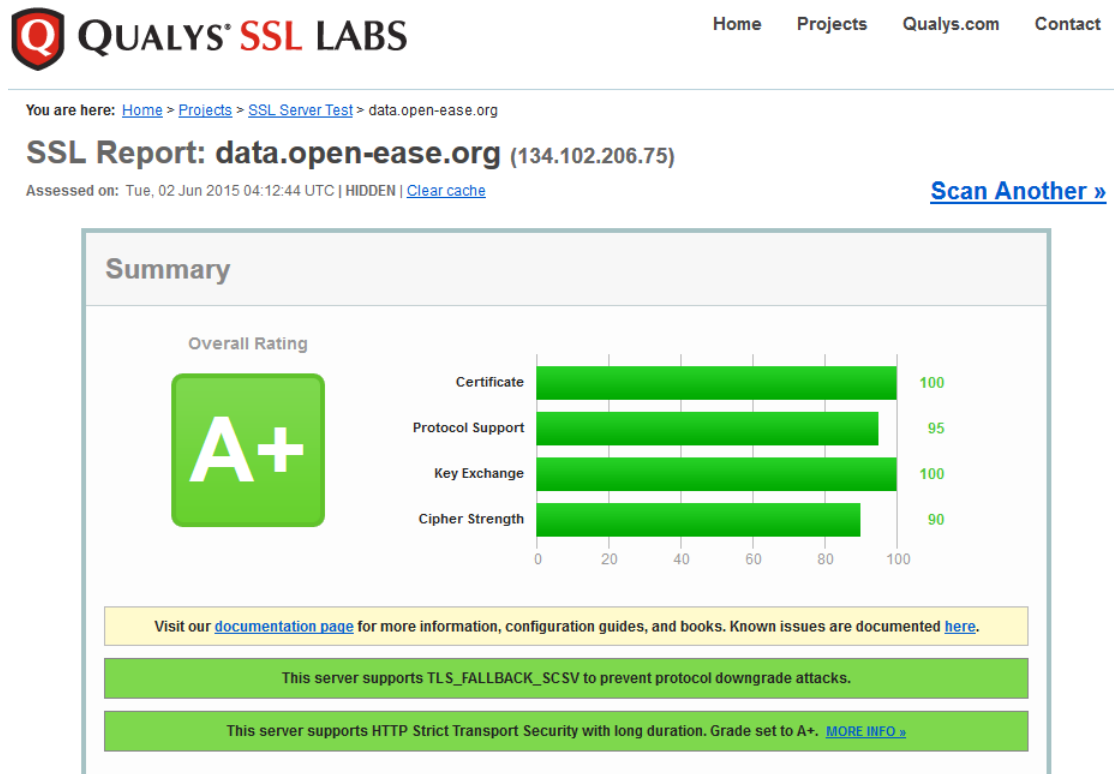


Abbildung 5.1 TLS-Testergebnis für openEASE im „SSL Server Test“ der Firma „Qualys, Inc.“

5.2 Penetrationstests

Bislang wurden Angriffe auf openEASE in dieser Arbeit nur theoretisch skizziert. Um die tatsächliche Gefährdung durch die beschriebenen Angriffsszenarien zu demonstrieren, werden in diesem Kapitel zwei Angriffe auf openEASE durchgeführt und dokumentiert. Es wird darauf hingewiesen, dass die Angriffe nicht auf die Produktivinstanz von openEASE gefahren wurden, sondern auf eine lokal eingerichtete Kopie (*http://localhost*) mit dem Versionsstand vor Beginn dieser Arbeit, um echte Nutzerdaten und den Host von openEASE durch die Angriffe nicht zu gefährden.

5.2.1 Administrationsrechte auf dem Host

Dieser Angriff ermöglicht es, Administrationsrechte auf dem Hostrechner zu erlangen. Benötigt werden: Ein mit dem Internet verbundener Computer, der Browser Firefox², das Firefox-Plugin *Tamper Data*³ und einen SSH-Client mit einem Private/Public-Schlüsselpaar. Folgende Schritte waren zur Durchführung des Angriffes notwendig:

1. Man öffnet openEASE im Browser, registriert ein neues Benutzerkonto und meldet sich an. In der oberen Navigationsleiste von openEASE öffnet man den Editor und legt ein neues Package an.
2. Das Plugin *Tamper Data* muss aktiviert werden. Dazu klickt man in der Firefox-Menüleiste auf Extras und auf *Tamper Data*, in dem sich öffnenden Fenster dann auf „Tamper beginnen“.
3. Im Editor klickt man auf eine der vorhandenen Dateien. Die Abfrage von *Tamper Data* wird mit Klick auf „Tamper“ bestätigt.
4. Im sich öffnenden Fenster entfernt man den einzigen POST-Parameter und fügt einen ungültigen Parameter hinzu, etwa `{"file":123}` und schickt die modifizierte Anfrage ab.
5. Anschließend stoppt man das Abfangen weiterer Requests und wiederholt die modifizierte Anfrage im Browser-Tab. Dazu wählt man sie aus der Liste in *Tamper Data* aus, klickt rechts auf sie und wählt „Im Browser wiederholen“.
6. Es öffnet sich ein neuer Browser-Tab mit einer Python-Fehlermeldung und einer Debug-Konsole. Durch Eingabe der im Anhang A.5 auf Seite 63 aufgelisteten Befehle wird durch Nutzung der Docker-API der öffentliche SSH-Schlüssel des Angreifers auf dem Docker-Host hinterlegt und ihm administrative Rechte einräumt.
7. Zum Schluss meldet man sich über SSH auf dem Host an unter Verwendung des privaten SSH-Schlüssels. Mit dem Programm *sudo* können beliebige Befehle als Administrator ausgeführt werden.

Möglich wird diese Vorgehensweise durch Sicherheitsmangel Nr. 8 (3.2.3.2 auf Seite 25) und Nr. 9 (3.2.3.3 auf Seite 26). Die in Schritt 3 beim Klicken auf die Datei ausgelöste Anfrage zum Lesen des Dateiinhaltes prüft den angegebenen Dateinamen nicht, was dazu führt, dass die Übermittlung der Zahl 123 statt des Dateinamens `'init.pl'` in Schritt 4 einen Fehler in *Webrob* auslöst. Die erneute Übermittlung im Browser in Schritt 5 ermöglicht das Anzeigen der Debug-Konsole. Die Erläuterung der Python-Befehle ist im Anhang A.5 auf Seite 63 zu finden. Screenshots von der Durchführung des Angriffes sind auf der dieser Arbeit beiliegenden CD zu entnehmen (Ordner `pentest/root`).

Folgende Vereinfachungen wurden zur Verkürzung des Angriffes angenommen:

- *Auf dem Host ist ein SSH-Server installiert* - Da ein Linux-basiertes Serversystem üblicherweise aus der Ferne gesteuert wird, kann vom Vorhandensein eines SSH-Servers ausgegangen

²<https://www.mozilla.org/de/firefox/new/>

³<https://addons.mozilla.org/de/firefox/addon/tamper-data/>

werden.

- *Der Benutzername ist bereits bekannt* - Auf dem nachgestellten System war der Benutzername `moritz` bereits bekannt, dies ist jedoch keine Voraussetzung, da z.B. vorher die `/etc/passwd`-Datei mit allen vorhandenen Nutzern und deren Heimverzeichnissen ausgelesen werden könnte.
- *Sudo ist auf dem Host installiert* - Sudo wird zum temporären Arbeiten mit administrativen Rechten häufig eingesetzt, ist aber hier prinzipiell nicht notwendig. Man könnte auch eine Shell-Anwendung in das Heimverzeichnis des Nutzers `moritz` kopieren, sie dem Nutzer `root` zugehörig machen und das SUID-Bit setzen, was bei der Ausführung als Nutzer `moritz` trotzdem eine Root-Shell startet (Diese Abwandlung des Angriffes wurde übernommen von Carlos Reventlovs Schilderung einer Docker-Sicherheitslücke⁴).

5.2.2 Zugriff auf fremde Nutzerdaten

Dieser Angriff demonstriert den Zugriff auf fremde Nutzerdaten. Da eine lokale Kopie von openEASE verwendet wurde und so keine echten Nutzerdaten vorhanden sind, sind vor dem Angriff zwei Benutzer angelegt worden: Ein Benutzer mit dem Namen *geheim* und einer Datei *firmengeheimnis.pl*, sowie ein Benutzer *tutor* mit der Datei *musterloesung.pl*. Zur Durchführung des Angriffes ist nur ein Browser notwendig. Folgende Schritte waren zur Durchführung des Angriffes notwendig:

1. Man öffnet openEASE im Browser, registriert ein neues Benutzerkonto und meldet sich an.
2. Man öffnet den Editor durch Klicken auf den entsprechenden Eintrag in der Navigationsleiste von openEASE.
3. Nun muss ein neues Package erstellt werden. Dazu fährt man mit der Maus über den Eintrag „Package“ und wählt „New“.
4. Die sich öffnende Abfragebox wird mit OK bestätigt, wodurch ein neues Package mit Namen „my_experiment“ angelegt wurde.
5. Im Anschluss wird direkt noch ein neues Package erstellt, in der sich öffnenden Abfragebox muss als Name „./“ eingegeben werden.
6. Die Dateien *musterloesung.pl* und *firmengeheimnis.pl* sind dann bereits in der Auswahlspalte des Editors zu sehen. Um sie herunterzuladen, fährt man über den Eintrag „Package“ und wählt Download.

Die fehlende Nutzerdatentrennung führt dazu, dass über die fehlende Überprüfung der Eingabeparameter des Editors alle Nutzerdaten eingesehen werden können (Sicherheitsmangel Nr. 11). In *Webrob* wird das zuletzt angelegte Package „./“ zu `/home/ros/user_data/` ausgewertet, da z.B. der Schrägstrich als Zeichen zugelassen wurde. Mithilfe des im vorherigen Angriff verwend-

⁴<http://reventlov.com/advisories/using-the-docker-command-to-root-the-host>, Abruf am 21. Mai 2015

ten Firefox-Plugin *Tamper Data*⁵ können zusätzlich, durch modifizierte Aufrufe der Funktionen zum Speichern oder Löschen von Dateien, fremde Nutzerdaten verändert und entfernt werden. Screenshots von der Durchführung des Angriffes sind auf der dieser Arbeit beiliegenden CD zu entnehmen (Ordner `pentest/data`).

5.2.3 Situation nach Behebung der Sicherheitsmängel

Die zuvor ausgeführten erfolgreichen Penetrationstests wurden auf dem Versionsstand von openEASE nach Implementierung der Verbesserungen in dieser Arbeit erneut durchgeführt.

Der erste Angriff scheiterte in dem Schritt, wo die Debug-Konsole durch Provokation eines Fehlers erscheinen sollte. Durch die Deaktivierung dieser ist es Nutzern nicht möglich, bei Fehlern eigene Python-Anweisungen auf dem Server auszuführen. Um auch die Schließung der Docker-API Schwachstelle zu testen, wurde temporär die Debug-Konsole wieder aktiviert. Es war darin nicht mehr möglich, die Docker-API direkt zu benutzen, sondern es konnten nur Anfragen an die *Dockerbridge* abgesetzt werden. Es gelang dabei nicht, unter Benutzung der angebotenen Schnittstellen Dateien oder Ordner des Hostsystems in einen Docker-Container einzubinden, wodurch auch der zweite Teil des Angriffes eigenständig unterbunden wurde.

Der zweite Angriff für das Auslesen der Nutzerdaten führte bei Eingabe von „./“ als Package-Namen zu keiner Reaktion von openEASE. Erstens hat die *Dockerbridge* durch Filtern von Eingabeparametern die Zeichenfolge als unzulässig erkannt und daher blockiert. Zweitens würde bei einer unterbleibenden Filterung der Zugriff nur auf den exklusiv zum Nutzer zugehörigen Datencontainer möglich sein, so dass fremde Nutzerdaten nie in den Einflussbereich anderer Nutzers gelangen. Anzumerken sei dazu noch, dass bei aktivierter Debug-Konsole der Zugriff auf alle Nutzerdaten darüber möglich ist. Aufgrund der Tatsache, dass der Editor je nach angemeldetem Nutzer auf dessen Nutzerdaten zugreifen muss, hätte sonst für jeden Nutzer eine eigene Instanz des Editors bereitgestellt werden müssen, um diese Variante des Angriffes zu verhindern. Notwendig ist dies nicht, da die Debug-Konsole nun standardmäßig deaktiviert ist.

5.3 Fazit

In dieser Arbeit wurde eine umfassende Sicherheitsanalyse der Webanwendung openEASE durchgeführt, in der kritische Sicherheitslücken und ernste Defizite bei Stabilität und Verfügbarkeit erkannt wurden. In der anschließenden Erarbeitung und Umsetzung eines Sicherheitskonzeptes zur Behebung der Schwachstellen aus der Analyse wurde nicht nur für jede Schwachstelle

⁵<https://addons.mozilla.org/de/firefox/addon/tamper-data/>

eine mögliche Lösung präsentiert, sondern auch bis auf zwei Ausnahmen in openEASE implementiert und integriert. Die Wirksamkeit der umgesetzten Maßnahmen wird durch tatsächlich durchgeführte Angriffe, die vor Umsetzung der Maßnahmen erfolgreich und nach Umsetzung der Maßnahmen erfolglos waren, anschaulich gezeigt.

Während die Prinzipien der Informationssicherheit durch eine Menge von anerkannten Publikationen als Grundlage für die Erklärung von in dieser Arbeit analysierter Schwachstellen von openEASE geeignet ist, ist die Behebung der Schwachstellen deutlich schwieriger zu begründen. Die implementierten und vorgeschlagenen Lösungen können qualitativ und quantitativ nur so umfassend sein, wie es nach Meinung des Autors für die Erfüllung der Informationssicherheitsprinzipien notwendig ist. Wie schon in der Zusammenfassung des Analysekapitels (3.4 auf Seite 35) erwähnt, ist es wahrscheinlich, dass trotz systematischer Untersuchung einige Sicherheitslücken in openEASE nicht oder nicht vollständig aufgedeckt wurden. Gleichmaßen ist es möglich, dass manche hier vorgestellten Fehlerbehebungen nicht alle möglichen Angriffswege schließen. Es wurde in die Umsetzung der Verbesserungen in openEASE sehr viel Aufwand investiert, so dass zum Abschluss dieser Arbeit die vollständig distanzierte Betrachtung und Bewertung nicht mehr möglich ist. Aus diesem Grund ist es üblich, derartige Analysen und Konzepte durch dritte Sachverständige prüfen zu lassen, was allerdings für den Umfang dieser Arbeit unangemessen wäre.

Nichtsdestotrotz ist durch die Schließung der ermittelten Sicherheitslücken die Qualität von openEASE hinsichtlich Vertraulichkeit, Integrität und Verfügbarkeit deutlich gestiegen. Als direkte Auswirkung der gesteigerten Verfügbarkeit ist die Stabilität von openEASE bei der alltäglichen Nutzung bemerkbar: Das zuverlässige Starten und Beenden von Nutzercontainern durch die *Dockerbridge* hat einige Fehlersituationen, die bei der Untersuchung von openEASE aufgefallen sind und zu einem funktionslosen Zustand geführt haben, nachhaltig beseitigt. Die Ergebnisse dieser Arbeit werden langfristig dazu beitragen, erfolgsverhindernde Faktoren wie Datendiebstahl, langsame Anfragenverarbeitung oder Systemausfälle von openEASE zu verhindern.

5.4 Ausblick

Nach Vollendung dieser Arbeit sind weitere mögliche Schritte zur Verbesserung der Sicherheit in openEASE durchführbar. Sie sind in dieser Arbeit in Kapitel 4.8 auf Seite 47 beschrieben und umfassen Lösungswege für Sicherheitsmängel, deren Umsetzung aus verschiedenen Gründen im Rahmen dieser Arbeit nicht möglich oder nicht sinnvoll war.

Weiterhin ist die Einführung einer Administrationsoberfläche zur leichteren Verwaltung von Sicherheitseinstellungen für Benutzeraccounts in openEASE empfehlenswert. So könnte beispielsweise eine Verwaltung und Kategorisierung nach Benutzerrollen vorgenommen werden, bei der

es privilegierte Nutzer gibt, die etwa bei Präsentationen von openEASE mehr Berechtigungen und Ressourcen zur Verfügung stehen haben.

Für die weitere Entwicklung von openEASE sollte zukünftig auch immer die Informationssicherheit bedacht werden: Es hat sich als hilfreich erwiesen, sich in die Rolle eines böswilligen Angreifers zu versetzen, Angriffsszenarien gegen die eigene Anwendung zu entwickeln und Angriffe mit verschiedenen böswilligen Zielen (Zerstörung, Datendiebstahl, Systemkontrolle etc.) testweise durchzuführen. Bei der Implementierung neuer Funktionen sollten die Entwickler sich selbst fragen: Kann damit die Sicherheit von openEASE geschwächt werden? Kann ein Angreifer von der Nutzung dieser Funktion Vorteile gewinnen? Sind die Berechtigungen, mit denen die Funktion ausgeführt wird, wirklich notwendig, oder können sie eingeschränkt werden? Mit Beachtung dieser Hinweise kann die Sicherheit von openEASE auch zukünftig verbessert und kontinuierlich auf hohem Niveau gehalten werden.

Appendix

A.1 Abbildungsverzeichnis

2.1	Die Oberfläche von openEASE	9
3.1	Virtuelle Maschine und Docker	18
3.2	Ausführung eines eigenen Kommandozeilenskriptes über die openEASE-Oberfläche	33
3.3	Komponentendiagramm von openEASE	34
4.1	Komponentendiagramm nach Umsetzung der Verbesserungen	50
5.1	TLS-Testergebnis für openEASE im „SSL Server Test“ der Firma „Qualys, Inc.“	52

A.2 Literatur

[Bis04] Matt Bishop. *Introduction to Computer Security*. Addison-Wesley Professional, 2004. ISBN: 0321247442.

[BRS15] Maik Baumgärtner, Sven Röbel und Jörg Schindler. »Cyberattacke auf Bundestag: Offenbar auch Rechner von Regierungsmitgliedern betroffen«. In: *Spiegel Online* (19. Mai 2015). URL: <http://www.spiegel.de/netzwelt/netzpolitik/cyberangriff-auf-bundestag-offenbar-auch-rechner-von-regierungsmitgliedern-betroffen-a-1034588.html> (abgerufen am 27.05.2015).

[BTW15] M. Beetz, M. Tenorth und J. Winkler. »Open-EASE – A Knowledge Processing Service for Robots and Robotics/AI Researchers«. In: *IEEE International Conference on Robotics and Automation (ICRA), Seattle, Washington, USA*. 2015.

- [Bui15] Thanh Bui. »Analysis of Docker Security«. In: *arXiv:1501.02967 [cs]* (13. Jan. 2015). arXiv: *1501.02967*. URL: <http://arxiv.org/abs/1501.02967> (abgerufen am 09.04.2015).
- [HGP15] Jens Heyens, Kai Greshake und Eric Petryka. *MongoDB databases at risk*. Jan. 2015. URL: http://www.cispa.saarland/wp-content/uploads/2015/02/MongoDB_documentation.pdf (abgerufen am 25.05.2015).
- [Men04] Paul Menage. *CGROUPS*. 2004. URL: <https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt> (abgerufen am 02.06.2015).
- [NB12] D. Nyga und M. Beetz. »Everything robots always wanted to know about housework (but were afraid to ask)«. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Okt. 2012, S. 243–250. DOI: *10.1109/IROS.2012.6385923*.
- [SS75] J.H. Saltzer und M.D. Schroeder. »The protection of information in computer systems«. In: *Proceedings of the IEEE* 63.9 (Sep. 1975), S. 1278–1308. ISSN: 0018-9219. DOI: *10.1109/PROC.1975.9939*.
- [TB13] Moritz Tenorth und Michael Beetz. »KnowRob: A knowledge processing infrastructure for cognition-enabled robots«. In: *The International Journal of Robotics Research* 32.5 (1. Apr. 2013), S. 566–590. ISSN: 0278-3649, 1741-3176. DOI: *10.1177/0278364913481635*. URL: <http://ijr.sagepub.com/content/32/5/566> (abgerufen am 09.04.2015).
- [Ten+15] Moritz Tenorth, Jan Winkler, Daniel Beßler und Michael Beetz. »Open-EASE: A Cloud-Based Knowledge Service for Autonomous Learning«. In: *KI - Künstliche Intelligenz* (30. Apr. 2015), S. 1–5. ISSN: 0933-1875, 1610-1987. DOI: *10.1007/s13218-015-0364-1*. URL: <http://link.springer.com/article/10.1007/s13218-015-0364-1> (abgerufen am 02.06.2015).

A.3 Liste der Abkürzungen

DoS Denial of Service, S. 6–8, 20, 24, 31

IAI AG Künstliche Intelligenz, S. 1, 8, 11, 15, 49

IDS Intrusion Detection System, S. 6

LSM Linux Security Module, S. 20

MAC Message Authentication Code, S. 45, 46

MitM Man-in-the-Middle, S. 5, 7

ROS Robot Operating System, S. 16, 21, 27, 29–32, 45

RPC Remote Procedure Call, S. 30, 40, 41

TLS Transport Layer Security, S. 4, 5, 39, 47, 51, 52, 59, 62

WSGI Python Web Server Gateway Interface, S. 42, 43

XSS Cross-Site-Scripting, S. 7

A.4 Glossar

API

Engl. für Application Programming Interface. Stellt eine spezifizizierte Schnittstelle zu Funktionen einer Anwendung dar.

S. 19, 21–24, 26, 39, 40, 43–47, 49, 53, 55, 64

framework

Eine Sammlung von Programmbibliotheken und Anwendungen innerhalb eines bestimmten Kontexts, innerhalb dessen die Entwicklung eigener Anwendungen durch Vorgabe fertiger Standardroutinen und Softwarearchitekturen vereinfacht werden kann.

S. 23, 25, 42

HMAC

Kurz für Keyed-Hash Message Authentication Code. Ein Verfahren, bei dem die Integrität einer Nachricht durch das Bilden einer Prüfsumme über die Nachricht und anschließendes Verschlüsseln der Prüfsumme sichergestellt wird. Entschlüsselt der Empfänger der Nachricht die Prüfsumme und vergleicht sie mit der errechneten Prüfsumme für die erhaltene Nachricht, kann so eine Manipulation festgestellt werden.

S. 39

honeypot

Ein absichtlich unsicher konfigurierter Dienst im Internet, um Angreifer anzulocken und deren Angriffstechniken und Aktionen zu beobachten.

S. 29

JPL

Ein für den Prolog-Interpreter SWI-Prolog existierendes Modul zum bidirektionalen Zugriff von Java auf Prolog und umgekehrt. Es ermöglicht den Aufruf von Javamethoden aus Prolog und das Absetzen von Prolog-Ausdrücken an SWI-Prolog aus Javamethoden.

S. 32, 48

JSON

Engl. für JavaScript Object Notation und ist eine einfache Notationsform für strukturierte Daten. Beispiel: { "Name": "Erika Musterfrau", "Alter" : 51, "Wohnort" : "Berlin" }
S. 19, 30, 40, 41

Kernel

Zentrale Softwarekomponente eines Betriebssystems, welche die grundlegenden Funktionen für darauf laufende Anwendersoftware bereitstellt.
S. 19–21, 48

Mount

Einbindung eines Dateisystem in ein Linux-System.
S. 19

NAT

Engl. für Network Address Traversal, ein Verfahren bei dem eine öffentliche IP-Adresse auf ein Netz von mehreren privaten IP-Adressen aufgeteilt wird. Es wird etwa in allen üblichen Routern für den Heiminternetzugang angewendet.
S. 30

Ontologie

Repräsentation von Wissen in Daten in einer maschinenlesbaren Form, die ein automatisiertes Ziehen von Schlussfolgerungen darauf (sog. Inferenz) erlauben.
S. 29

SSH

Engl. für Secure Shell und definiert ein Protokoll zur sicheren Bedienung einer Kommandozeile auf einem entfernten Computer.
S. 14, 53

SSL

Engl. für Secure Socket Layer, in der neuesten Version zu TLS umbenannt. Es bezeichnet ein weitverbreitetes Protokoll zur Sicherung von Vertraulichkeit und Integrität von Verbindungen im Internet. Mithilfe von symmetrischer und asymmetrischer Verschlüsselung schützt es Kommunikationspartner vor dem Abhören von Daten auf dem Transportweg. Durch eine zertifikatsbasierte Vertrauensinfrastruktur kann außerdem die Identität von Kommunikationspartnern überprüft werden (Prüfung des SSL-Zertifikates)
S. 4

tar Engl. für **t**ape **a**rchive, ein Format zum Bündeln von Dateien und Ordnern in einer einzigen Datei
S. 17, 38, 39

UNIX-Socket

Eine standardisierte Einrichtung zur prozessübergreifenden Kommunikation auf einem UNIX bzw. GNU/Linux basierenden Betriebssystem. Sie wird in Form einer speziellen Datei realisiert, die dazu von den miteinander kommunizierenden Prozessen gelesen und beschrieben werden kann.

S. 19, 23, 24

URL

Engl. für Uniform Resource Locator. Beispiel für eine URL ist <http://www.open-ease.org>

S. 22

WebSocket

Technologie für die Kommunikation zwischen einem Webbrowser mittels JavaScript und einer Serveranwendung. Eine Besonderheit ist, dass der Server durch die durch den Webbrowser aufgebaute Verbindung ohne vorherige Aktion des Browsers Daten an diesen übermitteln kann.

S. 30, 32, 35, 41, 44–47

A.5 Details zur Root-Sicherheitslücke

Die folgenden Pythonbefehle ermöglichen es, in der Debug-Konsole von openEASE einen neuen SSH-Schlüssel für den Benutzer `moritz` auf dem (nachgestellten) Hostsystem zu hinterlegen und diesem den Zugriff auf `sudo` ohne Passworteingabe zu ermöglichen.

```
1 import docker
2 client = docker.Client(base_url='unix://var/run/docker.sock', version='
    1.12', timeout=10)
3 client.create_container("ubuntu", tty=True, volumes=['/hosthome'], name='
    attack', command='bash -c "echo ssh-rsa AAAAB3NzaC1yc2EAAAABJQAAAQEAm
    +2wgC6fsZMh0rjvV08WbA85wnBRLhZvUssiajLhwhCWcTJD/iLrUgyLDeTbqAdR6xiokL
    /Uq6t3wCRgPKEDz1NIIdnqa8dEPY/1E1AtWEmSK0Yg0j3Lr7518Jzs/
    KJmR3wIjQ12kuQX6VT60FiWVuUB6zLIZogeIZIs4iuzhJh/
    QkcB4p5Td3WRfLRCSRZToyGTPle87hELI+jwGzzw84oC8lXCBayWRjc1LZW+50n7GZyANv
    /q00bBhlgQMeAJ9bnFE00EMN2R2b/
    jM8A1Bu9H4ck8mw9mXGk2dTng0yQygfWCm38fyw9nnisP0xT9P9prqF3+e8616P8NKB+
    Bw== rsa-key-20150526 >> /hosthome/moritz/.ssh/authorized_keys &&
    chown 1000:1000 /hosthome/moritz/.ssh/authorized_keys && chmod 600 /
    hosthome/moritz/.ssh/authorized_keys"')
4 client.start("attack", binds={'/home': {'bind': '/hosthome'}})
5 client.remove_container("attack")
```

```
6 client.create_container("ubuntu", tty=True, volumes=['/hostetc'], name='
    attack', command='bash -c "echo moritz ALL = NOPASSWD: ALL >> /hostetc
    /sudoers"')
7 client.start("attack", binds={'/etc': {'bind': '/hostetc'}})
8 client.remove_container("attack")
```

In der ersten Zeile wird die Python-Programmbibliothek zur Steuerung von Docker importiert. Danach erstellt der Aufruf der soeben importierten Programmbibliothek für die Instanziierung eines neuen Docker-Clients, wobei die in *Webrob* eingebundene Docker-API referenziert wird. Zeile 3 und 4 sorgen für die Erstellung und Ausführung eines Containers, der den Ordner `/home` des Hosts nach `/hosthome` einbindet und darin einen vom Angreifer spezifizierten SSH-Schlüssel für den Benutzer *moritz* aktiviert. Nachdem der Container in Zeile 5 wieder entfernt wird, wird in Zeile 6 und 7 ein neuer Container angelegt, der den Ordner `/etc` des Hosts nach `/hostetc` einbindet und darin für den Benutzer *moritz* den Zugang zu *sudo* ohne Passwortheingabe ermöglicht.

Die Kombination aus beiden Zugängen (SSH und *sudo*) ermöglichen einem Angreifer, sich über SSH auf dem Server anzumelden und über *sudo* administrative Rechte zu erlangen.

A.6 Inhalt des Datenträgers

- `thesis.pdf` - Dieses Dokument
- `openEASE.zip` - Eine Kopie der openEASE Projekt-Versionsverwaltung mit den Umsetzungen aus dem Sicherheitskonzept, Abgerufen am 02. Juni 2015
- `dependencies.zip` - Eine Kopie von Projekt-Versionsverwaltungen der Abhängigkeiten von openEASE (`knowrob`, `knowrob_addons`, `knowrob_webtools`, `nginx_proxy`, `docker-gen`), Abgerufen am 02. Juni 2015
- `pentest` - Ordner mit Screenshots der Durchführung von Penetrationstests