Security von Import-Diensten im Docker-Umfeld

Thomas Jäger

Department Informatik
Hochschule für Angewandte Wissenschaften Hamburg
Berliner Tor 7
20099 Hamburg
thomas.jaeger@haw-hamburg.de

Abstract. Der Import und die anschließende Verarbeitung von Daten sind zentrale Themen der heutigen Informatik und Softwareentwicklung. Die Verwendung von Daten im XML-Format ist seit langem in verschiedensten Systemen zum Austausch von Daten über das Internet oder Netzwerke etabliert. Immer öfter wird bei der Entwicklung solcher Systeme von klassischen monolithischen Softwarearchitekturen abgesehen. Stattdessen werden verteilte Microservice-Architekturen eingesetzt. Gründe hierfür sind Vorteile wie die verbesserte Wartbarkeit, Austauschbarkeit und Erweiterbarkeit des Systems. Im wachsenden Big Data Umfeld findet dieses Architekturmodell aufgrund der möglichen und benötigten Skalierbarkeit ebenfalls hohen Zuspruch. Das prominenteste Beispiel einer Implementierungsunterstützung ist die Container-Technologie Docker. Diese Kombination eines etablierten Daten-Formats mit neuen Softwarearchitekturen und Technologien stellt neue Fragen und Anforderungen an die Security. Diese Arbeit soll Motivation zur Überprüfung der Auswirkungen von XML-Angriffen auf eine typische Docker-Umgebung sein und konzentriert sich dabei auf zwei mögliche, mithilfe von XML-Nachrichten durchgeführte Angriffsformen: XML-Floodings und XML-Injections. Dazu werden mögliche Variationen und Risiken dieser Angriffe gezeigt und eine Vorgehensweise der Untersuchung skizziert.

1 Einleitung

Die Verwendung der Extensible Markup Language (XML) [RBG02] zum Transportieren von Daten über das Internet oder Netzwerke ist in heutigen Systemen weit verbreitet. Dazu zählen Unternehmensanwendungen wie Enterprise-Resource-Planning-Systeme (ERP), wo der Business-to-Business (B2B) Austausch von Daten über XML-Nachrichten läuft, aber auch öffentliche oder private APIs, die Daten im XML-Format zur Verfügung stellen. Auch Ansätze wie das Semantic Web nutzen ein erweitertes XML-Format. [Pok10] [ST07]
Nicht zuletzt aufgrund der Herausforderung, einer stetig wachsende Menge an zu verarbeitenden Daten zu begegnen, werden die traditionellen monolithischen Architekturmodelle in der Softwareentwicklung immer häufiger von verteilten, microservice-basierten Modellen abgelöst [Map16]. Dadurch werden

auch weitere Vorteile wie die verbesserte Wartbarkeit und Austauschbarkeit von Komponenten und die Erweiterbarkeit des Systems angestrebt. So profitieren verteilte, agentenbasierte Anwendungen, bspw. das MARS-Projekt der HAW Hamburg [HATC+16], bereits von vielen dieser Vorteile. Da sich Micoservice-Architekturen ebenfalls gut für Geräte des Internet of Things (IoT) eignen, ist eine weitere Verbreitung vorauszusehen [Tay16]. Mithilfe der Container-Technologie Docker [Doc17a] wird die Implementierung solcher Systeme auch für eine breite Masse an Entwicklern und Unternehmen zugänglich. So verbreiteten sich Docker-Microservice-Architekturen in den letzten Jahren schnell [New16] [Map16]. Spätestens mit dem Einsatz von Docker bei Spotify und Netflix, hat sich die Technologie auch in großen Unternehmen etabliert [Van15] [Mar15].
Die bereits erwähnte wachsende Menge an Daten, bekannt unter dem Begriff Big Data, schließt unstrukturierte, semi-strukturierte und strukturierte Datenformen ein. Daten im XML-Format zählen zu den semi-strukturierten Daten und haben demnach auch im Big Data Bereich Relevanz [Jan15]. Aufgrund dieser Datenvielfalt, werden zur Speicherung von Daten in Big Data Systemen meist nicht-relationale Datenbanken wie eine MongoDB [HK16] verwendet [MSFM16].
Durch diese neu eingesetzten Technologien und Architekturen, kommen neue Fragen und Anforderungen in der Security auf - bspw. im bereits erwähnten MARS-Projekt, zu welchem in einer Vorarbeit ein Überblick über mögliche Sicherheitslücken geschaffen wurde [Jä16]. Hierbei geht es um neue Angriffsformen, die erst durch Technologien wie Big Data, nicht-relationale Datenbanken und container-basierte Entwicklungen möglich werden und um Abwandlungen altbekannter Angriffe, deren Relevanz in Bezug auf das Verhalten dieser neuen Umgebung noch geprüft werden muss.
Diese Arbeit soll als Motivation für Untersuchungen von XML-Security Fragen im Umfeld von Docker, Web Service APIs (bspw. REST) und nicht-relationalen Datenbanken dienen. Im Fokus stehen zwei mögliche Angriffsformen: XML-Flooding und XML-Injections. Diese werden anhand von bereits bestehender Literatur in verschiedenen Variationen und in Bezug auf das neue Umfeld erläutert und deren Relevanz, mögliche Risiken und zu untersuchende Aspekte aufgezeigt. Im Anschluss wird ein möglicher Aufbau einer Versuchsumgebung skizziert und Methoden und Vorgehensweisen zur Durchführung der Untersuchung gezeigt. Risiken bei der Durchführung dieses Untersuchungsprojektes werden erläutert, bevor mit einem Fazit und einem Ausblick auf zukünftige Arbeiten abgeschlossen wird.

2 XML-Flooding (XDoS)

Ein XDoS-Angriff ist eine mithilfe von XML-Nachrichten durchgeführte Denial of Service Attacke (DoS). Dabei ist das Ziel, alle Resourcen eines Zielsystems mit einer Flut von Anfragen, in diesem Fall das Senden von XML-Nachrichten, zu verbrauchen oder zu blockieren und es damit an der Weiterarbeit zu hindern [MH16] [CXZB11]. Der aktuelle "Web Application Attack Report (waar)" [Imp15] von Imperva zeigt eine stetige Zunahme von Flooding-Angriffen, wozu

auch XDoS-Angriffe zählen.
In diesem Kapitel werden unterschiedliche Varianten von XDoS-Angriffen auf Basis der im Abschnitt 2.1 vorgestellten Quellen gezeigt. Gleichzeitig wird auf Risiken eingegangen, die speziell im Bezug auf das Docker-Umfeld untersucht werden müssen.

2.1 Related Work

Im Themengebiet des XML-Floodings gibt bereits einige relevante Arbeiten zu begutachten. So haben Abhinav Nath Gupta und Dr. P. Santhi Thilagam vom Department of Computer Science & Engineering, NITK Surathkal, Indien, im Jahr 2015 eine Übersicht über mögliche Angriffe mithilfe von XML-Nachrichten mit dem Titel "Attacks on Web Services need to secure XML on Web" [NGT13] verfasst. Ähnlich verhält es sich mit der etwas älteren Studie "Possible attacks on XML Web Services" [MH16] von Esmiralda Moradian und Anne Hakansson vom Department of Information Science an der Universität von Uppsala, Schweden, aus dem Jahr 2006. Beide Arbeiten zeigen neben anderen XML-Angriffsformen einige mögliche Ausprägungen von XDoS-Angriffen.
Spezialisiert auf XDoS und DoS Angriffe hat sich dagegen die Arbeit "Cloud security defence to protect cloud computing against HTTP-DoS and XML-DoS attacks" [CXZB11] von Ashley Chonka, Yang Xiang, Wanlei Zhou und Alessio Bonti. Der verteilte Charakter der hier beschriebenen Systeme macht diese Arbeit im Zusammenhang mit Docker besonders interessant.
Den Fokus auf verteilte Angriffsformen des DoS-Angriffs setzen die Veröffentlichungen "Cloud-based DDoS Attacks and Defences" [DOC13] von Marwan Darwish et al und "Enhancing Big Data Security with Collaborative Intrusion Detection" [TNH+14] von Zhiyuan Tan et al. Letzteres stellt dabei zusätzlich Bezug zum Big Data Themengebiet her. In den Abschnitten 2.2 - 2.5 werden die hier erwähnen Quellen weiterverwendet.

2.2 Oversized Payload

Die XDoS-Angriffsform des Oversized Payloads, auch als XML-Bomb bezeichnet, wird mithilfe einer einzelnen XML-Nachricht mit Übergröße durchgeführt. Dabei wird eine unter Umständen mehrere Gigabyte große XML-Nachricht bspw. über eine Web Service API importiert. Anschließend wird diese geparsed und in ein Datenobjekt gespeichert. Dadurch wird ein Overload des Parsers provoziert, um einen Ausfall dessen herbeizuführen [MH16] [NGT13].
In einem Docker-Umfeld mit Containern, die XML-Import-Webservices enthalten, könnte das verschiedene Auswirkungen haben. Zum einen stellt sich die Frage nach dem Verhalten des Containers, der den Web Service mit dem XML-Parser enthält. Das Objekt könnte beim Parsen Speicherplatz benötigen, der die festgelegte Größe des Containers übersteigt. Ist die Größe nicht festgelegt, könnte der Container sich so weit ausdehnen, dass er keinen Platz für andere Container, bspw. den Datenbankcontainer, mehr lässt. Ist für den XML-Import-Container ein festes Maximum an den Ressourcen CPU und Arbeitsspeicher

festgelegt [doc17c], könnte diese Grenze erreicht werden, was zum Ausfall des Dienstes zur Folge hat. Das Verhalten des Datenbankcontainers muss in diesem Fall betrachtet werden. Ist hier kein Limit festgelegt, könnte der Parser die gesamten weiteren Ressourcen wie CPU und Hauptspeicher des physischen Host-Systems benötigen, was wiederum die restlichen Container lahmlegen könnte [Toz16].

2.3 Coercive Parsing

Beim Coercive Parsing wird versucht, die CPU eines Hostsystems, auf welchem ein XML-Import-Dienst läuft, zu überlasten, indem in einer tief verschachtelten XML-Nachricht eine große Zahl an XML-Tags [1] geöffnet wird. Beim Parsen dieser Nachricht wird die Antwortzeit des Servers entscheidend reduziert, was in einen erfolgreichen XDoS-Angriff führt [CXZB11] [NGT13]. Für das Docker-Umfeld entstehen hier die selben Risiken wie beim Oversized Payload Angriff. Der Coercive Parsing Angriff kann mit einer übergroßen XML-Nachricht, wie im Oversized Payload Angriff, oder mit einem Multiple Files Flooding kombiniert werden.

2.4 Multiple Files Flooding

Beim Multiple Files Flooding handelt es sich um die klassische Form eines DoS-Angriffs wobei in diesem Fall statt einfacher Anfragen XML-Nachrichten benutzt werden, um einen Server auszuschalten. Das Ziel ist es, einen Fehler des Import-Services herbeizuführen, indem dieser bspw. in eine Endlosschleife getrieben wird. Dadurch werden sämtliche seiner Ressourcen blockiert, was dazu führt, dass der Service für andere unerreichbar ist [MH16].
Ein wesentlicher Unterschied zu den beiden zuvor genannten Angriffsformen ist hier, dass Multiple Files Flooding Attacken nicht nur auf den Parser des Import-Dienstes zielen. Da die Überlastung in diesem Fall vor allem durch die Menge an XML-Nachrichten verursacht werden soll, können im Docker-Umfeld zusätzliche Risiken entstehen. So ist es möglich, dass der Parser eine gewisse Zeit mit der Datenmenge zurecht kommt, diese an den Datenbank-Container weiterleitet und in der Datenbank ablegt. Zum einen könnte das dazu führen, dass der Speicherplatz des Datenbank-Containers, analog zu einem realen Datenbankserver, vollläuft [EAK13]. Diese Grenze kann entweder eine festgelegte Größe des Containers oder die physikalische Grenze des Host-Systems sein. Wird dieser Zustand erreicht, muss das Verhalten des Datenbankcontainers und der darauf laufenden Datenbank selbst geprüft werden. So könnte bspw. der Datenbestand auf der Datenbank in einen inkonsistenen Zustand geraten. Hat der Container keine festgelegte Speichergrenze, könnte er wiederum allen anderen Services den verfügbaren Speicherplatz auf dem Host-System stehlen. Zum anderen könnte der Speicherprozess der Datenbank selbst anderen Container benötigte Ressourcen wie CPU oder Hauptspeicher stehlen.

[1] XML-Tags sind Abgrenzer von XML-Elementen innerhalb einer XML-Nachricht [sel17]

2.5 Distributed Flooding (DXDoS)

Eine noch effektivere Form des XDoS-Angriffs stellt die Distributed XDoS-Attacke dar. Analog dazu gibt es den DDoS-Angriff zur DoS-Attacke [DOC13]. Dabei wird bspw. ein Multiple Files Angriff von vielen, verteilten Punkten auf ein Ziel durchgeführt. Das führt dazu, dass klassische Schutzmechanismen wie das serverseitige Begrenzen von Anfragen eines einzelnen Clients, ausgehebelt werden. Außerdem kann die Intensität eines Angriffs durch den Einsatz verteilter Angriffsagenten stark nach oben skaliert werden [CXZB11].
DDoS-Angriffe sind in der jüngsten Vergangenheit öffentlichkeitswirksam diskutiert worden, da einige großangelegte Angriffe auf bekannte Infrastrukturen erfolgreich durchgeführt wurden. Hier ist bspw. der Angriff auf das ebenfalls dockerbasierte Spotify zu nennen, bei welchem ein IoT-Botnetz die verteilten Angriffsagenten gestellt hat [Van15] [Bei16].
Eine besondere Form des DDoS-Angriffs ist in [TNH+14] beschrieben. Hier geht es um im Kontext von Big Data Ökosystemen, welche ebenfalls häufig mit der Docker-Technologie umgesetzt werden, zusätzlich entstandene Gefahren. So haben Big Data Ökosysteme meist mehrere Schnittstellen zur Außenwelt. So ist es nicht nur möglich, mithilfe einer Vielzahl an Agentensystemen eine DDoS-Attacke durchzuführen, sondern auch, parallel an mehreren Punkten anzugreifen. So können Schutzmechanismen wie Intrusion Detection Systeme an einzelnen Eingangspunkten keine Attacke erkennen, da erst die Summe der Angriffspakete eine erfolgreiche DDoS-Attacke herbeiführt.
Die erwarteten Risiken und möglichen Auswirkungen auf das Docker-System sind identisch zum Multiple Files Flooding. Eine Untersuchung der Gefahr eines solchen Angriffs mithilfe von XML-Nachrichten, könnte in einer zukünftigen Arbeit durchgeführt werden. Die hier vorliegende Arbeit beschäftigt sich lediglich mit einem möglichen Eintrittspunk der Angriffe.

3 XML-Injections

Auf Injections basierende Angriffe versuchen im Allgemeinen, schädliche Inhalte in Systeme einzuschleusen, um diese dort zur Ausführung zu bringen. Ziel ist es, dass Zielsystem zu manipulieren, Einsicht in Daten zu erlangen, sich Zugriff zu verschaffen oder Daten zu löschen. Durch das Importieren von XML-Nachrichten in Web Services, eröffnet sich ein möglicher Eingangspfad für Injections, die innerhalb dieser manipulierten Nachricht transportiert werden [NGT13]. In diesem Kapitel wird auf mögliche XML-Injections eingegangen, welche auf die Datenbank innerhalb eines Datenbankcontainers zielen. Mögliche Auswirkungen in der Docker-Umgebung werden aufgezeigt.

3.1 Related Work

Die bereits im Kapitel 2 verwendeten Übersichtsarbeiten von Abhinav Nath Gupta et al [NGT13] und Esmiralda Moradian et al [MH16] behandeln auch

mit XML-Nachrichten durchgeführte SQL-Injections. Matt Koch zeigt in seiner Arbeit "Web Application File Upload Vulnerabilities" [Koc15] zusätzlich, dass SQL-Injections innerhalb von Import-Nachrichten neben dem XML-Format auch im CSV-Format möglich sind.
Die Arbeit "No SQL, No Injection? Examining NoSQL Security" [RSB15] von Aviv Ron, Alexandra Shulman-Peleg und Emanuel Bronshtein beschäftigt sich mit Injections auf nicht-relationale Datenbanken, die nicht mit der Abfragesprache SQL arbeiten und zeigt, dass diese ebenfalls eine Gefahr darstellen können.
Eine Einordnung in das Big Data Umfeld gibt die Arbeit "Expanded Top Ten Big Data Security and Privacy Challenges" von der Cloud Security Alliance Big Data Working Group und betont dabei, dass hier auch NoSQL-Injections von Bedeutung sein können. Zum einen weil im Big Data Umfeld bevorzugt nicht-relationale Datenbanken eingesetzt werden und zum anderen, weil NoSQL-Injections immer noch nicht abschließend behandelt wurden und deswegen standardisierte Lösungsansätze vermisst werden [Gro13].

3.2 SQL-Injection

SQL-Injections sind nach wie vor eine Gefahr für viele Webanwendungen, was der Bericht des Ponemon Instituts des Jahres 2015 zeigt [Ins16]. Die Arbeiten von Abihinav Nath Gupta et al [NGT13], Esmiralda Moradian et al [MH16] und Matt Koch [Koc15] zeigen, dass SQL-Injections auch mithilfe von XML-Nachrichten in Systeme eingeschleust und zur Ausführung gebracht werden können. Allerdings gibt es bereits viele ausführlich erprobte Methoden, um SQL-Injections vorzubeugen. Beispielsweise zu nennen sind sogenannte Prepared Statements, mit deren Hilfe Sonderzeichen, die auf ausführbaren SQL-Code hinweisen, "escaped" werden können [AFS11]. Auch in der Arbeit von Abihinav Nath Gupta et al [NGT13] werden bereits einige Lösungsvorschläge beschrieben. Bei dem großen neuen Einsatzgebiet von Docker, dem Big Data Umfeld, werden relationale SQL-Datenbanken aufgrund der heterogenen Datenbeschaffenheit nicht bevorzugt verwendet [MSFM16]. Da auch kein spezielles Verhalten innerhalb der Docker-Umgebung zu erwarten ist, wird diese Art der Datenbank-Injection in der vorliegenden Arbeit nicht weiter behandelt. Die Relevanz und Häufigkeit von SQL-Injections könnte sich allerdings langfristig auf die im folgenden Abschnitt behandelten NoSQL-Injections übertragen.

3.3 NoSQL-Injection

NoSQL-Injections funktionieren prinzipiell wie die ihnen verwandten SQL-Injections, zielen aber auf nicht-relationale Datenbanken ohne SQL-Abfragesprache ab. Die Arbeit von Aviv Ron et al [RSB15] hat gezeigt, dass NoSQL-Injections über APIs auf Datenbanken wie bspw. eine MongoDB möglich sind. In Kombination mit der bereits gezeigten Tatsache, dass SQL-Injections mithilfe von XML-Nachrichten möglich sind, gilt es zu prüfen, ob dies auch

für NoSQL-Injections gilt. Auch sind NoSQL-Injections im Kontext des Docker-Umfeldes relevanter als die oben aufgeführten SQL-Injections. Dabei hängen die Risiken einer NoSQL-Injection weniger mit der Docker-Technologie selbst zusammen als mit den im Big Data und Docker-Umfeld häufig verwendeten nicht-relationalen Datenbanken. Desweiteren gibt es hierzu bislang kaum erprobte, generelle Lösungsansätze. [Gro13] [MSFM16]. In Kapitel 4.4 wird auf eine mögliche Untersuchungsmöglichkeit und in Kapitel 7 auf mögliche Folgearbeiten zur Begegnung der Risiken eingegangen.

4 Versuchsausführung

In diesem Kapitel wird skizziert, wie die im bisherigen Teil der Arbeit beschriebenen Annahmen und Gefahren in einem Versuchsaufbau getestet, gemessen und Ergebnisse erzielt werden können. Dazu werden zunächst Versuchsziele formuliert, der mögliche Aufbau der Versuchsumgebung beschrieben und konkrete Vorgehensweisen zur Versuchsdurchführung aufgezeigt.

4.1 Ziele

Das Ziel dieser praktischen Versuche ist es, die zuvor gezeigten möglichen XML-Angriffsformen in einer typischen Docker-Umgebung nachzustellen und auszuführen. Da diverse Arbeiten die Möglichkeit von XDoS-Angriffen bereits gezeigt haben, geht es in dieser Angriffskategorie vordergründig darum, die konkreten Auswirkungen auf das Docker-Umfeld und dessen Verhalten zu ergründen, zu messen und zu dokumentieren. Dabei sind insbesondere die bereits in Kapitel XML-Flooding (XDoS) genannten Aspekte zu untersuchen. Es sollen mit Messungen belegte Aussagen über das Verhalten der Docker-Umgebung im Angriffsfall entstehen.
Da im Falle der SQL-Injections bereits ausreichend Lösungsansätze entwickelt wurden und keine besonderen Auswirkungen innerhalb des Docker-Umfeldes zu erwarten sind, bspw. der Prepared Statements Ansatz [AFS11], der auf Code-Ebene umgesetzt wird und damit unabhängig vom Einsatz der Docker-Technologie funktioniert (siehe Kapitel SQL-Injection), wird hier keine Untersuchung vorgenommen. So geht es im zweiten Teil darum zu zeigen, ob in der beschriebenen Testumgebung und unter Verwendung von XML-Nachrichten, NoSQL-Injections möglich sind. Ziel ist es dabei nicht, eine Aussage über das gesamte Spektrum an möglichen NoSQL-Injections treffen zu können, sondern exemplarisch zu zeigen, ob eine NoSQL-Injection auf die eingesetzte nicht-relationale Datenbank (siehe Kapitel Umgebung) durchgeführt werden kann.

4.2 Umgebung

Die Versuchsumgebung soll repräsentativ für eine weit verbreitete und verwendete Form der Docker-Umgebung sein. Dazu werden zwei Docker-Container

eingerichtet (siehe Abbildung 1). Der erste Container beinhaltet den Import-Service, welcher für das Empfangen und Verarbeiten der XML-Nachrichten zuständig ist. Näheres zum Import-Service im Abschnitt *Import-Service*. Der zweite Container beherbergt eine Datenbank. Hier wurde eine MongoDB [HK16] gewählt, um eine mögliche Angriffsfläche für NoSQL-Injections zu bieten (siehe Kapitel 3.3) und sich damit nahe am Big Data Umfeld zu bewegen. Dabei hat der Datenbank-Container lediglich die Aufgabe, vom Import-Service empfangene Daten in der MongoDB abzulegen.
Orchestriert werden die beiden Container über das Tool Docker Compose [Doc17b]. Dieses ist in der Lage, die Kommunikationsschnittstellen und das Hoch-und Herunterfahren der Container zu koordinieren.
Zusätzlich zu den erwähnten Komponenten muss eine Möglichkeit des Monitorings der Docker-Container zur Verfügung stehen. Um bei den späteren Versuchsdurchführungen, insbesondere der XDoS-Angriffe, Metriken wie die Auslastung der CPU, des Arbeitsspeichers und des Speicherplatzes zu messen und zu dokumentieren, könnte ein Monitoring-Tool wie das auf der DockerCon 2016 [Doc16] vorgestellte Datadog [Dat17] eingesetzt werden. Hierzu wäre ein zusätzlicher Container notwendig, der in die Docker-Umgebung eingebunden wird.

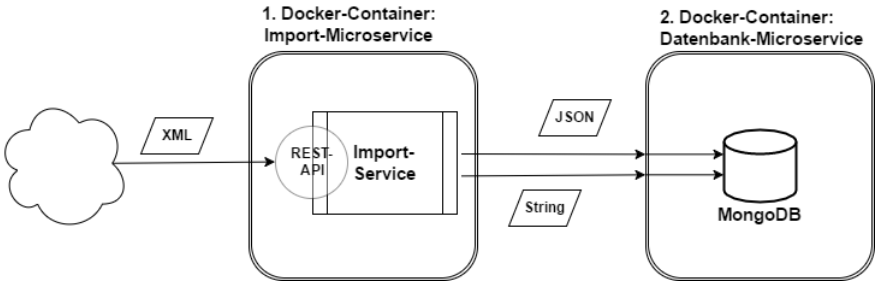

Fig. 1. *Versuchsumgebung*

Import-Service Der Import-Service im ersten Docker-Container (siehe Abbildung 1) soll in der Programmiersprache Java entwickelt werden. Java eignet sich hier besonders, weil die Docker-Technologie gerade hier sehr verbreitet ist [New16] und Java im Allgemeinen die derzeit populärste Programmiersprache darstellt (Stand Januar 2017 [Jav17]).
Die Anwendung soll eine REST-Schnittstelle anbieten, über die externe Benutzer XML-Nachrichten importieren können. Da sich die Versuche auf die zuvor beschriebenen XML-Angriffe konzentrieren sollen, wird an dieser Stelle von Sicherheitsmechanismen wie bspw. Webservice-Passwörtern für den Zugang zur REST-Schnittstelle abgesehen. Um Angriffe wie das Coercive Parsing simulieren zu können, ist es notwendig, die empfangenen XML-Nachrichten in der Anwendung zu parsen. Dafür soll sie die eintreffenden XML-Nachrichten in

JSON-Objekte umwandeln und diese anschließend an den Datenbank-Container senden, um sie in der MongoDB als MongoDB typisches Binary JSON-Objekt (BSON) abzulegen [mon]. Um zusätzliche Datenvariation simulieren zu können, soll die Anwendung außerdem in der Lage sein, die XML-Nachrichten in Strings umzuwandeln und diese ebenfalls in der MongoDB zu speichern. Desweiteren kann diese toString-Methode später variiert werden, um für die Versuche zur NoSQL-Injection Angriffsflächen zu bieten. Das wäre bspw. möglich, indem lediglich einzelne Werte aus der XML-Nachricht gelesen und als String an den Datenbank-Container gesendet werden.
Auf eine präzise Schema-Prüfung der XML-Nachrichten wird zu Versuchszwecken ebenfalls verzichtet, da insbesondere nicht-relationale Datenbanken und das Big Data Umfeld allgemein mit heterogenen Daten umgehen müssen [MSFM16].

4.3 Vorgehen

XDoS Um die verschiedenen, im Kapitel 2 aufgeführten Varianten des XDoS-Angriffs zu simulieren, ist es zunächst notwendig, Testdaten herzustellen. Im Fall des Oversized Payloads können Angriffsnachrichten wie in [NGT13] beschrieben und auf Abbildung 2 zu sehen, manuell erstellt werden, indem einzelne Tags x-fach multipliziert werden. Unterstützung kann hier bspw. das Tool SoapUI [Soa17] bieten, das die Durchführung von XML-Bomb-Angriffen als Security-Test unterstützt. Die Angriffsnachricht des Coercive Parsing kann ebenfalls wie in [NGT13] erläutert, durch das Öffnen x-facher Tags innerhalb der XML-Nachricht, händisch erzeugt werden.
Um XML-Flooding mit einer gleichen, immer wieder gesendeten XML-Nachricht durchzuführen, sollte ein Lasttest-Tool wie Apache JMeter [JMe17] verwendet werden. Damit können Testpläne erstellt werden, die automatisch ausführbar sind und damit eine regulierbare Anzahl von XML-Nachrichten an die REST-Schnittstelle der Anwendung senden. Auch die hier verwendete XML-Nachricht kann manuell angelegt werden.
Um unterschiedliche, heterogene Daten für das XML-Flooding anzulegen, müssen jedoch Hilfsmittel verwendet werden. Dabei könnte ein Tool wie XML-Mate hinzugezogen werden, mit dem eine große Anzahl von XML-Nachrichten mithilfe einer XML Schema Datei (XSD) und exemplarischen XML-Nachrichten generiert werden kann [HHGZ14]. Das Versenden der Anfragen erfolgt dabei wieder über ein Lasttest-Tool.

4.4 XML-Injection

In diesem Teil des Versuchs wird geprüft, ob die Möglichkeit besteht, NoSQL-Injections mithilfe von XML-Nachrichten durchzuführen. Dazu muss die XML-Nachricht manuell erstellt werden. Diese wird auf Basis einer Kombination der Erkenntnisse aus "No SQL, No Injection?" von Aviv Ron et al [RSB15] und "Automated and Effective Testing of Web Services for XML Injection Attacks" von Sadeeq Jan et al [JNB16] gebaut. In der ersten dieser Arbeiten werden

```
<Address>
<name>ABC</name>
<name>ABC</name>
<name>ABC</name>
....to 10000
</Address>
```

Fig. 2. *XML-Bomb, Quelle: Nath Gupta et al: "Attacks on Web Services need to secure XML on Web" [NGT13]*

mögliche NoSQL-Injections auf MongoDB Datenbanken gezeigt. Die zweite verwendet XML-Nachrichten, um klassische SQL-Injections durchzuführen. Ziel ist es dabei, exemplarisch die Auswirkungen einer erfolgreich durchgeführten NoSQL-Injection zu zeigen und zu dokumentieren.

5 Risiken

Mögliche Risiken beim Durchführen der Untersuchungen entstehen an unterschiedlichen Stellen. Der NoSQL-Injection Versuch beherbergt das Risiko, dass die Durchführung des Angriffs via XML-Nachricht nicht erfolgreich gelingt. In einem solchen Fall wäre dennoch nicht bewiesen, dass eine NoSQL-Injection in dieser Form unmöglich ist. Somit würde das Ergebnis dieses Versuchs an Wert verlieren. Das Risiko wird durch mangelnde Unterstützung an Tools zur Durchführung von NoSQL-Injections im Allgemeinen verstärkt.
Bei den XDoS-Versuchen zeigen sich Risiken vor allem in der Qualität der Ergebnisse. So könnten die eingesetzten Monitoring-Tools Effekte, die durch die Angriffe verursacht werden, nur ungenügend darstellen und eine Auswertung schwierig machen. Auch könnte ein Erkenntnisgewinn ausbleiben, sollte es nicht gelingen, XDoS-Angriffe nach dem entwickelten Konzept und mit den eingesetzten Tools durchzuführen oder falls die Docker-Technologie und die REST-Schnittstelle bereits ausreichende Gegenmaßnahmen beinhalten. In diesem Fall könnte man sich dennoch auf das Zeigen der Widerstandsfähigkeit der Docker-Komponenten gegen Angriffe dieser Art konzentrieren.
Technische Risiken könnten beim Durchführen von XDoS-Angriffen ebenfalls eine Rolle spielen. Insbesondere das XML-Flooding könnte angreiferseitig höhere Rechenkapazitäten als vorhanden erfordern. Das führt möglicherweise dazu, dass die Angriffe nicht die notwendige Last erzeugen, um erfolgreich zu sein. Dementsprechend könnten gewünschte zu beobachtende Effekte ausbleiben.

6 Fazit

In dieser Arbeit wurde die Relevanz einer Untersuchung von Import-Security im Docker-Umfeld gezeigt. Der Fokus wurde aufgrund der weiten Verbreitung

(siehe Einleitung) auf das XML-Nachrichtenformat gelegt.
Darüber hinaus wurden die beiden Kategorien XDoS-Angriffe (Kapitel 2) und XML-Injections (Kapitel 3) aus dem Umfeld der XML-Angriffe gewählt. Mögliche Angriffsformen dieser Kategorien wurden anhand von erwähnter Literatur erläutert und mögliche Risiken und Auswirkungen auf das Docker-Umfeld gezeigt. Dabei wurde im Falle der XDoS-Angriffe die Notwendigkeit einer konkreten Versuchsdurchführung im Docker-Umfeld deutlich. Eine Untersuchung des Sonderfalls der Distributed XDoS-Attacken wurde ebenfalls erwähnt, jedoch nicht im weiteren Verlauf der Arbeit behandelt, sondern als mögliche, wichtige Folgearbeit dargestellt.
Bei den XML-Injections ist die Untersuchung einer Kombination aus NoSQL-Injections [RSB15] mithilfe von XML-Nachrichten [JNB16] wichtig, da diese nicht durch Literatur abgedeckt wird. Dagegen wurden SQL-Injections im Allgemeinen und speziell mithilfe von XML-Nachrichten bereits zur Genüge behandelt und entsprechende Gegenmaßnahmen entwickelt, sodass keine weitere Untersuchung im Docker-Umfeld notwendig ist.
Im Anschluss wurde ein möglicher Versuchsaufbau inklusive eingesetzter Komponenten und Technologien skizziert (Kapitel 4.2). Auch wurden mögliche Herangehensweisen zur konkreten Testdatenerstellung, zur Durchführung der Angriffe und zum Monitoring der Docker-Umgebung, um Versuchsergebnisse sichtbar zu machen, aufgezeigt. Einige Tools die hier zur Hilfe herangezogen werden könnten wurden erwähnt.
Dem Aufzeigen von möglichen Risiken bei der Ausführung des Versuchs (Kapitel 5) folgt ein Fazit und ein Ausblick über mögliche resultierende Folgeprojekte.

7 Ausblick

Nach Durchführung der in dieser Arbeit beschriebenen Versuche und in Abhängigkeit der daraus resultierenden Ergebnisse, gibt es einige Möglichkeiten der Weiterarbeit und Vertiefung in der Thematik. Zum einen wäre bei erfolgreich durchgeführten Angriffen und entsprechenden Ergebnissen die Beschäftigung mit möglichen Abwehrmaßnahmen erforderlich. Dazu könnten die Möglichkeiten innerhalb der Docker-Technologie geprüft werden oder Lastverteilungsmechanismen, bspw. mit Netflix Zuul [Lar15], einbezogen werden. Darüber hinaus könnte eine Auseinandersetzung mit Intrusion Detection Systemen im Docker und Big Data Zusammenhang ein weiterer Weg sein, sich mit der Absicherung zu beschäftigen [TNH+14] [Jan15].
Des Weiteren könnten zusätzliche Untersuchungen möglicher Angriffen auf Import-Services im Docker-Umfeld durchgeführt werden. Beispielsweise eine Erweiterung auf Formate wie das CSV- oder JSON-Nachrichtenformat aber auch die Beschäftigung mit den in Kapitel 2.5 erwähnten Distributed XDoS-Angriffen. Diese könnten bspw. mithilfe eines eigenen Docker-Angriffsclusters, welches dazu aufgebaut werden müsste, durchgeführt werden. Hier wäre auch eine Erweiterung der Versuchsumgebung um multiple Eintrittspunkte möglich, um die im Kapitel 2.5 beschriebene Sonderform der DXDoS-Angriffe simulieren zu können. Dabei

gewonnene Erfahrungen aus der möglichen Simulation von DXDoS-Angriffen könnten auch zur Konzeption und Entwicklung eines DXDoS-Lasttesttools verwendet werden, da diese Form der XDoS-Angriffe toolseitig noch nicht abgedeckt ist.
Möglich wäre auch eine Folgearbeit, die den Risiken der in Kapitel 4.4 gezeigten und durchgeführten NoSQL-Injections begegnet. Hier könnte eine Möglichkeit gesucht werden, die Gefahr durch NoSQL-Injections in der beschriebenen Umgebung zu verkleinern oder zu verhindern. Als Ansatz könnten bspw. die in Kapitel 3.2 aufgeführten Maßnahmen gegen SQL-Injections dienen, indem deren Übertragbarkeit auf NoSQL-Injections untersucht wird.
Die aus der Durchführung der Versuche gewonnenen Erkenntnisse und Ergebnisse könnten in einer Folgearbeit auch an einer realen Anwendung erprobt werden. Anbieten würde sich hier bspw. das in der Einleitung bereits erwähnte MARS-Projekt der HAW Hamburg, welches die Docker-Technologie und entsprechende Import-Dienste verwendet [HATC+16] und in einer Vorarbeit bereits im Hinblick auf mögliche Sicherheitslücken betrachtet wurde [Jä16].

References

[AFS11] Aharon Abadi, Yishai A. Feldman, and Mati Shomrat. Code-motion for API Migration: Fixing SQL Injection Vulnerabilities in Java. In *Proceedings of the 4th Workshop on Refactoring Tools*, WRT '11, pages 1–7, New York, NY, USA, 2011. ACM.

[Bei16] Stefan Beirsmann. Störungen bei Spotify und Twitter: IoT-Botnet für massiven DDoS-Angriff benutzt, 2016.

[CXZB11] Ashley Chonka, Yang Xiang, Wanlei Zhou, and Alessio Bonti. Cloud Security Defence to Protect Cloud Computing Against HTTP-DoS and XML-DoS Attacks. *J. Netw. Comput. Appl.*, 34(4):1097–1107, July 2011.

[Dat17] Datadog, 2017. eingesehen: 08.02.2017.

[DOC13] M. Darwish, A. Ouda, and L. F. Capretz. Cloud-based DDoS attacks and defenses. In *International Conference on Information Society (i-Society 2013)*, pages 67–71, June 2013.

[Doc16] DockerCon 2016, 2016. eingesehen: 08.02.2017.

[Doc17a] Docker, 2017. eingesehen: 04.02.2017.

[Doc17b] Docker Compose, 2017. eingesehen: 05.02.2017.

[doc17c] docker.com. Limit a container's resources, 2017. eingesehen: 03.03.2017.

[EAK13] A. A. A. El-Aziz and A. Kannan. A survey on XML security. In *Recent Trends in Information Technology (ICRTIT), 2013 International Conference on*, pages 638–642, July 2013.

[Gro13] Cloud Security Alliance: Big Data Working Group. Expanded Top Ten Big Data Security and Privacy Challenges. 2013.

[HATC+16] Christian Hüning, Mitja Adebahr, Thomas Thiel-Clemen, Jan Dalski, Ulfia Lenfers, and Lukas Grundmann. Modeling Simulation As a Service with the Massive Multi-agent System MARS. In *Proceedings of the Agent-Directed Simulation Symposium*, ADS '16, pages 1:1–1:8, San Diego, CA, USA, 2016. Society for Computer Simulation International.

[HHGZ14] Nikolas Havrikov, Matthias Höschele, Juan Pablo Galeotti, and Andreas Zeller. XMLMate: Evolutionary XML Test Generation. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 719–722, New York, NY, USA, 2014. ACM.

[HK16] Hitesh Hasija and Dinesh Kumar. Compression & Security in MongoDB Without Affecting Efficiency. In *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies*, ICTCS '16, pages 96:1–96:6, New York, NY, USA, 2016. ACM.

[Imp15] Imperva. 2015 Web Application Attack Report (WAAR), 2015. eingesehen: 28.07.2016.

[Ins16] Ponemon Institut:. 2015 State of the Endpoint Report: User-Centric Risk, 2016. eingesehen: 28.07.2016.

[Jä16] Thomas Jäger. Security im MARS-Projekt. Hamburg, Germany, 2016. HAW Hamburg.

[Jan15] Amne Deepa.A. Jangla, Gurpreet Kaur. Development of an Intrusion Detection System based on Big Data for Detecting Unknown Attacks. 2015.

[Jav17] Java Popularität, 2017. eingesehen: 08.02.2017.

[JMe17] Apache JMeter, 2017. eingesehen: 09.02.2017.

[JNB16] Sadeeq Jan, Cu D. Nguyen, and Lionel C. Briand. Automated and Effective Testing of Web Services for XML Injection Attacks. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ISSTA 2016, pages 12–23, New York, NY, USA, 2016. ACM.

[Koc15] Matt Koch. Web Application File Upload Vulnerabilities. In *InfoSec Reading Room*. The SANS Institute, 2015.

[Lar15] Magnus Larsson. Building microservices with Spring Cloud and Netflix OSS, part 1, 2015. eingesehen: 18.08.2016.

[Map16] Simon Maple. Java Tools and Technologies Landscape Report 2016 , 2016.

[Mar15] Dianne Marsh. Netflix at DockerCon, 2015.

[MH16] Esmeralda Moradian and Anne Hakansson. Possible Attacks on XML Web Services. 2016.

[mon] MongoDB Architecture Guide.

[MSFM16] Julio Moreno, Manuel A. Serrano, and Eduardo Fernandez-Medina. Main Issues in Big Data Security. *Future Internet*, 8(3):44, 2016.

[New16] Business Cloud News. Exponential Docker usage shows container popularity, 2016.

[NGT13] Abhinav Nath Gupta and Dr. P. Santhi Thilagam. ATTACKS ON WEB SERVICES NEED TO SECURE XML ON WEB. 2013.

[Pok10] Jaroslav Pokorný. *XML in Enterprise Systems: Its Roles and Benefits*, pages 128–139. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[RBG02] Nicholas Routledge, Linda Bird, and Andrew Goodchild. UML and XML Schema. *Aust. Comput. Sci. Commun.*, 24(2):157–166, January 2002.

[RSB15] Aviv Ron, Alexandra Shulman-Peleg, and Emanuel Bronshtein. No SQL, No Injection? Examining NoSQL Security. *CoRR*, abs/1506.04082, 2015.

[sel17] selfhtml.org. XML/Regeln/Tags,Attribute, Wertzuweisungen und Kommentare, 2017. eingesehen: 03.03.2017.

[Soa17] SoapUI, 2017. eingesehen: 08.02.2017.

[ST07] Wei She and Bhavani Thuraisingham. Security for Enterprise Resource Planning Systems. *Inf. Sys. Sec.*, 16(3):152–163, May 2007.

[Tay16] Manu Tayal. IoT and Microservices Architecture, 2016.

[TNH+14] Z. Tan, U. T. Nagar, X. He, P. Nanda, R. P. Liu, S. Wang, and J. Hu. Enhancing Big Data Security with Collaborative Intrusion Detection. *IEEE Cloud Computing*, 1(3):27–33, Sept 2014.

[Toz16] Chris Tozzi. Docker Security – 6 Ways to Secure Your Docker Containers, 2016.

[Van15] Jonathan Vanian. How Spotify is ahead of the pack in using containers, 2015.