HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

MarkovShield

Man-in-the-Browser Detection

Bachelorarbeit

Abteilung Informatik

Hochschule für Technik Rapperswil

Frühjahrssemester 2017

Autor(en):	Matthias Gabriel, Philip Schmid
Betreuer:	Ivan Bütler
Experte:	Benjamin Fehrensen
Gegenleser:	Luc Bläser

Revision 1, Juni 2017
Bezugsquelle: https://eprints.hsr.ch

Hochschule für Technik Rapperswil
Oberseestrasse 10
8640 Rapperswil
Schweiz

[1] https://creativecommons.org/licenses/by-nc-sa/3.0/ch/legalcode.de

I.1. Abstract

Ein heutzutage weit verbreiteter Angriffsvektor auf Webapplikationen, im Besonderen E-Banking-Applikationen, stellen die Man-in-the-Browser Trojaner wie z.B. Gozi dar. Mittels der heutigen Schutzmechanismen ist es jedoch oft für den Betreiber der Webapplikation sehr schwierig oder gar unmöglich, eine solche Bedrohung erkennen zu können. Die entwickelte Security-Lösung MarkovShield setzt auf ein neuartiges Verfahren, welches die Requests auf eine Webapplikation in Echtzeit überprüft. Die Requests werden von einem Reverse Proxy erst nach einer positiven Analyse an die Webapplikation weitergeleitet. Die Analyse, welche mit dem Stream Processing Framework Apache Flink durchgeführt wird, detektiert Anomalien im aktuellen Benutzerverhalten, welche auf einen infizierten Computer schliessen lassen. Für die Analyse werden die Daten von vergangenen Sessions desselben Benutzers genutzt um ein Modell zu erstellen, welches das erwartete Benutzerverhalten möglichst akkurat widerspiegelt. Die erforderlichen Daten werden dabei durch das neu entwickelte Apache Modul mod_mshield erfasst und an Apache Kafka weitergeleitet, welches die verwendete Schnittstelle zu Apache Flink darstellt. Die implementierte Analyse setzt einerseits auf die Übergänge zwischen den aufgerufenen Seiten und andererseits auf die Anzahl der Aufrufe auf eine einzelne Webseite. Bei der Entwicklung der Lösung wurde grossen Wert auf die Erweiterbarkeit und Skalierbarkeit, sowohl der einzelnen Komponenten als auch der Gesamtlösung, gelegt.

I.2. Management Summary

Ausgangslage

Sicherheit ist in der IT eine Thematik, welche in den letzten Jahren immer wichtiger wurde. Insbesondere, aber nicht nur in der Finanzindustrie, sind die Investitionen in die IT-Sicherheit bedeutend verstärkt worden. Parallel zu den Investitionen der Firmen wächst allerdings auch die aufgewendete kriminelle Energie und es werden immer wieder neue Angriffsvektoren auf die Systeme und deren Benutzer entdeckt. Eine Attacke, welche auf solchen neuen Angriffsvektoren basiert, wird «Man-in-the-Browser»[2] genannt.

Kurzgefast wird dabei der Datenaustausch zwischen dem Browser des Benutzers und der Webapplikation direkt im Browser des Benutzers manipuliert. Dadurch können bestehende Sicherheitsmassnahmen wie die Verschlüsselung der Verbindung oder eine Erkennung von bösartigen Zugriffen anhand von computerabhängigen Merkmalen umgangen werden.

Die neu entwickelte Security-Lösung soll anhand von historischen und zugleich computerunabhängigen Session-Daten Anomalien detektieren und entsprechende Gegenmassnahmen treffen. Das Hauptziel ist dabei, dass die Bewertung der Zugriffe in Realtime geschieht und somit Gegenmassnahmen getroffen werden können, noch bevor der Zugriff überhaupt zur Webapplikation gelangt. Ein weiteres Ziel dieser Arbeit besteht darin, dass die zu entwickelnde Security-Lösung leicht in die bestehende Infrastruktur integriert werden kann und dass an der zu schützenden Webapplikation somit möglichst wenig angepasst werden muss.

Vorgehen und Technologien

Die Arbeit konnte auf den Erkenntnissen einer als geheim klassifizierten Masterarbeit aufbauen. In dieser Masterarbeit wurden verschiedene Algorithmen analysiert und zu jedem wurden entsprechende Empfehlungen abgegeben. Zudem wurde mit zwei erfolgversprechenden Algorithmen ein Post-Processing Prototyp erstellt. Basierend auf diesen Erkenntnissen, wurden in den ersten Wochen die zusätzlichen Anforderungen für eine Realtime-Lösung evaluiert und daraus eine entsprechende Architektur ausgearbeitet. Insbesondere wurden dabei darauf geachtet, dass die Abläufe und die Kommunikation der Komponenten aufeinander abgestimmt sein müssen. Es stellte eine besondere Herausforderung dar, Technologien zu finden, welche die Realtime Anforderung erfüllen und wie diese zu einer performanten Architektur zusammengesetzt werden können.

[2] (Gühring, 2006)

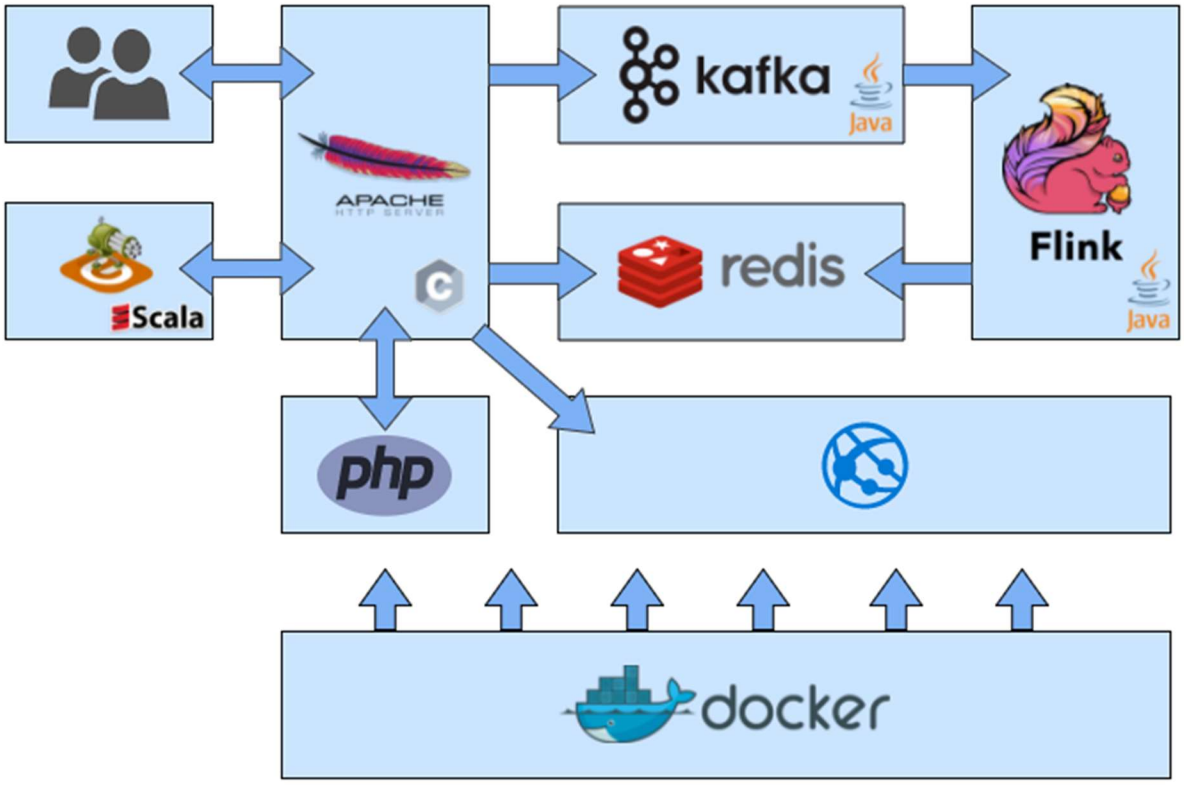


Abbildung 1 Architektur und Technologien

Gegen Ende der Arbeit wurden Performancetests durchgeführt um zu überprüfen, ob der gewählte Ansatz und die Architektur den Anforderungen genügen. Darüber hinaus konnte mit Hilfe dieser Tests eruiert werden, wo Performanceengpässe in der Architektur bestehen und in welchem Mass die User Experience von der Security-Lösung beeinflusst wird.

Die grössten Risiken der Arbeit bestanden darin, dass das Zusammenspiel der gewählten Architektur nicht wie angedacht funktioniert oder die Auswertung der Zugriffe zu viel Zeit beansprucht.

Ergebnisse

Als Resultat dieser Arbeit wurde die Security-Lösung MarkovShield entwickelt, welche in der Lage ist, in Realtime Zugriffe auf eine Webapplikation zu bewerten, bevor diese den Webserver erreichen. Wird ein Zugriff als verdächtig erkannt, so werden entsprechende Gegenmassnahmen getroffen. Zum Bespiel muss sich der Benutzer über einen zweiten Weg zusätzlich authentifizieren oder der Zugriff wird blockiert und der Benutzer entsprechend informiert.

Die Security Lösung wurde so aufgebaut, dass sie in Zukunft ohne grössere Umstände um ergänzende Auswertungsalgorithmen erweitert werden kann. Zugleich wurde bei der Auswahl der Technologien sichergestellt, dass beim Einsatz in einer produktiven Umgebung die einzelnen Komponenten gut skaliert werden können.

Um interessierten Personen einen Einblick in die Security-Lösung MarkovShield zu geben, wurde eine Demonstrations-Applikation erstellt, welche innert kurzer Zeit auf nahezu jedem System in Betrieb genommen werden kann.

Wie sich gegen Ende der Arbeit aufgrund der Performancetest-Resultate gezeigt hat, leistet die gewählte Architektur die geforderte Geschwindigkeit und Zuverlässigkeit, damit die kritischen Requests ausgewertet und grosse Verzögerung weitergeleitet werden können. Die Antwortzeiten liegen dabei in einer Zeit, welche keinen merkbaren Einfluss auf die User Experience hat.

Ausblick

Der Source Code der entwickelten Lösung wurde zum Projektende veröffentlicht. Dies ermöglicht es, dass Interessierte einen genaueren Einblick in die Lösung erhalten und den Code selbst verwenden und weiterentwickeln können.

Neben den bereits implementierten Funktionalitäten wurden diverse potentielle Erweiterungsmöglichkeiten erarbeitet und festgehalten. Seitens Entwickler besteht das Interesse, die MarkovShield Security-Lösung auch nach dem Ende der Bachelorarbeit zusammen mit dem Betreuer weiter zu entwickeln. Ein konkretes Ziel nach der Arbeit besteht darin, die Security-Lösung im Hacking-Lab[3] zu integrieren. Mittels dieser Integration können sehr wertvolle Erkenntnisse über das Laufzeitverhalten im produktiven Umfeld gesammelt und für künftige Weiterentwicklungen verwendet werden.

[3] (hacking-lab.com, 2017)

I.3. Inhaltsverzeichnis

I.4. Aufgabenstellung

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

1 Ausgangslage

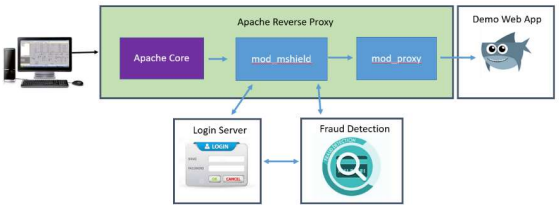
1.1 Einleitung

Viele Schweizer Banken leiden im E-Banking Bereich an sogenannten "Man in the Browser" Trojanern (MitB). Das sind spezielle Browser-Trojaner, welche unbemerkt versteckte Aktionen ausführen. Beispielsweise können diese MitB Trojaner im Hintergrund eine Auslandsüberweisung erfassen und das Konto des Kunden leerräumen.

Für eine Bank ist es sehr schwierig, solche Trojaner Aktivitäten zu erkennen, da dieser im Browser des Benutzers läuft. Durch das Aufkommen von Machine Learning Algorithmen hat sich die Situation zu verbessert. Durch Clickstreaming Analysen unter Anwendung von Markov Ketten gibt es Ansätze einen Unterschied zwischen dem Verhalten des Benutzers und dem Verhalten des Trojaners zu erkennen und die Aktionen des Trojaners unschädlich zu machen.

In einer den Studenten verfügbaren, jedoch geheim klassifizierten Master Thesis, wurde ein post-processing Approach in der Programmiersprache "R" entwickelt. Im Rahmen von dieser HSR Bachelor Arbeit soll durch Philip Schmid und Matthias Gabriel ein Real-time Erkennungssystem entwickelt werden.

1.2 Rahmenbedingung



Die Lösung soll als Apache Reverse Proxy implementiert sein, welche man vor die eigentlich zu schützende Webanwendung schaltet. Der Reverse Proxy übernimmt die Funktion es Türwächters.

Idealerweise muss man keine Anpassungen an der zu schützenden Web Anwendung vornehmen.

1.3 Diskussion mit den Studenten

Die Studenten Schmid und Gabriel haben zu Beginn des Projektes in verschiedenen Workshops mit Ivan Bütler die Anforderungen an das System ermittelt, um daraus eine Lösung zu konzipieren.

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL

1.4 Deliverables und Prioritäten

Die folgende Darstellung zeigt die Arbeiten und Prioritäten des MitB Detection Frameworks. Diese Liste ist am Meeting vom 22.3.2017 entstanden. Die mit HOCH spezifizierten Arbeiten sind Kernkomponenten die für das Arbeiten des Systems absolut notwendig sind.

Komponente	Details	Priorität
Apache Modul	Entwicklung mod_mshield auf Basis von mod_but. Erweiterung für das Fraud Detection	HOCH
Infrastruktur	System Engineering der Risk Engine Basis Infrastruktur. Damit ist die Message Queue und Persistierung der Daten und Models gemeint	HOCH
Logik Risk Engine	Umsetzung der Fraud Detection Algorithmen anhand der Master Thesis	HOCH
Testing Fraud Detection	Aussage über das Laufzeitverhalten des Fraud Detection System.	HOCH
Engine API	Engine API für den Login Service, damit dieser den Suspicious Level eines Benutzers abfragen kann. Workaround: Der Login Service simuliert das Suspicious Level.	TIEF
Update Login Service	Aktualisierter Login Service mit dem API Call zur Engine und Bereitstellung vom Step-Up mit SMS der E-Mail Token Workaround: Der Login Service simuliert das Verhalten. Statische Page	TIEF
Web GUI Konfiguration und URL	Bereitstellung einer Webanwendung für die Konfiguration der zu schützenden Web App und Bewertung der Kritikalität (siehe URL Risk Level)	TIEF

Komponente	Details	Priorität
	Workaround: Wir geben der Engine die Daten als Konfig File mit.	
Demo Case	Bereitstellung der Ergebnisse als Docker, OVA oder ähnliches für Personen, welche das Framework später austesten wollen	TIEF

1.5 Auftrag

Entwicklung eines Real-time Fraud Detection Systems auf Basis eines Reverse Proxy und unter Anwendung der Machine Learning Algorithmen der Master Thesis. Prüfung der These und Validierung der Echtzeitanforderungen.

1.6 Erwartetes Ergebnis

Es wird ein funktionierender Prototyp erwartet, welcher die Kernfunktionen (HOCH) umfasst und im Hacking-Lab integriert ist.

II. TECHNISCHER BERICHT

II.1. Einführung

II.1.1. Problemstellung und Vision

Sicherheit ist in der IT eine Thematik, welche in den letzten Jahren immer wichtiger wurde. Insbesondere, aber nicht nur in der Finanzindustrie, sind die Investitionen in die IT-Sicherheit bedeutend verstärkt worden. Parallel zu den Investitionen der Firmen wächst allerdings auch die aufgewendete kriminelle Energie und es werden immer wieder neue Angriffsvektoren auf die Systeme und deren Benutzer entdeckt. Eine Attacke, welche auf solchen neuen Angriffsvektoren basiert, wird «Man-in-the-Browser»[4] genannt.

Kurzgefast wird dabei der Datenaustausch zwischen dem Browser des Benutzers und der Webapplikation direkt im Browser des Benutzers manipuliert.[5] Dadurch können bestehende Sicherheitsmassnahmen wie die Verschlüsselung der Verbindung oder eine Erkennung von bösartigen Zugriffen anhand von computerabhängigen Merkmalen, wie der IP-Adresse, umgangen werden.

Mithilfe von neuen Analysen soll diese bestehende Bedrohung durch «Man-in-the-Browser»-Angriffe und somit auch der grosse finanzielle Schaden minimiert werden. Die neu getroffenen Massnahmen sollen dabei möglichst wenig zusätzliche Interaktion mit dem Benutzer erfordern und dabei die User Experience nicht vermindern.

Eine möglichst einfach zu startende Demo-Applikation soll den Zugang von interessierten Personen zu MarkovShield vereinfachen und die Möglichkeiten einer solchen Analyse-Software aufzeigen.

Durch den geplanten Einsatz im Hacking-Lab kann der Einsatz der Lösung in einer realen Umgebung getestet und allfällige Verbesserungsmöglichkeiten realisiert werden.

[4] (Gühring, 2006)
[5] Eine detaillierte Erklärung ist im Kapitel II.2.1 Man-in-the-Browser zu finden.

II.1.2. Ziele und Unterziele

Im Rahmen der Bachelorarbeit soll eine neue Security-Lösung entwickelt werden, welche darüber entscheidet, ob der aktuellste Request zulässig ist. Die Bewertung wird jeweils vom Reverse Proxy vor der Weiterleitung eines kritischen Requests angefordert. Dadurch ist es möglich die Webapplikation vor allfällig kompromittierten Requests zu schützen.

Für die Analyse werden bestimmte Session-Metadaten, wie zum Beispiel die Reihenfolge der aufgerufenen Webseiten analysiert und mit einem Erwartungswert verglichen. Dieser Wert wird für jeden Benutzer regelmässig anhand von historischen Daten berechnet. Auf den Einbezug von Applikationsspezifischen Daten wird bewusst verzichtet, damit die Lösung möglichst flexibel einsetzbar ist. Zudem existieren für den Hauptanwendungsfall, den E-Banking Applikationen, bereits umfassende Analysen der Transaktionsdaten. Durch den Verzicht ist es möglich sich von anderen Lösungen zu differenzieren.

Eine weitere Anforderung an die neue Lösung besteht darin, dass sie leicht in die bestehende Infrastruktur integriert werden kann und dass an der zu schützenden Webapplikation somit möglichst wenig angepasst werden muss.

II.1.3. Rahmenbedingungen

Als Basis der Bachelorarbeit dient eine vertrauliche Masterarbeit aus dem Jahre 2016, welche sich mit der theoretischen Machbarkeit einer Analyse von Session-Daten auf Anomalien betreffend «Man-in-the-Browser»-Attacken beschäftigt hat. Vor allem die mathematischen Berechnungen zur Erkennung der Anomalien beruhen zum grossen Teil auf den Erkenntnissen dieser Arbeit. Die wesentlichsten Erkenntnisse sind im Kapitel II.2.3 Erkenntnisse der Masterarbeit aufgeführt.

Der Hauptteil der Arbeit widmet sich deshalb der Integration einer solchen Lösung in die IT-Infrastruktur und den Schwierigkeiten, welche zusätzlich durch die Echtzeitanalyse entstehen.

II.2. Stand der Technik

II.2.1. Man-in-the-Browser

Da der «Man-in-the-Browser»-Angriff in unserer Arbeit eine wichtige Rolle einnimmt, ist es von Vorteil einen etwas detaillierteren Einblick in die Funktionsweise dieses Angriffes zu werfen.

Der Grundgedanke des Angriffes ist dem des viel bekannteren «Man-in-the-Middle»-Angriffs sehr ähnlich[6]. Die vom Benutzer eingegebenen Daten sollen manipuliert und an den Webserver weitergeleitet werden. Dabei wird aber im Gegensatz zum «Man-in-the-Middle»-Angriff nicht die Verbindung zwischen dem Computer und dem Webserver manipuliert, sondern der Browser wird direkt angegriffen. Dies erlaubt es dem Angreifer sowohl die vom Benutzer eingegebenen Daten als auch die darzustellende Webseite direkt zu manipulieren. Wichtig zu beachten dabei ist, dass diese Manipulationen vom Benutzer nicht erkennt werden können, da auf die richtige Webseite zugegriffen wird und auch die verschlüsselte Verbindung intakt ist.

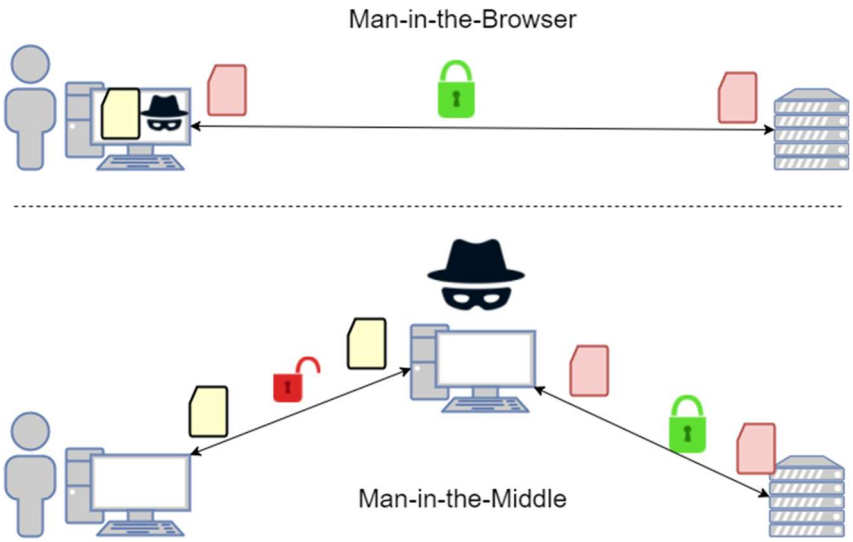


Abbildung 2 Unterschiede von Man-in-the-Browser und Man-in-the-Middle

Die Manipulationen im Browser werden je nach Trojaner auf eine andere Art und Weise durchgeführt. Unter anderem geschieht dies über Browser Helper Objects, Browser Extensions, DOM Module Interfaces, UserScripts oder API-Hooking. Auf eine Vertiefung der verschiedenen Angriffspunkte wird bewusst verzichtet, da diese für den Kontext dieser Arbeit nicht von grösserer Bedeutung sind.

[6] (OWASP, 2016)

Ein wichtiger Punkt für das grundsätzliche Verständnis ist jedoch, dass der Angriff erst nach der erfolgreichen Authentifizierung eingreift und somit auch eine klassische Two-Factor-Authentification erfolgreich umgeht.[7]

Ebenfalls ist es für den Betreiber der Webapplikation nicht mehr möglich anhand von computerspezifischen Daten eine Unterscheidung zwischen gerechtfertigten und betrügerischen Transaktionen zu machen, da alle Anfragen direkt vom Computer des Benutzers stammen. Bisherige Analysen haben häufig Werte, wie IP-Adresse, den Fingerprint des Browsers oder Ähnliches verwendet um potentiell missbräuchliche Transaktionen aufzufinden.

II.2.2. Bestehende Lösungsansätze

Da der Schaden durch «Man-in-the-Browser» Angriffe relativ hoch einzuschätzen ist, bestehen bereits verschiedenste Lösungskonzepte, welche ihren Teil zur Bekämpfung des Trojaners übernehmen. Die verschiedenen Massnahmen können dabei grob in folgende Kategorien unterteilt werden:

- Antivirus
- Hardened software
- Out-of-band transaction verification
- Server side security

II.2.2.1. Antivirus

Sobald die Programmsignaturen der verwendeten Trojaner bekannt sind, kann versucht werden diese über einen klassischen Antivirus zu entdecken und zu entfernen. Es ist anzunehmen, dass die Erfolgsrate der Antiviren-Software im Bereich «Man-in-the-Browser» relativ tief ist. Dies liegt unter anderem an der sich ständig ändernden Malware[8].

Leider fanden sich keine aktuellen Zahlen zur Effizienz von Antiviren-Softwares in Bezug auf «Man-in-the-Browser»-Attacken.

[7] (Cain, 2014)
[8] (Williamson, 2014)

II.2.2.2. Secure Client

Unter dem Überbegriff Secure Client sind verschiedene Lösungsansätze zusammengefasst, welche sich mit dem Client auseinandersetzen. Unter anderem ist dabei das Ziel die Software, mit welcher die zu schützenden Transaktionen durchgeführt werden, möglichst sicher zu gestalten. Ein Problem welches sich hier immer wieder zeigt ist, dass der Client normalerweise ausserhalb des Einflussbereichs des Betreibers liegt. Ausserdem stehen diese Massnahmen oft in direktem Konflikt mit der Benutzerfreundlichkeit.

Eine Lösung, welche immer wieder aufgeführt ist, nennt sich Hardened Browser. Man versteht darunter einen speziellen Browser, welcher zusätzliche Security Requirements erfüllt. Zum Beispiel erlauben solche Browser keine Erweiterungen, sind statisch kompiliert und somit von den Trojanern schwieriger anzugreifen. Ein solcher Browser würde zusätzlich zum normalen Browser auf dem Client-Computer installiert. Der Hardened Browser würde vom Benutzer im Normalfall ausschliesslich für Webseiten mit erhöhter Sicherheit genutzt werden, da er im Vergleich zu einem normalen Browser weniger Komfort bietet.

Zwei generelle Probleme, welche sich bei Secure Clients immer wieder stellen, sind die serverseitige Unterscheidung zwischen Secure und Insecure Clients sowie die Sensibilisierung der Benutzer einen Secure Client zu benutzen.

II.2.2.3. Out-of-band transaction verification

Unter «out-of-band transaction verification» versteht sich im Normalfall eine Verifikation der Transaktion auf einem anderen Weg. Dies funktioniert ähnlich, wie die weitverbreitete Two-Factor-Authentifizierung und kann zum Beispiel über ein SMS oder einen Telefonanruf durchgeführt werden. Ein wesentlicher Unterschied dazu ist, dass die Verifikationsnachricht Informationen über die Transaktion enthält. Dadurch ist es dem Endbenutzer möglich, die gewünschten Transaktionsdetails mit den über den zweiten Kanal erhaltenen realen Transaktionsdetails zu vergleichen. Für dieses Verfahren ist jedoch eine sehr gute Benutzersensibilisierung notwendig. Ein Problem, welches hierbei auftritt ist, dass weder das häufig verwendete Smartphone noch der Übertragungskanal über SMS für sichere Informationen ausreichend geschützt sind. Somit ist die zusätzliche Verifikation der Transaktion zwar eine weitere Schutzmassnahme, welche jedoch unter Umständen durch eine zusätzliche Infektion des Mobilgeräts aufgehoben werden kann.

Die sicherste Variante von «out-of-the-band transation verification» wird EMV Cap OTP mit Signatur genannt und benötigt ein zusätzliches Gerät.[9] [10] Dabei muss der Benutzer Teile der Transaktion im eigenständigen Gerät eintippen, welches im Anschluss die Transaktion signiert.

II.2.2.4. Anomalie Detection

Bei der Anomalie Detection wird, wie der Name schon sagt, versucht mit verschiedenen Analysen verdächtigte Transaktionen zu finden und gegebenenfalls zusätzlich zu überprüfen. Diese Analysen basieren entweder auf Meta-Daten, wie zum Beispiel des IP-Adress-Verlaufs der Person, oder des Fingerprints des Browsers, oder aber direkt auf Transaktions-Daten wie zum Beispiel Zielkonto oder Betrag der Transaktion.

II.2.3. Erkenntnisse der Masterarbeit

Da diese Arbeit stark auf den Erkenntnissen der erwähnten Masterarbeit aufbaut, sind im folgenden Kapitel die wichtigsten Erkenntnisse hervorgehoben.

Grundsätzlich fällt die Art der Vorgehensweise unter die bereits erwähnte «Anomalie Detection». Hier wird versucht aufgrund von Transaktions-Meta-Daten eine Analyse durchzuführen um manipulierte Transaktionen ausfindig zu machen.

Grundsätzlich sieht die Masterarbeit vor, regelmässig pro Benutzer mehrere «Models» zu berechnen, welche das benutzerspezifische Verhalten festhalten. Danach können die Metadaten

[9] (alanthl, 2011)
[10] (Entrust Inc., 2014)

von einzelnen Sessions mit dem Model überprüft werden. Dabei wird ein Scoring-Process genutzt, auf welchen in den nachfolgendenden Kapiteln genauer eingegangen wird.

Es wird dabei zwischen zwei konkret verwendeten Models unterschieden:

- Transition Model
- Frequency Model

II.2.3.1. Transition Model

Eines der Models, welches für das Scoring verwendet wird, ist das Transition Model. Hier werden die Übergänge zwischen den verschiedenen URLs analysiert, welche ein Benutzer normalerweise während einer Session durchläuft. Damit können Unstimmigkeiten im Ablauf des Clickstreams gefunden werden. Für das Model wird dabei eine Transition Matrix erstellt, welche jegliche Übergänge mit ihrer jeweiligen Eintritts-Wahrscheinlichkeit festhält. Da die verschiedenen Webseiten hier als Zustand in einem System modelliert werden können und jeder Zustand des Systems nur vom vorherigen Zustand abhängig ist, kann dies auch als Markov-Chain bezeichnet werden.[11]

Ein kleines Beispiel; angenommen es existieren folgende Seiten auf der Webseite:

- start.html (a)
- overview.html (b)
- transation.html (c)
- news.html (d)
- logout.html (e)

Dann sieht die Transition Matrix zum Beispiel wie folgt aus:

	start.html	overview.html	transaction.html	news.html	logout.html
start.html	0	0	0	0	0
overview.html	1	0.25	0.5	0.25	0
transaction.html	0	0.5	0	0	0
news.html	0	0.25	0	0.25	0
logout.html	0	0	0.5	0.5	0
Ende der Session	0	0	0	0	1

Tabelle 1 Beispiel einer Transition Matrix

[11] (Wai-Ki Ching, 2013)

In jeder Spalte ist dabei eine Ursprungswebseite und in jeder Zeile eine Zielwebseite eingetragen. In jedem Feld wird nun die Wahrscheinlichkeit eingetragen, dass der Benutzer von der Ursprungswebseite auf die Zielwebseite wechselt. Die Summe einer Spalte ist dabei immer exakt 1.

Konkret bedeutet dies, dass der Benutzer, nachdem er die Seite «overview.html» aufgerufen hat, in 25% der Fälle nochmals dieselbe Seite aufruft, in 50% der Fälle auf die Seite «transaction.html» wechselt und in 25% der Fälle «news.html» besucht.

Visuell dargestellt ergibt sich folgendes Bild, welches die gleichen Zustandsübergänge abbildet:

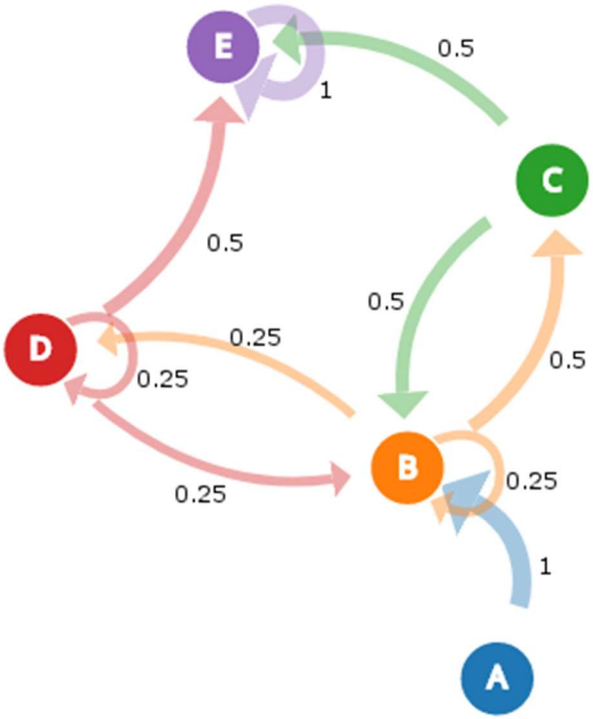


Abbildung 3 Visualisierung einer Markov Chain

II.2.3.2. Frequency Model

Zusätzlich zum Transition Model wird ein Frequency Model berechnet, um Anomalien bei der Anzahl von Aufrufen einer bestimmten Seite zu entdecken.

Grundsätzlich werden beim Frequency Model lediglich die Anzahl der Aufrufe auf eine bestimmte Seite gezählt. Daraus wird dann ein maximales Limit definiert, mit welchem die aktuelle Session verglichen wird.

II.2.3.3. Scoring

Für das Scoring einer Session wird diese unabhängig voneinander mit jedem Model geprüft. Dabei generiert jedes Model einen eigenen Scoring-Wert, welche dann schlussendlich aufsummiert werden. Zusätzlich dazu wird vor der Inbetriebnahme der Lösung für jede URL ein sogenanntes Risk Level definiert, welches dann in den Scoring-Wert als Multiplikator einfliesst. Dadurch können Unregelmässigkeiten bei Seiten mit erhöhtem Risiko stärker in das Scoring einbezogen werden als Unregelmässigkeiten bei normalen Seiten. Es sind dabei drei verschiedene Risiko-Level vorgesehen:

- Low Risk Seiten ohne Kundendaten
- Medium Risk Seiten mit Kundendaten wie z.B. Account-Nummern
- High Risk Seiten auf welchen Datenmutationen durchgeführt werden können

Transition Model

Bei der Berechnung des Transition Model Scores eines bestimmten Clickstreams wird jede Transition isoliert mit dem Transition Model verglichen. Wenn diese Transition laut dem Model eine 0%-Chance für eine solche Transition hat, wird der Transition Score des Clickstreams erhöht. Wie hoch diese Erhöhung ausfällt ist dabei vom Risk Level der Ziel-URL abhängig.

Frequency Model

Bei der Auswertung eines Clickstreams in Bezug auf das Frequency Model wird die Anzahl der Clicks pro Seite gezählt. Ist diese Anzahl grösser als die maximal zulässige Zahl, so wird der Frequency Score des Clickstreams erhöht. Diese Erhöhung ist wiederum vom Risk Level der Ziel-URL abhängig.

II.2.4. Defizite

Die meisten bisherigen Lösungsansätze haben das Grundproblem, dass mindestens ein Teil der Sicherheitskomponenten ausserhalb der Verfügungsgewalt des Unternehmens liegt und das Unternehmen dadurch auf die Kooperation des Kunden angewiesen ist. Aus der Sicht des Kunden ist jede zusätzliche Sicherheitsvorkehrung jedoch oft primär eine Einschränkung und

eine Reduktion der Benutzerfreundlichkeit. Von den erwähnten Lösungsansätzen benötigt lediglich die Kategorie «Anomalie Detection» keine Unterstützung des Kunden. Bisherige «Anomalie Detection»-Systeme setzten vor allem auf die Analyse der Transaktionsdaten, wie zum Beispiel das Empfänger-Konto und die Höhe des zu überweisenden Betrages. Dadurch sind sie sehr stark auf die zu schützende Applikation angepasst und benötigten ein hohes Domänenwissen. Ebenfalls ist häufig eine gute Integration des Analysesystems in die Applikation nötig.

II.2.4.1. Masterarbeit

Während der Bachelorarbeit wurde intensiv mit den gewonnenen Erkenntnissen der Masterarbeit sowie des implementierten Prototyps gearbeitet. Dabei wurde in der Berechnung des Scorings bei beiden verwendeten Models eine mögliche Schwachstelle festgestellt.

Bei der Auswertung des Transition Models wird für jede Transition in einem Clickstream die Wahrscheinlichkeit des Auftretens überprüft. In der Implementation wird die Höhe der Auftretenswahrscheinlichkeit jedoch ignoriert. Dies deutet jedoch auf einen Fehler hin, da bei der Berechnung des Models die verschiedenen Wahrscheinlichkeiten eine grosse Rolle spielen. Im Kapitel III.8.1.6 Model Builder Implementationsdetails wird genauer auf die erarbeitete Lösung eingegangen. Wenn trotzdem auf die alte Berechnung des Transition Scores zurückgegriffen werden soll, empfiehlt es sich das Model entsprechend anzupassen und nur einen binären Wert zu bestimmen. Dadurch kann die Berechnung des Models stark vereinfach werden.

Bei der Berechnung des Frequency Models werden mithilfe der externen R-Library «extremevalues» Outlier detektiert und zusätzlich ausgewertet. Es wird überprüft, dass der Wert kleiner ist als x-mal die Standardabweichung, wobei x konfigurierbar ist. Dies jedoch ohne den Erwartungswert miteinzubeziehen. Stattdessen ist das standardmässige x auf 10 eingestellt. Bei einer Normalverteilung sind bereits in einem Intervall mit der dreifachen Standardabweichung vom Erwartungswert über 99.7% der Messwerte zu finden.[12]

Für die Berechnung des Frequency Scores wurde danach lediglich der höchste aufgetretene Wert mit dem aktuellen Wert verglichen. Wenn der neue Wert grösser ist als der alte Maximalwert, wird dies als Verstoss angesehen. Vor der Verwendung dieses Models empfiehlt es sich diese Punkte genau zu untersuchen. Im Kapitel III.8.1.7 Model Implementationsdetails wird genauer auf den in MarkovShield verwendeten Lösungsansatz eingegangen.

[12] (Götze & Deutschmann, 2002)

II.3. Umsetzungskonzept

In diesem Kapitel wird eine grobe und allgemein verständliche Übersicht über die verwendeten Grundkonzepte vom MarkovShield erläutert. Weiterführende Informationen sind im Teil III SW-Projektdokumentation nachzulesen.

II.3.1. Namensgebung

Zu Beginn der Arbeit bestand die Herausforderung, einen sprechenden, einzigartigen und zugleich einfach merkbaren Produktnamen für die Gesamtlösung zu finden. Dies sollte insbesondere die Kommunikation im Projektteam intern als auch die Erklärung des Projektes gegenüber Aussenstehenden vereinfachen.

Als Produktname wurde «MarkovShield» gewählt. Der erste Teil «Markov» steht für den russischen Mathematiker Andrey Markov[13] (14 Juni 1956 - 20 Juli 1922) und seine Theorie über die Markov Chains[14], welche im Rahmen dieser Arbeit einen entscheidenden Einfluss haben. Der zweite Teil «Shield» steht für die verallgemeinerte Funktionalität der Gesamtlösung, nämlich eine Webapplikation gegen unbefugte Zugriffe zu schützen.

II.3.2. Beschreibung des Lösungskonzeptes

Die Funktionalität von MarkovShield kann grundsätzlich folgendermassen unterteilt werden:

- Informationsbeschaffung
- Datenpersistierung
- Analyse der Daten
- Aufbau der Models

Zusätzlich dazu kommen noch mehrere Nebenkomponenten, welche für die Lösung zusätzlich unterstützen oder erweitern.

[13] (Wikipedia.org, 2017)
[14] (Wikipedia.org, 2017)

II.3.3. Informationsbeschaffung

Während der Bachelorarbeit wurde das Apache Modul «MOD_BUT» von Ivan Bütler gefokt und anschliessend erweitert. Dadurch können die von der Analysesoftware benötigten Meta-Daten direkt auf dem Apache Reverse Proxy extrahiert werden. Die Reverse Proxy Funktionalität wird mit Hilfe von mod_proxy[15], mod_headers[16], mod_rewrite[17] diversen weiteren Apache Modulen realisiert und in Verbindung mit dem eigenen Modul und ModSecurity[18] (optional) erweitert. In der Systemarchitekturansicht wird der Reverse Proxy zwischen der zu schützenden Applikation und den Clients platziert. Auf dieser Zwischenkomponente können für den Client transparent diverse Sicherheitsaspekte wie Pre-Authentication eingeführt und betrieben werden.

Bisherige Funktionalitäten von MOD_BUT[19]:

- Pre-authentication
- Support für verschiedene Authentifizierungs-Levels
- Session hiding
- Single-Sign-On (SSO)

Durch die in dieser Arbeit neu implementierten Funktionalitäten ist es möglich, zu jedem Request Meta-Informationen wie zum Beispiel die aufgerufene URL, den genauen Zeitpunkt des Aufrufs und die aktuelle Session an die entwickelte Analysesoftware weiterzuleiten. Zudem ist es möglich jede URL einem Risk Level zuzuordnen, welche bereits im Kapitel II.2.3.3 Scoring kurz erwähnt wurde. Dieses Risk Level wird zusätzlich zu den bereits erwähnten Meta-Daten an die Analysesoftware gesendet und kann von den verschiedenen Models für das Scoring benutzt werden. Die Risk Level sind dabei, im Gegensatz zum Vorschlag der Masterarbeit, nicht auf drei verschiedene Stufen festgelegt, sondern es sind ganzzahlige Werte im Intervall von Null bis Tausend erlaubt. Durch den Wegfall der Einschränkung wird die Flexibilität erhöht und reale Szenarien können besser abgebildet werden.

Neben dem Risk Level für die URLs kann auch ein Risk Threshold definiert werden. Wenn das Risk Level einer aufgerufenen URL den Grenzwert überschreitet, ändert sich das Verhalten des Moduls. Nachdem es die Informationen an die Analyse-Software geschickt hat, blockiert es den Request des Benutzers solange, bis ein Analyseresultat vorliegt. Entsprechend dem Resultat werden dann weitere Schritte vorgenommen. Es gibt folgende Analyseresultate:

---

[15] (The Apache Software Foundation, 2017)
[16] (The Apache Software Foundation, 2017)
[17] (The Apache Software Foundation, 2017)
[18] (Trustwave Holdings, Inc., 2017)
19 Für genauere Informationen über die bestehende Funktionalität sei auf die MOD_BUT Dokumentation verwiesen.

Resultat	Bedeutung	Weiterführende Schritte
OK	Der Clickstream der Session scheint ok zu sein	Der Request wird normal an die Applikation weitergeleitet.
SUSPICIOUS	Im Clickstream wurden Unregelmässigkeiten erkannt	Der Benutzer muss seine Identität zusätzlich verifizieren. Dafür wird ein Step-Up angefordert. Der Benutzer kann sich dabei zum Beispiel durch die Eingabe eines SMS-Tokens zusätzlich authentifizieren[20].
FRAUD	Der Clickstream hat starke Unregelmässigkeiten ist sehr wahrscheinlich gefährlich	Die Session des Benutzers ist mit ziemlicher Sicherheit infiziert und wird deshalb beendet. Ebenfalls wäre es denkbar weitere Zugriffe des Benutzers temporär zu verhindern und manuell über den eventuellen Befall seines Systems zu informieren.[21]

Tabelle 2 Analyseresultate

Das ergänzte Apache Modul wird nach Absprache mit dem Ersteller neu mod_mshield genannt, um eine klare Abgrenzung zwischen den zwei Versionen zu haben.

[20] Im Kapitel III.2.5 Bedenken bezüglich Step-Up wird auf die Problematik dieses Vorgehens eingegangen
[21] Mehr zu weiterführenden Ideen im Kapitel III.12.1 Möglichkeiten der Weiterentwicklung

Folgende Daten werden bei jedem Click an die Analyse-Software übergeben:

Bezeichnung	**Beschreibung**
sessionUUID	Eindeutige Nummer, welche die Session identifiziert, in welcher der Click ausgeführt wurde.
clickUUID	Eindeutige Nummer, welche den einzelnen Click identifiziert.
urlRiskLevel	Wert, welcher das Risiko der aufgerufen URL wiederspiegelt[22].
url	Wert, welcher der aufgerufenen URL entspricht.
timeStamp	Wert, welcher dem genauen Zeitpunkt des Aufrufes entspricht.
validationRequi-red	Wert, welcher bestimmt, ob eine Analyse benötigt wird.

Tabelle 3 Click-Informationen

Zusätzlich dazu werden bei einem Login folgende Daten an die Analyse-Software übergeben:

Bezeichnung	**Beschreibung**
sessionUUID	Eindeutige Nummer, welche die Session identifiziert, in welcher der Click ausgeführt wurde.
userName	Der User, welcher mit dieser sessionUUID assoziiert werden kann.

Tabelle 4 Session-Informationen

[22] Siehe II.2.3 Erkenntnisse der Masterarbeit

II.3.4. Datenpersistierung

Die vom mod_mshield extrahierten Daten müssen erstens zur Analyse-Engine kommen und zweitens für das spätere Verarbeiten, zum Beispiel für den Aufbau der Models, persistiert werden. Zuerst wurden diese zwei verschiedenen Aufgaben getrennt voneinander betrachtet. Es wurde einerseits eine Lösung für die Kommunikation zwischen den verschiedenen Komponenten und andererseits eine Möglichkeit zur Persistierung aller gesammelter Daten gesucht.

Während der Evaluation der Komponenten und spezifischen Produkten hat sich jedoch herausgestellt, dass es ein Produkt gibt, welches beide Aufgaben in Kombination löst. Apache Kafka beschreibt sich selbst als «distributed streaming platform»[23] und bietet folgende Kernfunkionalitäten an:

- Publish & Subscribe to streams of records
- Process streams of records
- Store streams of records

In MarkovShield werden alle diese Kernfunktionalitäten benutzt. Die gesammelten Daten werden direkt vom mod_mshield in ein bestimmtes Topic geschrieben, welches der Publish & Subscribe Funktionalität entspricht. Danach werden die verschiedenen Clicks der Benutzer aggregiert und mit weiteren Informationen angereichert, welches der Funktionalität «Process» entspricht.[24] Zusätzlich dazu bietet Kafka eine grosse Auswahl an Konfigurationsmöglichkeiten in Bezug auf die Persistierung der Daten. Jede Message, welche in Kafka published wird, wird automatisch in einer internen Datenbank gespeichert und kann repliziert werden, sofern dies erwünscht ist. Zudem bietet Apache Kafka die Möglichkeit die Daten nur eine gewisse Zeit lang zu persistieren, eine sogenannte Retention-Policy, und danach automatisch zu entfernen. Dies wird über eine sogenannte Retention-Policy definiert und kann bei temporärem Zwischenspeichern sehr hilfreich sein. Im Kapitel III.6.2 Persistenz wird genauer auf die verschiedenen Konfigurationsmöglichkeiten von Apache Kafka und die gewählten Einstellungen eingegangen.

II.3.5. Analyse der Daten

Für die realtime Analyse der Clickstream-Daten wird eine zusätzliche Komponente eingesetzt. Eine wichtige Eigenschaft dieser Komponente ist das sogenannte «True Streaming Processing», dies bedeutet, dass jedes Event einzeln abgearbeitet wird. Dies ist zwingend notwendig um eine möglichst tiefe Latenz zu erreichen. Es wird deshalb die Stream Processing Engine

[23] (Apache Software Foundation, kein Datum)
[24] Mehr dazu in den Kapiteln III.3.7 Zusätzliche Funktionalität sowie III.8.1.8 GlobalKTable vs KTable

Apache Flink eingesetzt. Eine detaillierte Auswertung der verschiedenen Streaming-Engines kann im Kapitel III.4.1 Engine nachgelesen werden.

Bei der Analyse wird bei jedem Aufruf einer kritischen URL der gesamte bisherige Clickstream mit mehreren userspezifischen Models geprüft und je ein Wert berechnet, welches das momentane Verdachtslevel wiederspiegelt. Der Durchschnitt dieser Werte wird danach in die drei möglichen Kategorien OK, SUSPICIOUS und FRAUD eingeteilt.[25] Wichtig zu beachten ist dabei, dass für jeden Aufruf einer kritischen URL eine eigene Analyse durchgeführt wird. Nachdem das Analyseresultat für den momentanen Clickstream berechnet wurde, wird dieses zurück an das Apache Modul geliefert, sodass die weiterführenden Schritte eingeleitet werden können. Dabei wurde die Kommunikation des Resultats durch die Verwendung eines Key-/Value-Stores entkoppelt.[26]

Zusätzlich dazu werden die Clickstreams mit dem jeweiligen Resultat in Apache Kafka geschrieben, um später die userbasierten Models neu zu berechnen.

II.3.6. Berechnung der Models

Um die verschiedenen Models jedes Benutzers aktuell zu halten, müssen diese regelmässig neu berechnet werden. Neben der realtime Analyse erledigt die Stream Processing Engine daher auch die Berechnung der Models. Dafür stehen alle Clickstreams inklusive des damaligen Analyseresultats eines Benutzers zur Verfügung, welche sich in einem definierten Zeitbereich befinden. Bei der Standardeinstellung ist dieser Zeitbereich das letzte halbe Jahr. Ältere Clickstreams gelten als veraltet, da sich das Benutzerverhalten über in dieser Zeit verändert haben könnte. Aus diesem Grund werden sie nicht mehr in die Berechnungen des Models miteinbezogen. Ebenfalls herausgefiltert werden jegliche Clickstreams, welche das Rating FRAUD oder SUSPICIOUS erhalten haben. Die bestehenden Implementationen für die Berechnung der Models basieren stark auf der Masterarbeit. Trotzdem wurden aus Gründen der Einfachheit gewisse Anpassungen vorgenommen. Ein Austausch oder eine Erweiterung der Models sollte mit einem geringen implementations-spezifischen Aufwand verbunden sein, da diese Codeteile möglichst modular aufgebaut wurden.

II.3.6.1. Transistion-Model

Eines der implementierten Models ist das Transition Model, welches die Wahrscheinlichkeit einer gewissen Transition festhält. Dieses basiert vollständig auf den im Kapitel II.2.3.1 Transition Model beschriebenen Erkenntnissen.

[25] Tabelle 2 Analyseresultate
[26] Mehr dazu im Kapitel III.8.1.1 Kommunikation des Ergebnisses

II.3.6.2. Frequency Model

Zwei weitere implementierte Models sind sogenannte Frequency Models. Diese Model sollen eine Aussage darüber treffen, wie wahrscheinlich eine gewisse Anzahl von Seitenaufrufen pro Seite ist. Die in der Masterarbeit vorgesehene Berechnung ist im Kapitel II.2.3.2 Frequency Model festgehalten. Bei der genaueren Begutachtung sind jedoch gewisse Defizite aufgetreten, welche im Kapitel II.2.4.1 Masterarbeit dokumentiert sind. Deshalb wurde dieses Model mit einer teilweise abweichenden Implementierung realisiert. Generell wird von einer Normalverteilung der einzelnen Frequencies ausgegangen. Im einen Model werden die potentiellen Outlier deshalb über den Erwartungswert und die Standardabweichung bestimmt werden. Dabei wird davon ausgegangen, dass eine dreifache Standardabweichung ein gutes Resultat liefert, da damit statistisch gesehen über 99.7% der potentiellen Datenpunkte innerhalb der Verteilung liegen.

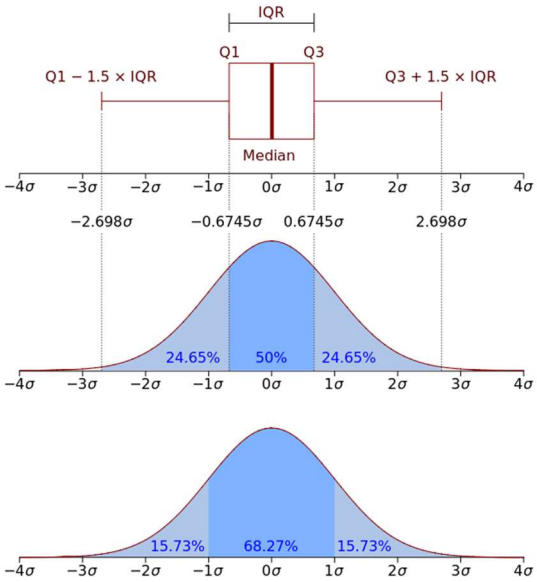


Abbildung 4 IQR vs Probability Density Function[27]

Das andere implementierte Frequency Model setzt auf Inter-Quartile-Range (IQR). Dies ist eine etwas weniger bekannte Berechnungsmethode, welche in der Datenanalyse oft für die Outlierdetection verwendet wird. Dabei wird die Differenz zwischen dem ersten und dem dritten Quartil, der IQR, mit dem Faktor 1.5 multipliziert und vom ersten Quartil abgezogen und zum dritten Quartil hinzugezählt. Dadurch ergibt sich ein Intervall, welches alle zugelassenen Werte enthält. Damit sind 99.3% der Werte im Intervall enthalten. Wie in der Abbildung 4 gut abgebildet, führen die zwei gewählten Varianten zu einem ähnlichen Resultat, die Variante

[27] (Jhguch & Chen-Pan, 2012)

mit IQR ist jedoch weniger anfällig auf Outlier[28], weshalb dies die momentan eingebundene Methode zur Berechnung des Frequency Models darstellt.

Die genauen Implementierungsdetails sind im Kapitel III.8.1.6 Model Builder Implementationsdetails festgehalten.

II.3.6.3. Random-Model

Neben den drei implementierten Models, welche für den produktiven Einsatz vorgesehen sind, wurde zusätzlich das sogenannte Random-Model implementiert. Dieses kann vor allem für Demonstrationszwecke eingesetzt werden und berechnet eine vom Clickstream unabhängigen Score. Dabei kann konfiguriert werden, wie viele der kritischen Requests SUSPICIOUS oder FRAUD Werte liefern sollen.

[28] (Wahl, 2013)

II.3.7. Architektur und Datenfluss

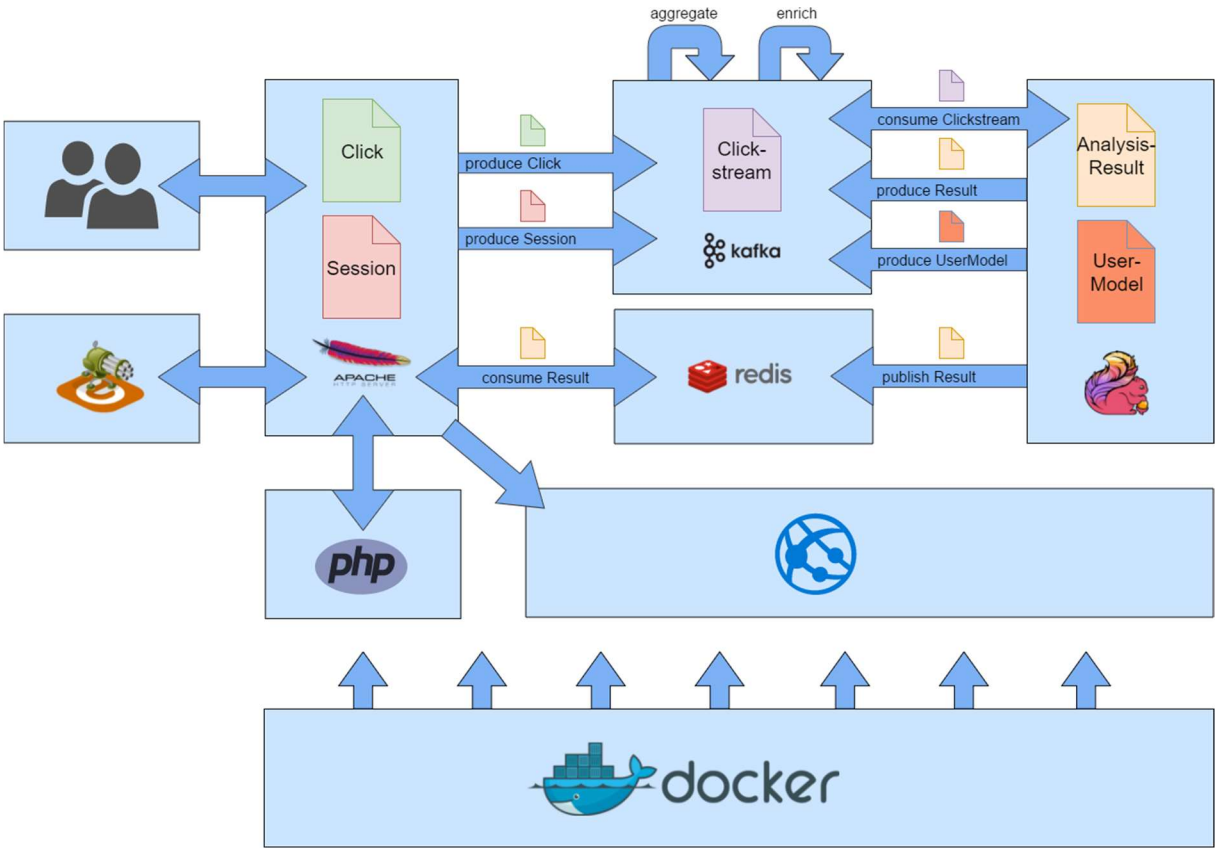


Abbildung 5 Architekturübersicht mit Datenflow

In der Abbildung 5 Architekturübersicht mit Datenflow, welche die Systemarchitektur von MarkovShield abbildet, ist zusätzlich die Datenentstehung, sowie der Datenfluss eingezeichnet. Sowohl die Clicks, als auch die Sessions entstehen im entwickelten Apache Modul mod_mshield und werden dann an Apache Kafka übergeben. Eine Session wird dabei einmalig nach dem Login erstellt, ein Click-Objekt bei jedem Request eines Benutzers. In Apache Kafka wird dann der Clickstream erstellt, welcher aus der Aggregation aller Clicks derselben Session besteht und zusätzlich durch das eventuell vorhandene User Model angereichert wird.

Apache Flink analysiert anschliessend den bereits aggregierten Clickstream. Daraus wiederum entsteht das Analyseresultat, welches sowohl an Redis als auch an Kafka übergeben wird. Redis dient dabei als Antwortkanal für mod_shield und in Apache Kafka findet die Persistierung des Resultats für die spätere Weiterverwendung statt.

Losgelöst vom Datenfluss, welcher durch einen Benutzer ausgelöst wird, findet in regelmässigen Abständen das Berechnen der Models in Flink statt. Dafür werden die persistierten Clickstream-Daten, inklusive den Analyseresultaten, aus Kafka verwendet.

II.4. Resultate

II.4.1. Zielerreichung

Während der Bachelorarbeit wurde eine funktionsfähige Infrastruktur aufgebaut, mit welcher die Analysen in Echtzeit durchgeführt werden können. Dabei wurden diverse Systeme so kombiniert, dass dieses System keine Anpassungen an der zu schützenden Applikation benötigt. Das System ist damit als reine Infrastrukturkomponente zu klassifizieren.

Während dem Projekt wurde grossen Wert auf eine starke Skalierbarkeit der einzelnen Komponenten sowohl als auch auf das gesamte System gelegt. Somit konnten auch unter Systemlast vernünftige Response-Times erreicht werden.

Bei der Implementation der beschriebenen Models aus der Masterarbeit wurde aufgrund der fehlenden Zeit und der hohen mathematischen Komplexität auf gewisse Details verzichtet. Dafür wurde bei der Architektur in diesem Bereich auf eine leichte Erweiterbarkeit geachtet.

Nachfolgend ist eine kurze Übersicht über die gewünschten Teile der Softwarelösung gegeben. Eine genauere Beschreibung der einzelnen Komponenten ist im Kapitel III.3 System Architektur zu finden.

Komponente	**Priorität**	**Gelöst**	**Kommentar**
Apache Modul	HOCH	Ja	Diese Komponente wurde ohne Restriktionen implementiert.
Infrastruktur	HOCH	Ja	Diese Komponente wurde ohne Restriktionen implementiert.
Logik Risk Engine	HOCH	Teilweise	Die Logik wurde zum Teil vereinfacht um eine verkürzte Implementationszeit zu ermöglichen.
Testing Fraud Detection	HOCH	Ja	Es wurden sowohl umfassende Systemtests als auch Load-Tests durchgeführt.
Engine API	TIEF	Ja	Eine REST-API wurde für die wichtigsten Daten erstellt um die Machbarkeit und das Potential aufzuzeigen

Update Login Service	TIEF	Nein	Der Login-Service wurde stark vereinfacht implementiert um einen kleinen Implementationsaufwand zu ermöglichen.
Web GUI Konfiguration und URL	TIEF	Nein	Auf ein Web GUI für die Konfiguration wurde komplett verzichtet.
Demo Case	TIEF	Ja	Die gesamte Infrastruktur steht als Docker Compose File zur Verfügung.

Tabelle 5 Deliverables und Prioritäten

II.4.2. Ausblick auf die Weiterentwicklung

Generell sind für die Weiterentwicklung der Lösung sehr viele Ideen vorhanden. In diesem Kapitel werden die wichtigsten kurz aufgezeigt. Für konkretere Details zu den Weiterentwicklungen sei auf Kapitel III.12.1 Möglichkeiten der Weiterentwicklung verwiesen.

II.4.2.1. User-based Rating

Nach der Analyse eines Clickstreams wird das Resultat zwischengespeichert und entsprechende Massnahmen zum Schutz der Applikation werden getroffen. Diese Massnahmen sind momentan jedoch nur während der aktiven Session anwendbar. In einer Erweiterung von MarkovShield könnten diese Massnahmen auf alle zukünftigen Sessions des Benutzers ausgeweitet werden.

II.4.2.2. Dashboard zur Visualisierung der Models und Analyseresultate

Es würde sich anbieten zur bestehenden REST-API ein Dashboard zu erstellen, auf welchem die userbasierten Models und die kompletten Analyseresultate dargestellt werden könnten. Dieses Dashboard könnte zudem erweitert werden, um Häufungen von negativen Analyseresultaten darstellen zu können.

II.4.2.3. Web UI zur Konfiguration

Zusätzlich zur Visualisierung von Daten mittels Dashboard wäre es wünschenswert, die MarkovShield Einstellungen über das Dashboard konfigurierbar zu machen. Insbesondere die Zuteilung des Risk Levels zu einer URL bietet grosses Potential zur Verbesserung der Benutzerfreundlichkeit für den Administrator der Lösung.

II.4.2.4. Erweiterung des Login Servers

Der bestehende Login Server sollte soweit erweitert werden, dass eine Two-Factor-Authentifizierung unterstützt wird. Diese wird für das Step-Up benötigt und ist momentan aus zeitlichen Gründen stark vereinfacht worden. Des Weiteren könnte der Login Server so erweitert werden, dass dieser eine Benutzerverwaltung (z.B. ein LDAP) anspricht. Zur Erhöhung der Sicherheit wäre es zudem denkbar, dass der Login Server den Login Status eines Benutzers mit einem Timestamp versieht und anschliessend den gesamten Status signiert.

II.4.2.5. Erweiterung um weitere Models

Eine weitere Möglichkeit zur Weiterentwicklung bietet sich im Bereich der Models an. Die Schwierigkeit besteht dabei vor allem darin, ein einzelnes oder mehrere neue Models auszuarbeiten. Die Integration neuer Models in MarkovShield ist dank der berücksichtigten Erweiterbarkeit sehr schnell und einfach zu realisieren.

II.4.2.6. Anbindung an eine bestehende Risk Engine

In vielen bestehenden Infrastrukturen, welche von den Analysen von MarkovShield profitieren können, wird es bereits eine Risk Engine geben. MarkovShield muss für eine Integration in eine solche Umgebung entsprechend angepasst werden.

II.4.2.7. Absicherung der Kommunikation

Vor der Nutzung in einer produktiven Umgebung sollte die gesamte Kommunikation zwischen den verschiedenen Komponenten abgesichert werden. In der Realisation wurde aus zeitlichen Gründen darauf verzichtet.

Einerseits sollten entsprechende Autorisierungssysteme auf den verschiedenen Komponenten aktiviert werden und andererseits wäre die verschlüsselte Kommunikation der verschiedenen Teile wünschenswert. Jede der verwendeten Komponenten unterstützt dabei die verschlüsselte Kommunikation.[29][30][31][32]

Des Weiteren könnte eine sinnvolle und sichere Netzwerksegmentierung für die Übertragung der verschiedenen Arten von Daten zu einer weiteren Verbesserung der internen Sicherheit des Systems beitragen.

II.4.2.8. Information einer Kontaktperson

Wenn ein Clickstream als FRAUD erkannt wurde, sollte nicht nur die momentane Session terminiert werden, sondern ebenfalls eine oder mehrere Kontaktpersonen benachrichtigt werden. Die Kontaktperson könnte dann mit dem entsprechenden Kunden in Kontakt treten und ihn über die betreffende Session und den allfälligen Befall seines Computers informieren. Es wäre stark zu empfehlen, dass sich dabei der persönliche und bereits bekannte Kundenberater beim Kunden meldet.

[29] (Juma, 2016)
[30] (librdkafka, 14)
[31] (redis, kein Datum)
[32] (Flink, kein Datum)

III. SW-PROJEKTDOKUMENTATION

III.1. Anforderungsspezifikation

Nachfolgend werden die funktionalen und nicht funktionalen Anforderungen an MarkovShield aufgezeigt und detailliert beschrieben.

III.1.1. Anforderungen an die Arbeit

Die Arbeit sollte aufgrund des zugrundeliegenden Wissens und des existierenden Post Processing Prototypen der Masterarbeit in Kombination mit neu erarbeiteten Anforderungen und Erkenntnissen eine Realtime Lösung als Resultat zum Vorschein bringen.

Die Hauptfunktionalität ändert sich dabei nicht und liegt darin, für jeden Zugriff auf die zu schützende Webapplikation zu entscheiden, ob dieser ein potentiell gefährlicher Zugriff darstellt und entsprechende Gegenmassnahmen zu treffen.

III.1.2. Actors

Die MarkovShield Applikation besitzt die folgenden Actors mit den aufgezeigten Interessen und Kenntnissen.

- *Benutzer*: Ein Benutzer hat das Interesse, denn Service, welcher durch die Webapplikation zur Verfügung gestellt wird, vollumfänglich zu nutzen und dadurch für sich einen Mehrwert zu generieren. Ihm ist bewusst, dass er der Webapplikation unter Umständen sensible Daten anvertraut und hat das Interesse, dass diese fachgerecht geschützt sind.
- *Betreiber*: Der Betreiber kennt die Bedürfnisse seiner Benutzer und ist sich bewusst, dass seine Webapplikation vor potentiellen Cyberangriffen geschützt werden sollte. Er kennt seine Webapplikation ausreichend, sodass er weiss, wo sich die Sicherheitskritischen Bereiche befinden.

III.1.3. User Stories

- *Webapplikation benutzen*: Als Benutzer der Webapplikation möchte ich den von der Webapplikation angebotenen Service im vollen Funktionsumfang und ohne nennenswerte Zwischenfälle benutzen können. Als Benutzer der Webapplikation möchte ich, dass meine persönlichen Daten, auf welche nur ich Zugriff haben sollte, zusätzlich geschützt sind. Dies insbesondere auch dann, sollte ich mit einer Man-in-the-Browser Malware infiziert werden.
- *Schädliche Aktionen verhindern*: Als Betreiber der Webapplikation möchte ich, dass eine Angriff auf die Webapplikation frühzeitig erkannt wird. Somit können keine schädlichen Aktionen ausführt werden. Zugleich hat dies den Nutzen, dass ich anschliessend keine Haftung übernehmen muss, da kein Schaden entstanden ist.

- *Kritische Bereiche definieren*: Als Betreiber der Webapplikation möchte ich beliebig definieren können, welche Webapplikationsbereiche als unkritisch oder kritisch zu betrachten sind.
- *Überblick erhalten*: Als Betreiber der Webapplikation möchte ich eine Übersicht erhalten, welche Benutzer Sessions als SUSPICIOUS oder FRAUD gekennzeichnet wurden. Dies kann mir helfen zu erkennen, welche Benutzer eventuell eine weitere manuelle Intervention benötigen.

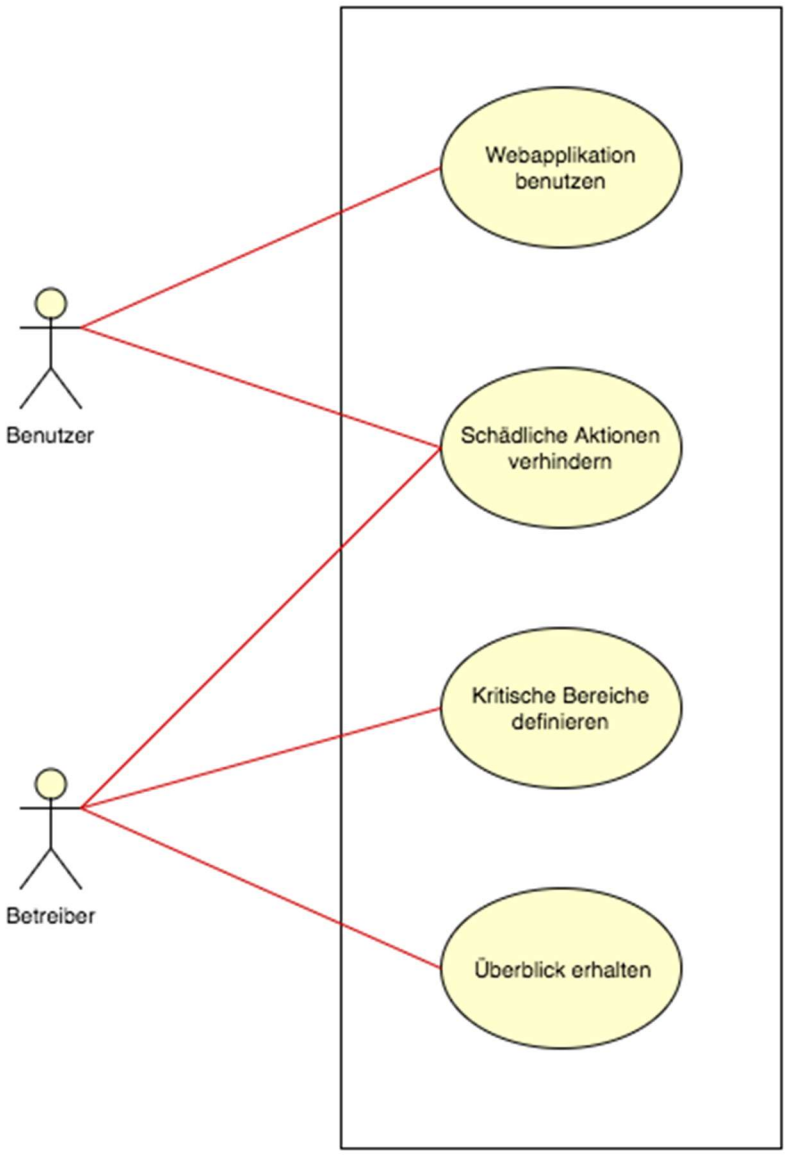


Abbildung 6 User Story Diagramm

III.1.4. Nicht-funktionale Anforderungen

III.1.4.1. Performance

Die Performance ist bei MarkovShield sehr wichtig, da sich eine Verzögerung der Analyse direkt auf die User Experience niederschlägt. Insbesondere zu beachten ist die Zeit, welche ein Request benötigt, um vom Eintreffen auf dem Reverse Proxy bis zum effektiven Weiterleiten an die Webapplikation zu gelangen.

Damit der Benutzer der Webapplikation im Normalfall nahezu nichts von all dieser Auswertung zu spüren bekommt, muss die benötigte Zeit die gesamte Analyse in 95% der Fälle unterhalb 500 Millisekunden befinden - dies bei rund maximal 100 gleichzeitigen Sessions auf dem Reverse Proxy.

Solange den User innerhalb einer Sekunde eine Antwort erreicht, scheint er die Kontrolle zu behalten.[33] Da die Analyse nur einen Teil dieser Zeit ausmacht, sollte sie nicht die ganze Sekunde benötigen. Allerdings hat sich in der Vergangenheit gezeigt, dass bereits kleine Änderungen in der Response time einen grossen Einfluss auf die Benutzerzufriedenheit haben. Aus diesem Grund sollte versucht werden die Verzögerung durch die Analyse möglichst klein zu halten.[34]

III.1.4.2. Sicherheit

Auch die Sicherheit spielt in der MarkovShield Gesamtlösung eine grosse Rolle. Ist die Fraud Detection & Prevention Funktionalität auf dem Reverse Proxy aktiviert, so muss sichergestellt werden können, dass kein Request auf eine kritische URL zur Webapplikation weitergeleitet wird, bevor diese nicht zuvor durch die MarkovShield Engine und deren Modelle überprüft wurde. Des Weiteren darf kein Request zur Webapplikation weitergeleitet werden, welcher zuvor von der Engine als SUSPICIOUS oder FRAUD bewertet wurde.

Eine Verschlüsselung der Kommunikation zwischen den einzelnen MarkovShield Komponenten muss von den Komponenten her unterstützt sein, da sich mindestens die Reverse Proxy Komponente im Regelfall in der DMZ befindet und sich dadurch erhöhte Sicherheitsanforderungen ergeben. Für den Prototyp von MarkovShield ist diese Anforderung noch nicht zwingend, da dieser nicht für den produktiven Einsatz ausgelegt ist.

[33] (Nielsen, 2010)
[34] (Linden, 2006)

III.1.4.3. Skalierbarkeit

Die MarkovShield Gesamtlösung sollte für kleine bis mittlere Setups, mit maximal 100 gleichzeitigen Sessions, ohne grössere Anpassungen an der Architektur lauffähig sein.

Beim Einsatz von externen Komponenten ist zudem darauf zu achten, dass diese in der Lage sind über das oben erwähnte Limit zu skalieren.

III.1.4.4. Deployment

MarkovShield sollte so einfach wie möglich in Betrieb genommen werden können. Ein solch vereinfachtes Deployment hat zudem den Vorteil, dass relativ unkompliziert und schnell eine Demo Applikation zur Verfügung gestellt werden kann.

Aus den zuvor erwähnten Punkten ergibt sich, dass die MarkovShield Komponenten als einzelne Container zur Verfügung gestellt werden sollen. Bevorzugt soll Docker als Container Technologie verwendet werden da Docker am weitesten verbreitet und einfach zu bedienen ist.

III.2. Anforderungsanalyse

Aufgrund der Basisanforderungen wurden in einem ersten Schritt drei verschiedene Varianten ausgearbeitet. Sie unterscheiden sich im Verhalten, mit welchem das System auf einen potentiell gefährlichen Zugriff reagiert. Nachfolgend wird ein kurzer Überblick über diese drei Varianten, sowie deren Vor- und Nachteile gegeben.

III.2.1. Variante 1

Die Variante 1 sieht MarkovShield als ein Teil der Infrastruktur vor und benötigt somit keine Anpassungen der Webapplikation. MarkovShield unterscheidet in dieser Variante nur zwischen den zwei verschiedenen Status OK und FRAUD.

Der Ablauf bei der Feststellung einer Anomalie sieht wie folgend aus:

1. Session auf dem Reverse Proxy über mod_but terminieren
2. Benutzer auf eine Statusseite weiterleiten
3. Kontaktperson der Applikation informieren

Im Vergleich zu den anderen Varianten ergeben sich folgende Vorteile:

- Die Applikation bleibt unverändert
- Das Deployment ist relativ einfach, da keine Abhängigkeit mit der Applikation besteht
- Die Entwicklung von MarkovShield ist vereinfacht, da keine Differenzierung zwischen den Fällen SUSPICIOUS und FRAUD gemacht werden muss
- Es braucht keine Step-Up-Funktionalität

Im Vergleich zu den anderen Varianten ergeben sich folgende Nachteile:

- Es existiert keine Unterscheidung von verschiedenen Verdächtkeitslevels. Daraus folgt auf einem hohen Abstraktionslevel gesehen, dass es wahrscheinlich mehr False-Positives geben wird.
- Durch die höhere Anzahl von False-Positives leidet die Benutzerfreundlichkeit stark.

III.2.2. Variante 2

Auch die Variante 2 sieht MarkovShield als Teil der Infrastruktur vor, benötigt jedoch zusätzliche Daten und deshalb kleinere Anpassungen an der Webapplikation oder eine andere Bezugsquelle der Informationen. MarkovShield unterscheidet in dieser Variante zwischen drei verschiedenen Status OK, SUSPICIOUS und FRAUD

Der weitere Ablauf nach der Feststellung einer Anomalie unterscheidet sich hierbei zwischen dem Status SUSPICIOUS und FRAUD.

Beim Status SUSPICIOUS wird eine zusätzliche Authentifizierung des Benutzers benötigt. MarkovShield übernimmt dabei die Authentifizierung über SMS oder E-Mail, benötigt dafür jedoch die betreffenden zusätzlichen Benutzerdaten. Diese Daten könnten über eine API bei der zu schützenden Webapplikation oder einer anderen zur Verfügung stehenden Komponente in Echtzeit abgefragt werden.

Beim Status FRAUD sieht der Ablauf wie folgt aus:

1. Session auf dem Reverse Proxy über mod_mshield terminieren
2. Benutzer auf eine Statusseite weiterleiten
3. Kontaktperson der Applikation informieren

Im Vergleich zu den anderen Varianten ergeben sich folgende Vorteile:

- Applikation bleibt weitgehend unverändert
- Genauere Unterscheidung der verschiedenen Fällem; dadurch treten weniger False-Positives auf
- Bessere Benutzerführung, da im SUSPICIOUS-Fall noch die Chance besteht, das Vertrauen in die Session durch Eingabe eines korrekten Codes aufrecht zu halten

Im Vergleich zu den anderen Varianten ergeben sich folgende Nachteile:

- Es braucht eine Anpassung an der Applikation oder einer anderen Komponente, welche die benötigten Benutzerdaten zur Verfügung stellt
- Mod_mshield muss sich den Authentifizierungsstatus merken können
- Mod_mshield muss mehrere verschiedene Verhaltensweisen implementieren

III.2.3. Variante 3

Diese Variante bringt eine stärkere Einbeziehung der Applikation mit sich und kann somit nicht mehr als eigentliche Infrastrukturkomponente bezeichnet werden.

Im Gegensatz zu Variante 2 übernimmt in Variante 3 nicht MarkovShield, sondern die Applikation selbst die zusätzliche Authentifizierung des Benutzers. Dadurch können die Benutzerdaten in der Webapplikation selbst gehalten werden. MarkovShield informiert hier lediglich die Webapplikation, wenn ein Benutzer eine zusätzliche Authentifizierungsstufe benötigt.

Neben den Vorteilen von Variante 2 ergibt sich vor allem der Vorteil, dass die Komplexität im mod_mshield sinkt. Zusätzlich dazu müssen keine Benutzerdaten an eine andere Applikation übergeben werden. Stattdessen erhöht sich die Kopplung mit der zu schützenden Applikation und diese muss nun die zusätzliche Authentifizierung übernehmen.

III.2.4. Entscheidung

Nach einer Abwägung der Vor- und Nachteile wurde in enger Absprache mit dem Betreuer Ivan Bütler entschieden die Umsetzung von Variante 2 weiterzuverfolgen, um damit eine Unterscheidung zwischen den verschiedenen Status OK, SUSPICIOUS und FRAUD bieten zu können. Auf eine Anbindung an die Applikation für die zusätzlichen Benutzerdaten, wie zum Beispiel die Telefonnummer, wird während dieser Arbeit aus zeitlichen Gründen verzichtet und anstelle dessen eine statische Webseite angezeigt, welche die Basisfunktionalität aufzeigen soll. Für eine produktive Nutzung müsste deshalb einerseits die Anbindung an einen SMS-Gateway und anderseits der Bezug der Benutzerdaten implementiert werden.

III.2.5. Bedenken bezüglich Step-Up

Während der Analyse der verschiedenen Varianten traten Bedenken an der Wirksamkeit einer Step-Up-Funktionalität zutage. Diese Methode ist stark auf die Zusammenarbeit mit dem Benutzer und seiner Achtsamkeit ausgelegt. Eine solche Interaktion mit dem Benutzer ist zwar oft nicht zu verhindern, jedoch im Idealfall auf ein Minimum zu reduzieren.

Ein Problem dabei ist, dass der Hacker die volle Kontrolle über den Browser des Benutzers hat. Bei der Aufforderung ein zusätzliches Token einzugeben, kann dieser die Aufforderung an den Benutzer weitergeben, welcher den Token daraufhin eingibt.

Das Prinzip mit dem Step-Up besitzt, wie aufgezeigt, mehrere Schwächen. Jedoch ist es sehr einfach verständlich und zeigt es ideal, in welchem Fall eine Reaktion getätigt würde. Das Prinzip eignet sich deshalb nur für momentan schwach geschützte Webapplikationen.

Die Reaktion von MarkovShield auf ein negatives Analyseresultat kann in Zukunft ohne grössere Probleme ausgetauscht werden. Für einen produktiven Einsatz in einer E-Banking Umgebung wäre dies zum Beispiel Transaktionen für einen bestimmten Zeitraum zu verhindern oder für Transaktionen dieses Benutzers in der nächsten Zeit zusätzliche Analysen anzufordern. Diese Reaktionen benötigen jedoch eine direkte Integration in die Webapplikation.

III.3. System Architektur

In einem Projekt mit einer hohen Systemkomplexität ist die System Architektur sehr zentral, weshalb diese Überlegungen in einem eigenen Kapitel dokumentiert sind.

Bereits sehr früh haben sich die wichtigsten Architekturkomponenten herauskristallisiert. Diese haben sich im Laufe der Arbeit zugegebenermassen noch geändert, die grundsätzlichen Überlegungen sind jedoch immer noch gültig.

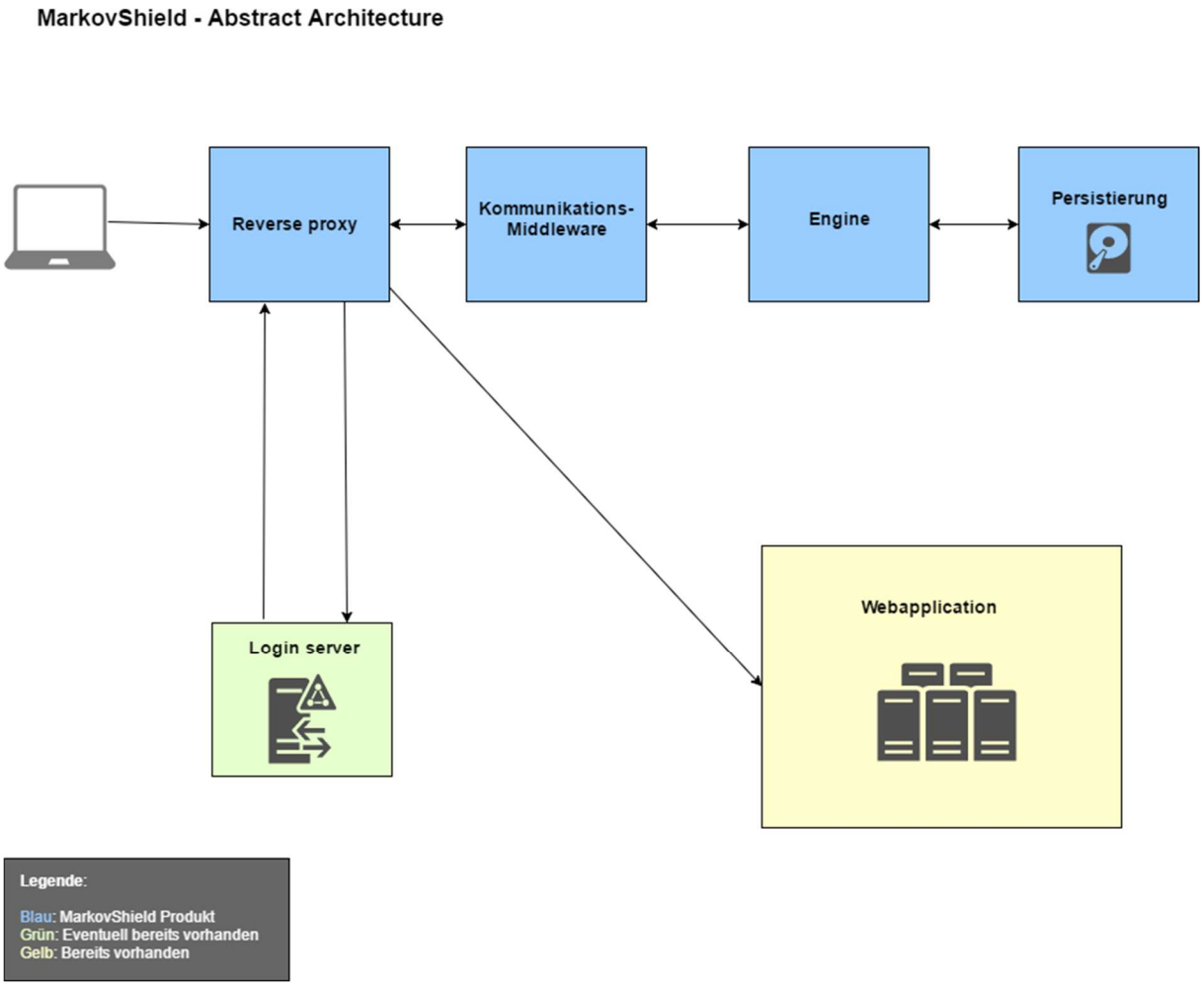


Abbildung 7 Abstrakte Infrastruktur (Erster Entwurf)

III.3.1. Reverse Proxy

Wie bereits im Kapitel II.3.3 Informationsbeschaffung erwähnt, setzt die Reverse Proxy Komponente auf dem Apache Modul MOD_BUT auf. Dieses umfasst bereits folgende Funktionalitäten:

- Sessionhandling (Aufbau und Terminierung, Cookiehandling)
- Pre-Authentication
- Redirection
- Support für verschiedene Authentifizierungs-Levels

Das Modul benötigt nun zusätzlich die folgenden zusätzlichen Funktionalitäten:

- Anbindung an die Kommunikations-Middleware
- Extraktion der Session-Daten (Session, User) bei einem Login
- Extraktion der Meta-Daten eines Requests (Session, URL, Timestamp)
- Blockierung eines Requests, wenn dieser von der Engine überprüft werden muss
- Konfigurationsmöglichkeit für die URL Risk Levels

III.3.2. Kommunikations-Middleware

Die Kommunikations-Middleware ist hauptsächlich für die Kommunikation zwischen mod_mshield und der Engine zuständig. Über diese werden sowohl die Meta-Daten der Requests als auch die Session-Daten beim Login an die Engine zugestellt. Zudem wird auch das Resultat der Validierung über die Kommunikations-Middleware zurück an mod_mshield kommuniziert.

III.3.3. Engine

Die in diesem Abschnitt genannte Komponente Engine ist eigentlich für zwei verschiedene Aufgaben zuständig. Zum einen übernimmt sie die komplette Analyse der aggregierten Clickstream-Daten und zum anderen das regelmässige Berechnen der Models.

Für die Analyse der Clickstream-Daten ist dabei die wichtigste Messgrösse die Zeit zwischen dem Aufruf und der Antwort, da diese sich direkt auf die Verzögerung des User-Requests an die Applikation auswirkt. Hierbei ist es wichtig, eine konstant minimale Verzögerung zu realisieren und nicht nur die durchschnittliche Verzögerung klein zu halten, um damit ein möglichst gutes Benutzererlebnis bieten zu können.

Bei der Berechnung der Models ist hingegen vor allem die durchschnittliche Berechnungszeit zu minimieren, da eine kleine Verzögerung den Endbenutzer nicht direkt beeinflusst.

Bei der Realisierung von Models sollte zudem darauf geachtet werden, vor allem die Auswertung eines Clickstreams mit dem Model zu optimieren, da diese viel zeitkritischer ist.

III.3.4. Persistierung

Die Aufgabe der Persistierungs-Komponente liegt darin, die gesammelten Session-Informationen aufzubewahren und für die Berechnung der Models zugänglich zu machen. Ebenfalls sollen die berechneten Models persistiert werden, damit bei einem allfälligen Ausfall des Systems keine neuen Berechnungen benötigt werden und somit eine kürzere Recovery Time erreicht werden kann.

III.3.5. Login-Server

Der Login-Server ist eine bereits existierende Komponente, welche allenfalls noch ausgebaut werden muss. Sie ist für das eigentliche Login der Benutzer zuständig und wird von MOD_BUT beziehungsweise neu mod_mshield angesprochen, um die Pre-Authentication Funktionalität anbieten zu können. Der Login-Server muss vor einem produktiven Einsatz ausgetauscht werden, da er in der momentanen Implementation stark vereinfacht wurde.

Genauere Informationen über die Kommunikation von mod_mshield mit dem Login-Server können dem Kapitel III.5.2 Login Prozess entnommen werden.

III.3.6. Webapplikation

Die Webapplikation ist die zu schützende Applikation, welche im Hauptinteresse des Benutzers steht.

III.3.7. Zusätzliche Funktionalität

Es gibt mehrere Funktionalitäten, welche sehr relevant sind, jedoch in den ersten Versionen der System-Architektur noch nicht definitiv einer Komponente zugeordnet wurden.

- Die einzelnen Clicks, welche von mod_mshield an die Kommunikations-Middleware übergeben werden, müssen zu einem Clickstream aggregiert werden.
- Die Zusammenführung der Session-Informationen mit dem Clickstream
- Die Zusammenführung der richtigen Models mit dem Clickstream

Für alle diese Funktionalitäten war in den ersten Entwürfen entweder die Kommunikations-Middleware oder die Engine selbst zuständig.

Aufgrund des fehlenden Wissens im Bereich Stream Processing war es am Anfang noch nicht möglich definitiv festzulegen, in welcher Komponente diese Funktionalitäten am besten realisiert werden.

III.4. Evaluation der Komponenten

Während bei einigen Komponenten bereits vom Anfang an klar war, welche Software dafür eingesetzt wird, mussten aufgrund der ausgearbeiteten Anforderungen die anderen Softwarekomponenten erst noch gesucht und die Architektur gegebenenfalls entsprechend angepasst werden. Es geht hierbei vor allem um die drei Komponenten Kommunikations-Middleware, Engine sowie Persistierung. Zusätzlich muss beachtet werden, dass diese drei Komponenten eine gute Integration ineinander besitzen.

Es wurden zum Anfang hin verschiedenste Kombinationen von Message-Queues als Komponente Kommunikations-Middleware und Stream Processing Engines gesucht, um einen groben Überblick über die vorhandenen Softwarelösungen zu haben.

III.4.1. Engine

III.4.1.1. Apache Spark Streaming

Apache Spark Streaming überzeugt durch seine API in verschiedensten Sprachen wie Scala, Java, Python und R. Die Apache Spark Streaming API ist zudem sehr stark an die API von Apache Spark angelehnt, was insbesondere dann von Vorteil, wenn bereits Apache Spark Applikation im selben Umfeld existieren. Es ist somit möglich Code, welcher für Apache Spark entwickelt wurde, mit kleineren Anpassungen zu übernehmen. Ebenfalls als Pluspunkt festzuhalten ist die sehr grosse Community, welche sich über die Jahre entwickelt hat. Entsprechend wird es im Vergleich zu anderen Lösungen einfacher sein, Hilfe bei komplexeren Problemstellungen zu bekommen.

Apache Spark Streaming basiert auf sogenannten Micro-Batches, es sammelt also Streaming-Daten über eine kurze Zeit und löst dann den Job aus. Dies ist auch der Grund für die sehr ähnliche API. Leider wird dadurch die erreichte Latenz stark erhöht. Laut Haupt-Maintainer Databricks ist die zusätzliche Latenz im Normalfall kein Problem[35].

> *In terms of latency, Spark Streaming can achieve latencies as low as a few hundred milliseconds. Developers sometimes ask whether the micro-batching inherently adds too much latency. In practice,batching latency is only a small component of end-to-end pipeline latency. … Developers sometimes ask whether the micro-batching inherently adds too much latency. In practice,batching latency is only a small component of end-to-end pipeline latency.* [36]

[35] (Das, Zaharia, & Wendell, 2015)
[36] (Das, Zaharia, & Wendell, 2015)

Es finden sich jedoch mehrere Erfahrungsberichte, dass Latenzen unter einer Sekunde sehr schwer zu erreichen sind.[37][38] Selbst die erwähnten mehreren 100 Millisekunden sind für diesen bestimmten Use-Case zu viel, da eine möglichst zeitnahe Antwort benötigt wird.

Zudem beschreibt das ursprüngliche Paper, auf welchem Spark Streaming basiert eine Latency von ungefähr einer Sekunde.

> *For both applications, we measured the maximum throughput achievable on different-sized clusters with an end-to-end latency target of either 1 or 2 seconds. By end-to-end latency, we mean the total time between when a record enters the system and when it appears in a result, including the time it waits for its batch to start.*[39]

Es existiert jedoch eine Erweiterung von Apache Spark Streaming namens Drizzle, welche sich auf Low-Latency spezialisiert hat. Mehr Informationen dazu finden sich im nächsten Abschnitt.

III.4.1.2. Drizzle

> *Drizzle is a low latency execution engine for Apache Spark that is targeted at stream processing*[40]

Drizzle setzt auf Apache Spark auf, benutzt jedoch ein neues Modell zum Aufteilen der Tasks in Batches sowie einen neu implementierten Scheduler. Dadurch können die Mikro-Batches viel kleiner gemacht werden und damit auch die Latenz stark verringert werden.

[37] (Leonhardi, 2016)
[38] (Farivar & Knusbaum, 2016)
[39] (Zaharia, Das, Li, Shenker, & Ion, 2012)
[40] (Venkataraman, Drizzle: Low Latency Execution for Apache Spark, 2015)

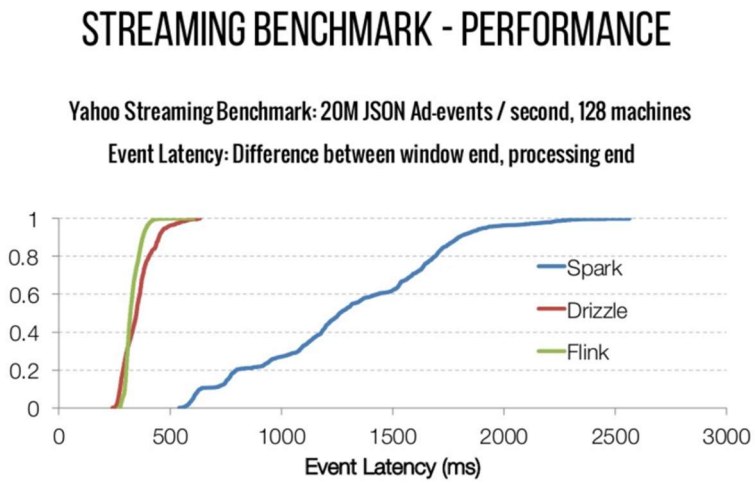


Abbildung 8 Latenzvergleich Flink, Drizzle, Spark Streaming[41]

Drizzle ist leider momentan nur ein Research Prototyp von AMPLab, von UC Berkeley, woher auch Apache Spark ursprünglich stammt. Es wird über eine eventuelle Integration von Drizzle in Apache Spark diskutiert.

III.4.1.3. Apache Flink

Apache Flink ist ein Stream Processing Framework, welches auf sogenanntes Record-at-at-time Streaming setzt. Dabei wird im Gegensatz zum Batch-Processing jeder Record einzeln behandelt. Dadurch lassen sich viel tiefere Latenzzeiten erreichen. Für eine genauere Unterscheidung des Micro-Batch-Models und dem Record-at-a-time Model empfiehlt sich das Kapitel 2 von «Drizzle: Fast and Adaptable Stream Processing at Scale»[42]

Es bietet neben einer Java-API auch eine Scala-API an und hat sich dem Gedanken «Streaming-first» untergeordnet.

[41] (Venkataraman, Drizzle—Low Latency Execution for Apache Spark: Spark Summit East talk by Shivaram Venkataraman, 2017)

[42] (Venkataraman, et al.)

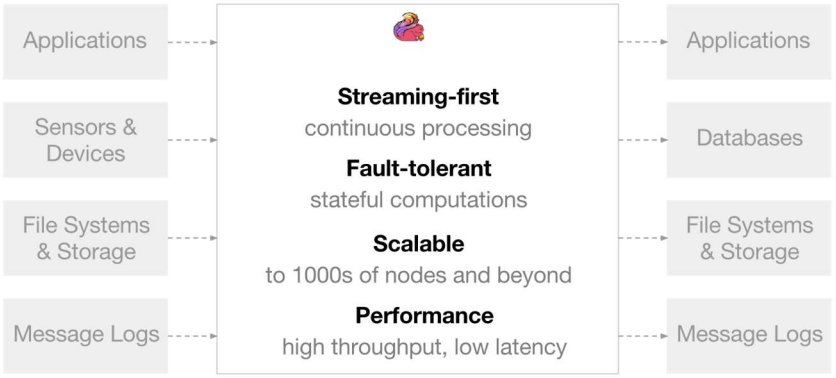


Abbildung 9 Apache Flink

Daneben ist es natürlich gut skalierbar und besitzt eine gute Fehlertoleranz und Performance.

Neben der Streaming API bietet Flink auch eine Batch-Processing API an. Im Gegensatz zu Apache Spark behandelt Apache Flink einen Batch jedoch als Spezialfall eines Streams.[43]

Mit Apache Flink ist es möglich eine Exactly-Once Garantie zu erreichen, dies könnte in Zukunft für eine Sicherheitsapplikation von grossem Interesse sein.

Flink besitzt verschiedene Deployment-Arten, welche es sehr flexibel einsetzbar machen:

- Local
- Cluster Standalone
- Cluster YARN
- Cluster MESOS
- Cloud EC2
- Cloud GCE

III.4.1.4. Apache Storm

Wie Apache Flink setzt auch Apache Storm auf «Record-at-at-time Streaming» und ist somit in der Lage eine tiefe Latenz zu gewährleisten. Sehr viele bekannte Firmen setzen Apache Storm ein, was auch zu einer entsprechend grossen Community führt.

Im Gegensatz zu Apache Flink unterstützt Storm nur At-least-once Garantien und kann deshalb durch eine leicht andere Architektur sogar noch schnellere Zeiten erreichen.

[43] (Apache Flink)

III.4.1.5. Apache Samza

Apache Samza gleicht Apache Storm auf den ersten Blick, lediglich die interne Architektur der beiden Frameworks ist unterschiedlich. Trotzdem gibt es verschiedene Unterschiede und der grösste ist, dass Samza über ein Statemanagement-System verfügt. Dies erlaubt es gewisse Daten auf der gleichen Maschine abzulegen und somit schneller wieder abzufragen.

III.4.1.6. Apache Gearpump

Apache Gearpump ist eine weitere Streaming Engine, welche sich auf das Bearbeiten von einzelnen Records spezialisiert hat. Es verspricht einen extrem hohen Durchsatz und trotzdem eine tiefe Latenz.

> *Per initial benchmarks we are able to process 18 million messages per second (message length is 100 bytes) with a 8ms latency on a 4-node cluster.*[44]

Es basiert auf dem Akka Actor Model, wodurch es zudem auch sehr hohen Anforderungen an die Stabilität genügt.

Wie auch Apache Flink bietet Apache Gearpump eine Exactly-once Garantie an.

Apache Gearpump kennt wie auch Flink verschiedene Deployment Modes:

- Local Mode
- Cluster Standalone Mode
- YARN Mode
- Docker Mode

III.4.1.7. Entscheidung

Nach einer langen und genauen Evaluation wurde Apache Flink als Stream Processing Framework ausgewählt. Apache Flink bietet eine sehr gute Möglichkeit die Latenz tief zu halten und bietet zudem verschiedene Deployment Modes an.

III.4.2. Kommunikations-Middleware

Mit der Wahl von Apache Flink hat sich auch die Auswahl der Kommunikations-Middleware auf Apache Kafka und RabbitMQ reduziert. Diese zwei Middlewares sind von Apache Flink unterstützt und es finden sich Ressourcen dafür auf deren Webseite. Ein Vorteil von Apache Kafka ist die sehr hohe Verbreitung im Stream Processing Umfeld und damit die sehr gute

[44] (Apache Gearpump)

Integration auch in andere Stream Processing Frameworks. Ein weiterer Vorteil, welcher im Laufe der Evaluation entdeckt wurde, ist, dass sich Apache Kafka auch als längerfristige Persistierungskomponente eignet und damit die Systemkomplexität verringert werden kann. Aus diesem Grund wurde der Einsatz von Apache Kafka präferiert.

III.4.3. Persistierung

Da momentan keine weiterführenden Analysen der gesammelten Daten vorgesehen sind, wurde auf eine eigene Persistierungskomponente verzichtet und die Persistierung direkt in der Middleware durchgeführt. Dafür müssen lediglich die einzelnen Topics von Apache Kafka entsprechend konfiguriert werden.[45]

A key aspect of the Kafka architecture is that it handles persistence well. A Kafka broker can store many TBs of data.[46]

Kafka's performance is effectively constant with respect to data size so storing data for a long time is not a problem.[47]

Wenn zu einem späteren Zeitpunkt eine zusätzliche Komponente benötigt wird, lässt sich diese sehr einfach an Apache Kafka anbinden und kann alle gesammelten Daten nachträglich beziehen. Der Grund dafür liegt in der Architektur von Apache Kafka. Abstrakt betrachtet kann Apache Kafka als ein strukturiertes commit-Log angesehen werden, welches die einzelnen Updates beinhaltet.[48] Dabei besitzt jeder Kafka Consumer seine eigene Position im commit-Log und kann somit jederzeit mit dem Lesen an seinem letzten bekannten Ort fortfahren.

III.4.4. Zusätzliche Funktionalität

Die im Kapitel III.3.7 Zusätzliche Funktionalität erwähnten Funktionalitäten wurden im Verlaufe der Arbeit ebenfalls der Kommunikations-Middleware zugeordnet. Dies hat den Grund, dass in Apache Flink zum momentanen Zeitpunkt ein Join von zwei Streams nur über ein Window möglich wäre und auch erst zum Ende des Windows berechnet werden könnte. Im MarkovShield ist es jedoch notwendig eine sofortige Anreicherung der Clicks mit Session- und User-Daten zu berechnen. Eine Erweiterung von Flink für diesen Use-Case ist jedoch bereits angedacht und in Diskussion.[49] Apache Kafka Streams unterstützt die benötigte Anreicherung der Daten und übernimmt diese in der finalen Version von MarkovShield.

---

[45] Siehe auch Kapitel III.6.2 Persistenz und III.8.1.11 Kafka Topic Creator
[46] (Kreps, 2015)
[47] (Apache Kafka)
[48] (Kreps, 2015)
[49] (Del Monte, Krettek, & al)

III.5. System-Sequenzdiagramme

In diesem Kapitel geht es hauptsächlich darum, wie die Abläufe unter Betrachtung der jeweiligen Situation zwischen den verschiedenen Systemen ablaufen. Die hier abgebildeten System-Sequenzdiagramme (SSDs) stellen die effektiv umgesetzten Abläufe dar.

Dabei wurde ein Detailierungsgrad gewählt, welcher für diesen Kontext das Relevante aufzeigt und das weniger Wichtige der Übersichtlichkeit halber weglässt. So zum Beispiel finden unteranderem in Realität noch die folgend aufgelisteten Abläufe statt. Für eine genauere Erläuterung dieser sei jedoch auf die MOD_BUT Dokumentation verwiesen.

- *Cookie Try*: Die bestehende MOD_BUT Funktionalität «Cookie Try» prüft, ob der Client Cookies im Browser erlaubt hat. Ein Client muss Cookies erlaubt haben, damit er mit dem Apache Modul kommunizieren kann. Falls der Client dies nicht hat, wird er auf eine Errorpage weitergeleitet und kann nicht via Reverse Proxy auf die Webapplikation zugreifen.
- *Cookie Store*: Mittels des Cookies Stores ist mod_mshield in der Lage, Cookies vom Backend daran zu hindern, zum Client gesendet zu werden. So kann zum Beispiel ein Cookie mit einer jsessionid vom Backend zum Client in mod_mshield zurückgehalten und temporär im Cookie Store zwischengespeichert werden. Dabei speichert sich mod_mshield, welche Cookies zu einer Session gehören. Wenn nun der Client erneut einen Request auf das Backend tätigt, wird dieses Cookie dem Request auf dem Reverse Proxy wieder angehängt bevor er an das Backend weitergeleitet wird.

III.5.1. Session Initialisierung

Zunächst sollte betrachtet werden, wie eine Session erstellt wird, wenn ein Client ohne gültiges Cookie auf die Webapplikation zugreift. Dies findet normalerweise in zwei Szenarien statt:

- Ein Client benutzt diese Webapplikation zum ersten Mal
- Der Client hat die Webapplikation über eine längere Zeit nicht besucht und die Session ist abgelaufen

Ein abstrahierter Session Initialisierungsablauf ist im Diagramm 1 Session Initialisierung beschrieben. Der Übersichtlichkeit halber wurde weggelassen, dass beim ersten Zugriff nicht direkt eine Antwort zurückkommt, sondern dass dort in der effektiven Umsetzung zuerst noch der Cookie Try Vorgang stattfindet. Nach diesem Vorgang wird der Client auf die ursprünglich zugegriffene URL weitergeleitet und somit die Antwort der Webapplikation erhalten.

Im ersten Abschnitt («First time visit») kann erkannt werden, wie mod_mshield als erstes intern eine Session anlegt. Anschliessend wird der eigentliche Request auf die Webapplikation weitergeleitet und die Antwort schliesslich mit dem Cookie A dem Client zurückgeschickt. Im «Loop»-Abschnitt wird dargestellt, wie der Client im Anschluss an den «First time visit»-Ablauf das Cookie A immer mitschickt. Durch die im Cookie A enthaltene SessionID kann mod_mshield wiederum Rückschluss auf die intern abgelegte Session erhalten und erkennt so, dass der Client in der kürzlich vergangenen Zeit auf die Webapplikation zugegriffen hat. Der «Loop»-Anschnitt ist so lange gültig, bis der Client auf eine geschützte URL zugreift. In einem solchen Fall wird ein Login Prozess gestartet, welcher im folgenden Abschnitt III.5.2 genauer erläutert wird.

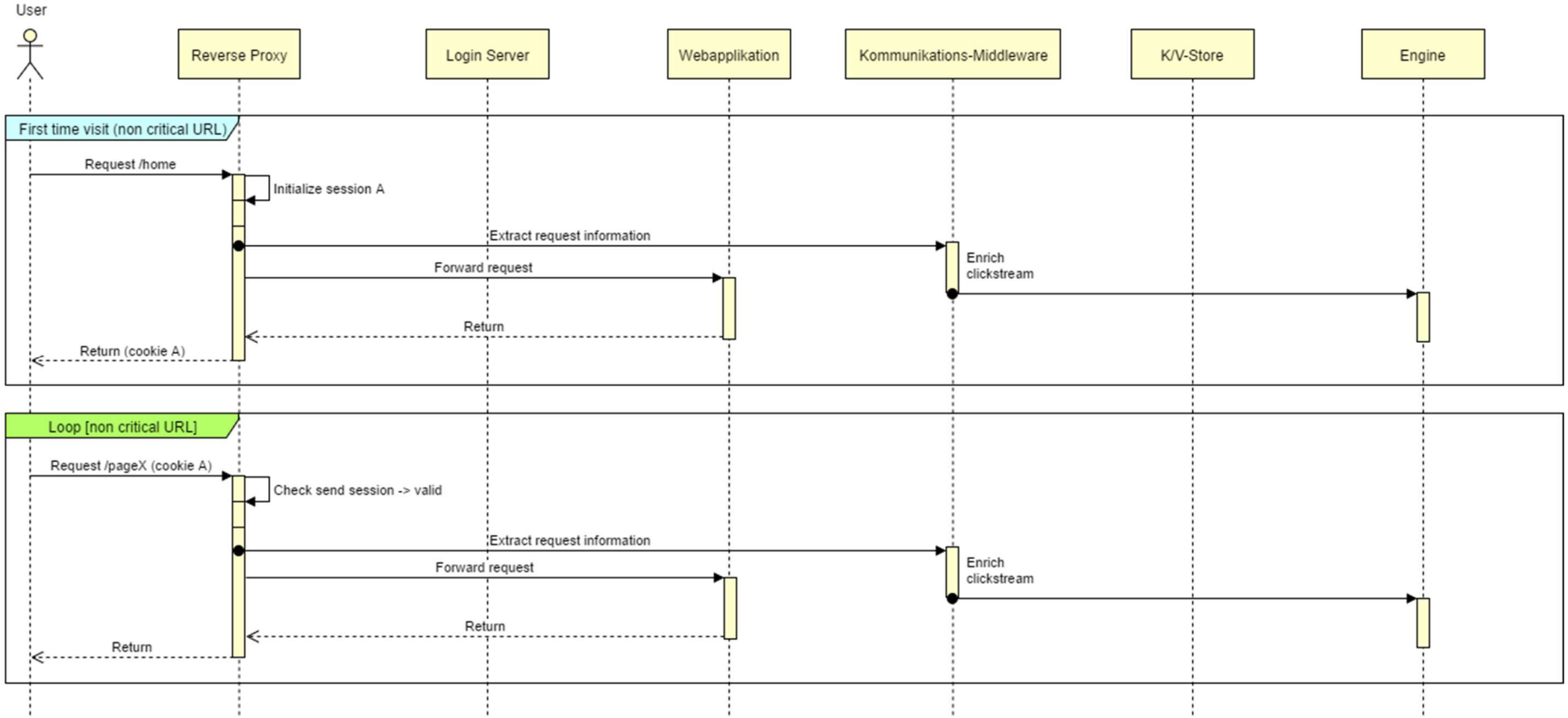


Diagramm 1 Session Initialisierung

III.5.2. Login Prozess

Als nächstes wird der Login Prozess betrachtet, da dieser eine gewisse Komplexität aufweist und zugleich von grosser Bedeutung für die nachfolgenden Abläufe ist.

Im Diagramm 2 Login Prozess wird aufgezeigt, wie der Login Prozess abläuft, wenn ein Client auf eine durch MarkovShield geschützte URL zugreift, auf welcher die Benutzer Authentifikation aktiviert ist. Der folgend aufgezeigte Ablauf setzt darauf auf, dass der Client bereits eine MarkovShield Session besitzt. Diese hatte jedoch bis anhin eine Authentication Strength von 0, was bedeutet, dass der Client noch nicht eingeloggt ist.

Wenn ein Client auf eine Webapplikation URL zugreift, auf welcher die MarkovShield Benutzer Authentifikation aktiviert ist, muss seine Session Authentication Strength gleich oder grösser der jeweilig verlangten Strength sein. Sollte sie zu klein sein, wird der Client auf den entsprechend konfiguriert Login Server weitergeleitet, bei welchem er sich dann authentifizieren muss.

Ist die Authentifikation erfolgreich, schickt der Login Server als Cookie eine entsprechende Information an mod_mshield. In mod_mshield wird dann eine neue Session initialisiert und die vorhergehende invalidiert. Anschliessend wird dem Client ein neues Cookie (Cookie B) mit der SessionID der neuen Session als Value mitgeschickt und der Client wird auf die ursprünglich aufgerufene URL weitergeleitet, sofern der Login Server in seinem Login Status Cookie mod_mshield nicht anderes vorgesehen hat. Wenn der Client nun weitere URLs aufruft, wird vom Browser her nur noch das Cookie B mitgeschickt.

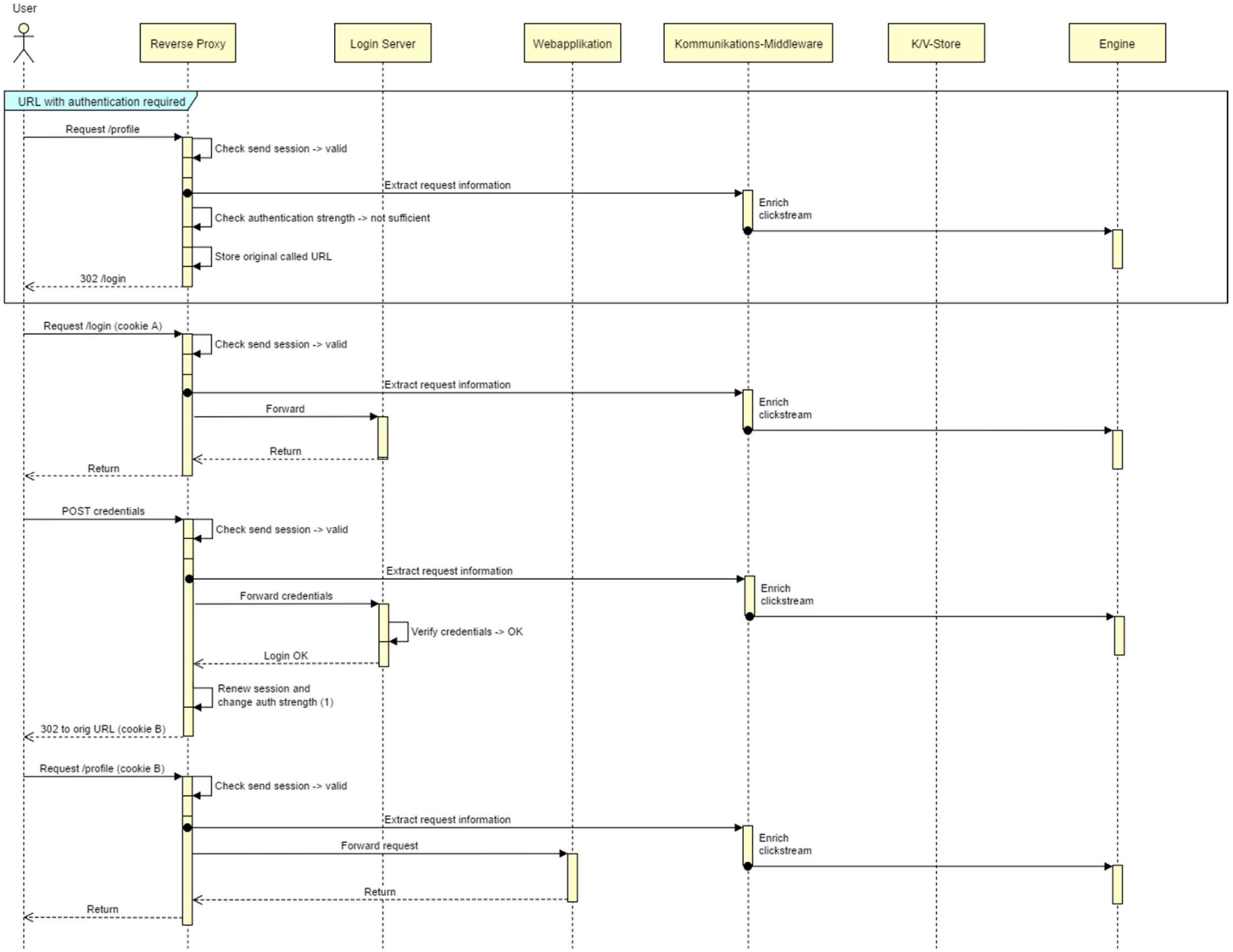


Diagramm 2 Login Prozess

III.5.3. UUID

Wie unter III.5.1 Session Initialisierung erläutert, wird jedem Client mit dem ersten Request via Reverse Proxy auf die Webapplikation eine Session zugeteilt. Diese Session ist mittels einer SessionID (192 Bit Entropie) eindeutig identifizierbar, welche dem Client als Cookie Value mitgeschickt wird. Zugleich werden Session Informationen zusammen mit der SessionID im Shared Memory des mod_mshield Apache Modules abgelegt. Beim Login eines Clients, wie unter III.5.2 Login Prozess beschrieben, wird eine neue Session erstellt, eine neue SessionID generiert und die alte Session gleichzeitig invalidiert.

Eine Differenzierung der zwei SessionIDs ist aus Sicherheitstechnischen Gründen unumgänglich, obwohl beide Sessions zur gleichen logischen Session eines Benutzers gehören. Um diese Zuordnung zu realisieren, wird eine sogenannte UUID eingeführt. Die UUID wird grundsätzlich bei der Session Initialisierung generiert und der Session hinterlegt. Bei einem Step-Up wird die UUID in die neue Session übernommen. Die UUID bleibt so lange bestehen, bis sich der Benutzer ausloggt oder die Session aufgrund eines Timeouts ungültig wird. Mit Hilfe der folgenden zwei Diagrammen wird dieser Vorgang genauer aufgezeigt. Diagramm 4 UUID Teil 2 ist als nahtlose Fortsetzung von Diagramm 3 UUID Teil 1 zu sehen.

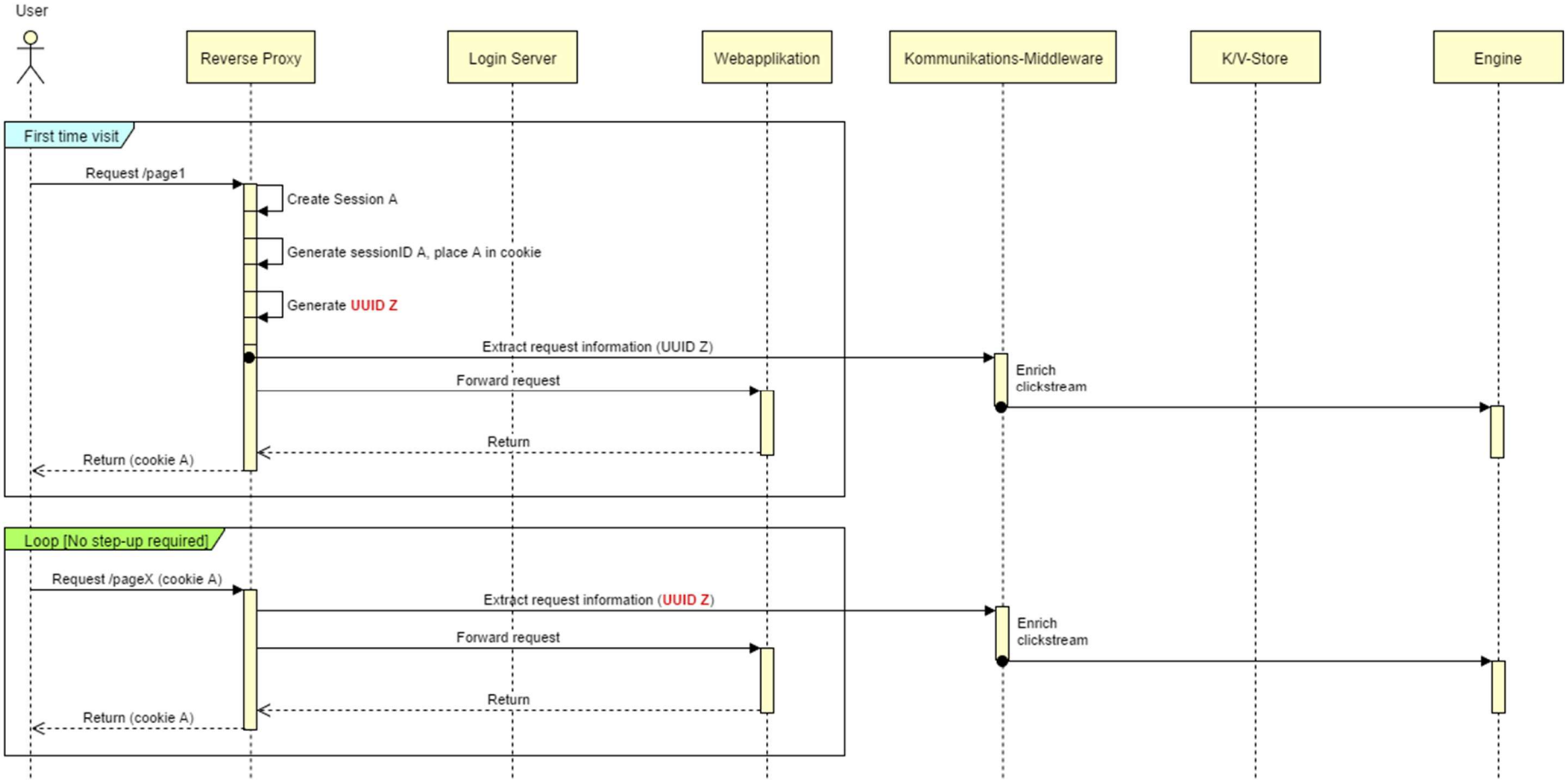


Diagramm 3 UUID Teil 1

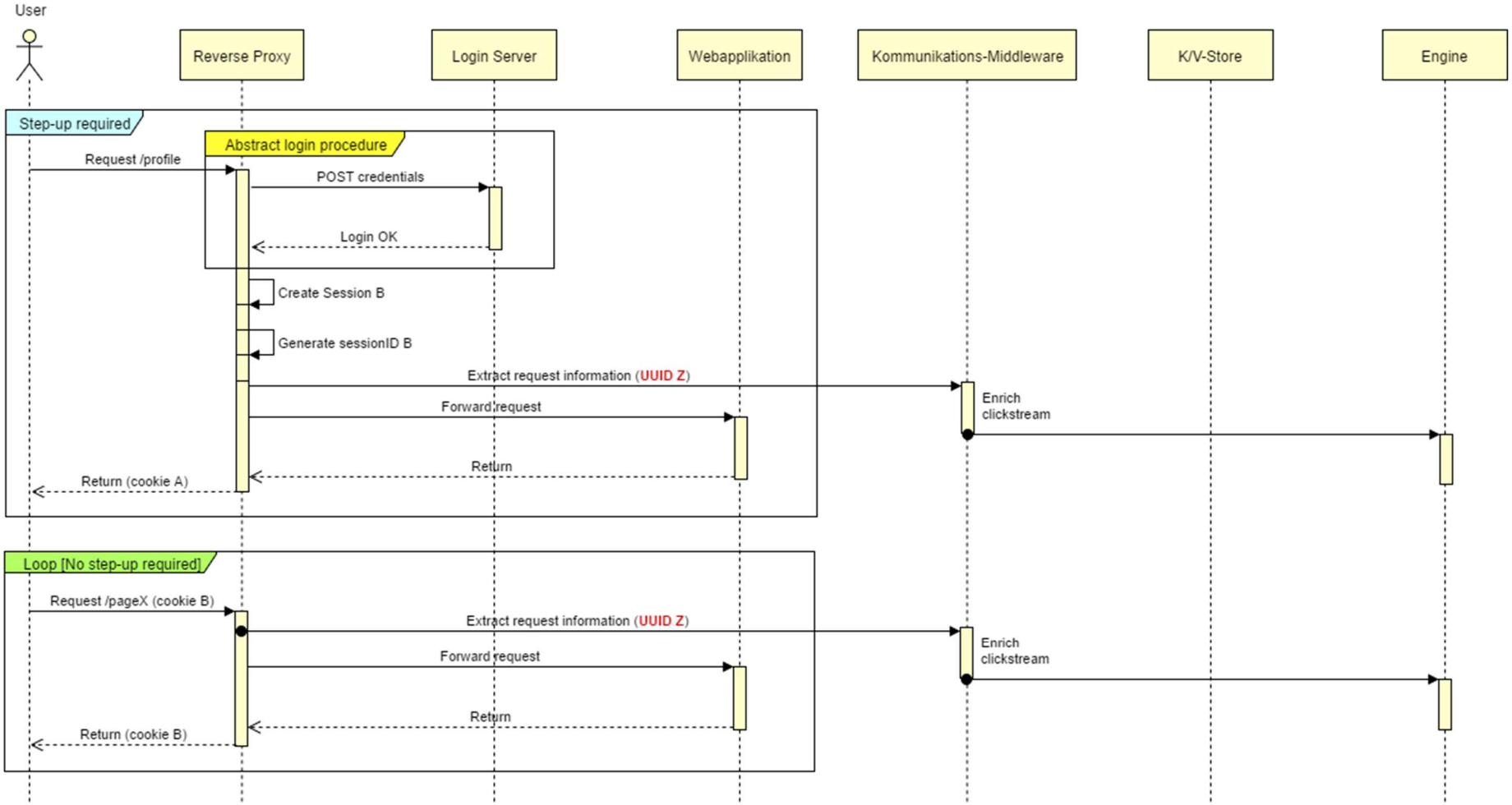


Diagramm 4 UUID Teil 2

III.5.4. OK Case

In diesem Kapitel geht es darum aufzuzeigen, wie die Kommunikation und der Ablauf zwischen den MarkovShield Komponenten im Normalfall aussieht. Dies bedeutet, dass von der Engine kein Verdacht auf ein FRAUD oder auch keine eindeutige Erkennung eines FRAUDs zustande kam. Dies ist der am häufigste auftretende Fall, sobald eine Analyse benötigt wird.

Im «non critical URL»-Teil des Diagrammes ist zu erkennen, dass bei einem Request auf eine nicht kritische URL lediglich Request Informationen in die Kommunikations-Middleware übertragen werden, jedoch nicht auf ein Analyseergebnis der Engine gewartet wird.

Die Unterscheidung von «nicht kritischen» und «kritischen» URLs findet über ein URL spezifisches Risk Level statt. Dieses wird mit einem konfigurieren Grenzwert verglichen. Dieser Grenzwert bestimmt, ab welchem Risk Level ein Request bewertet werden muss. Entsprechend bestimmt mod_mshield darüber auch, ob auf ein Resultat gewartet werden muss, bevor der Client Request an die Webapplikation weitergeleitet darf.

Ruft der Client mit einem Request eine URL auf, welche ein Risk Level über dem eingestellten Grenzwert hat, so wird der Request auf dem Reverse Proxy zu lange zurückgehalten, bis ein Resultat von der Engine über den Key-/Value-Store empfangen wurde. Ist die Session mit OK bewertet worden, wird der Request nun an die Webapplikation weitergeleitet.

Sollte aufgrund eines Fehlers einer Komponente oder aufgrund des Erreichens eines frei konfigurierbaren Timeouts kein Resultat vorliegen, so wird der Request nicht an die Webapplikation weitergeleitet. Stattdessen wird der Benutzer auf eine Error Seite weitergeleitet. Dies stellt sicher, dass die Webapplikation ausschliesslich erreichbar ist, wenn die MarkovShield Engine die Session als ungefährlich bewertet. Die Verbindung zum Key-/Value-Store wird absichtlich vor dem extrahieren der Request Informationen in die Kommunikations-Middleware aufgebaut, um allfällige Timing Probleme präventiv zu eliminieren.

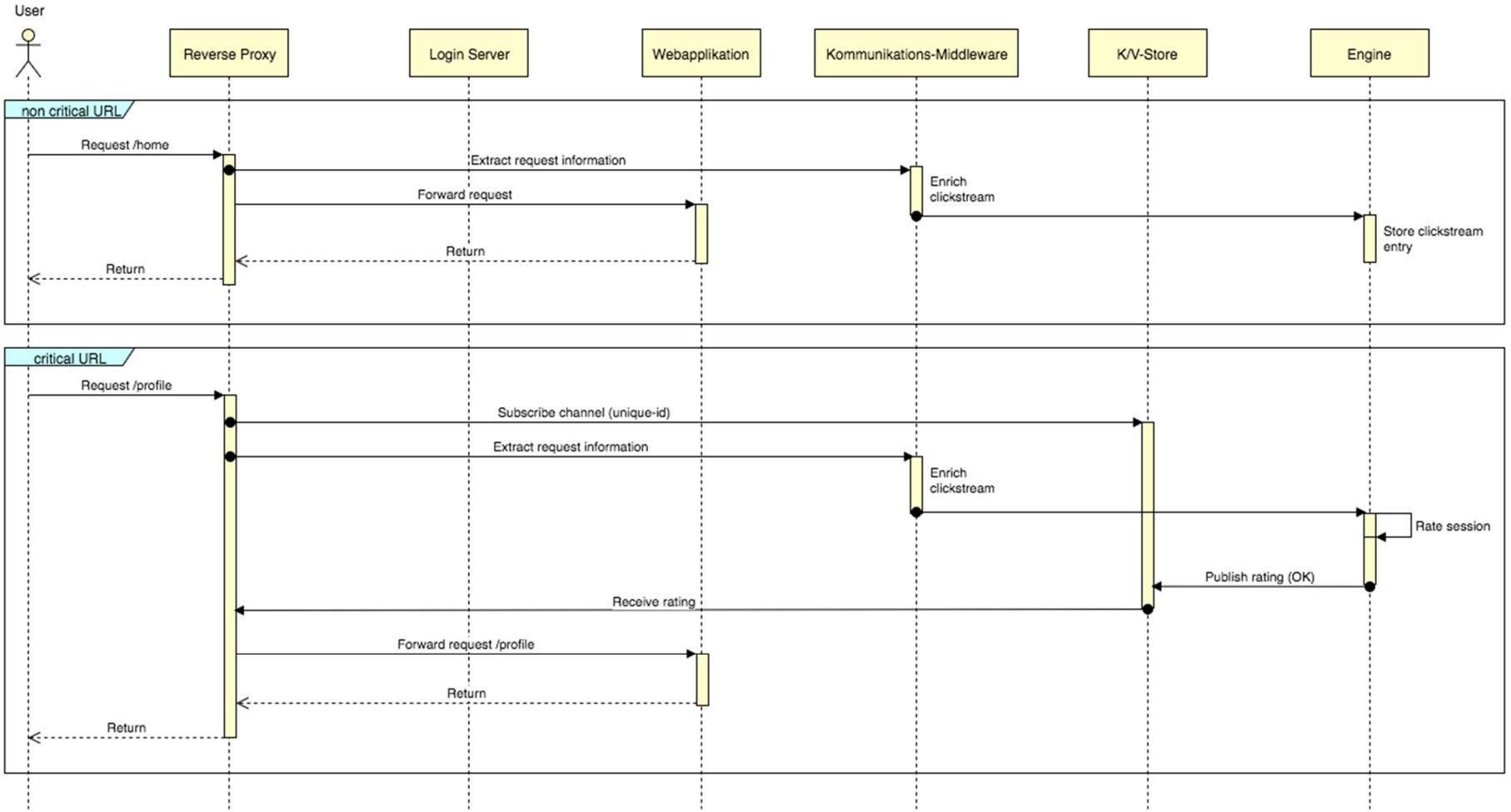


Diagramm 5 OK Case

III.5.5. SUSPICIOUS Case

Im SUSPICIOUS Case SSD geht es darum zu erläutern, wie sich MarkovShield verhält und kommuniziert, sollte von der Engine ein Verdacht auf einen FRAUD erkannt werden.

Innerhalb des «critical URL»-Bereiches im Diagramm 6 SUSPICIOUS Teil 1 ist zu erkennen, wie mod_mshield eine Verbindung zum Key-/Value-Store herstellt und die Request Informationen an die Kommunikations-Middleware sendet. Dieser Teil ist soweit identisch mit dem unter III.5.4 OK Case beschrieben Fall. Da die Session von der Engine neu als möglicherweise vorhandener Betrug erkannt wird, gibt die Engine via Key-/Value-Store SUSPICIOUS als Resultat zurück. Mod_mshield wiederum befiehlt aufgrund der damit nicht mehr ausreichenden «Authentication Strength» der Session ein Step-Up und leitet den Client somit an den Login Server 2 um. Wichtig zu vermerken ist hier, dass der eigentliche Request noch nicht an die Webapplikation weitergeschickt wurde, da noch nicht eindeutig ist, ob es sich mit grosser Wahrscheinlichkeit wirklich um den Benutzer handelt oder ob seine Login Daten gerade missbraucht werden.

Es ist vorgesehen, dass sich der Benutzer auf dem Login Server 2 durch eine erweiterte Authentifizierung stärker authentifizieren kann. Im Bereich «Step-Up» ist der schematische Ablauf aufgezeigt, wie sich der Benutzer durch eine zusätzliche Eingabe eines Tokens über einen zweiten Weg authentifiziert. Nachdem mod_mshield vom Login Server 2 eine positive Antwort erhalten hat, wird entsprechend die «Authentication Strength» der Session hochgestuft und der Benutzer auf die ursprüngliche URL weitergeleitet.

Wie sich im zweiten Teil des Diagramm 7 SUSPICIOUS Teil 2 entnehmen lässt, greift nun der Client erneut auf die kritische URL zu und der Request wird erneut von der Engine als SUSPICIOUS eingestuft. Da allerdings innerhalb der mod_mshield Session die «Authentication Strength» bereits genügend ist, wird der Request zur Webapplikation weitergeleitet.

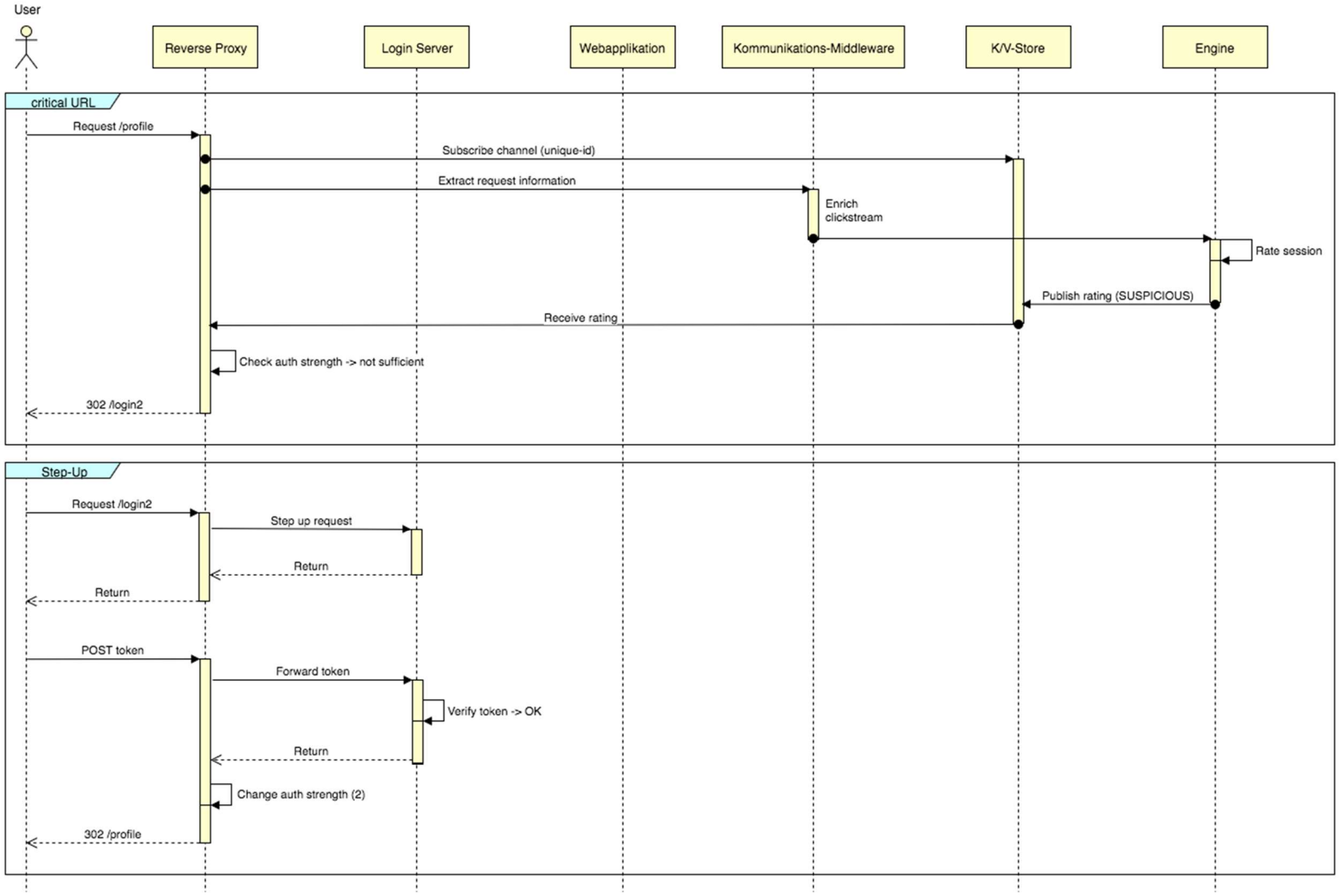


Diagramm 6 SUSPICIOUS Teil 1

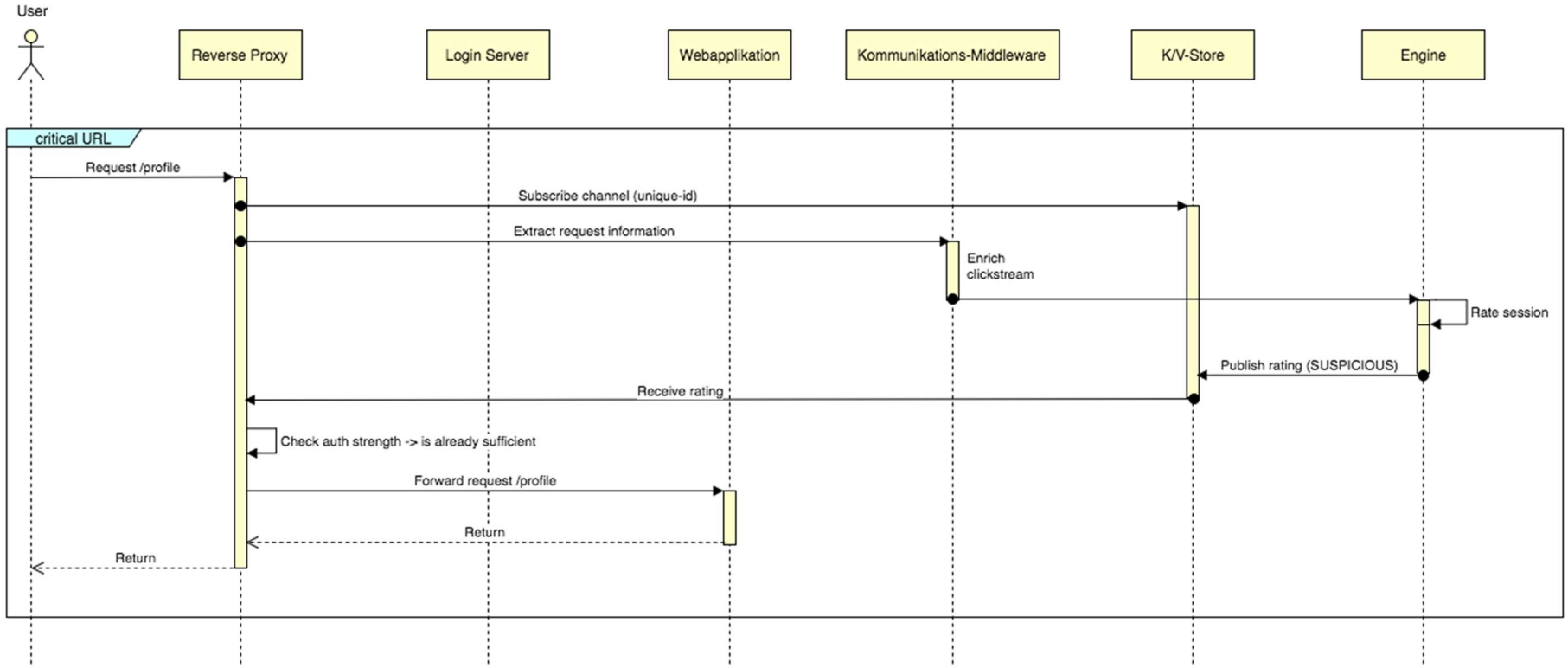


Diagramm 7 SUSPICIOUS Teil 2

III.5.6. FRAUD Case

Ergänzend zu den bereits aufgezeigten Fällen gibt es den Fall, dass eine Session von der MarkovShield Engine als FRAUD eingestuft wird.

Wird im mod_mshield Apache Modul von der Engine das Resultat FRAUD empfangen, so wird der wartende Request nicht an die Webapplikation weitergeleitet. Der Benutzer wird in diesem Fall auf eine entsprechende Error Seite weitergeleitet. Gleichzeitig wird die Session auf dem Reverse Proxy gelöscht, damit der Benutzer nicht erneut über diese Session auf die Webapplikation zugreifen kann.

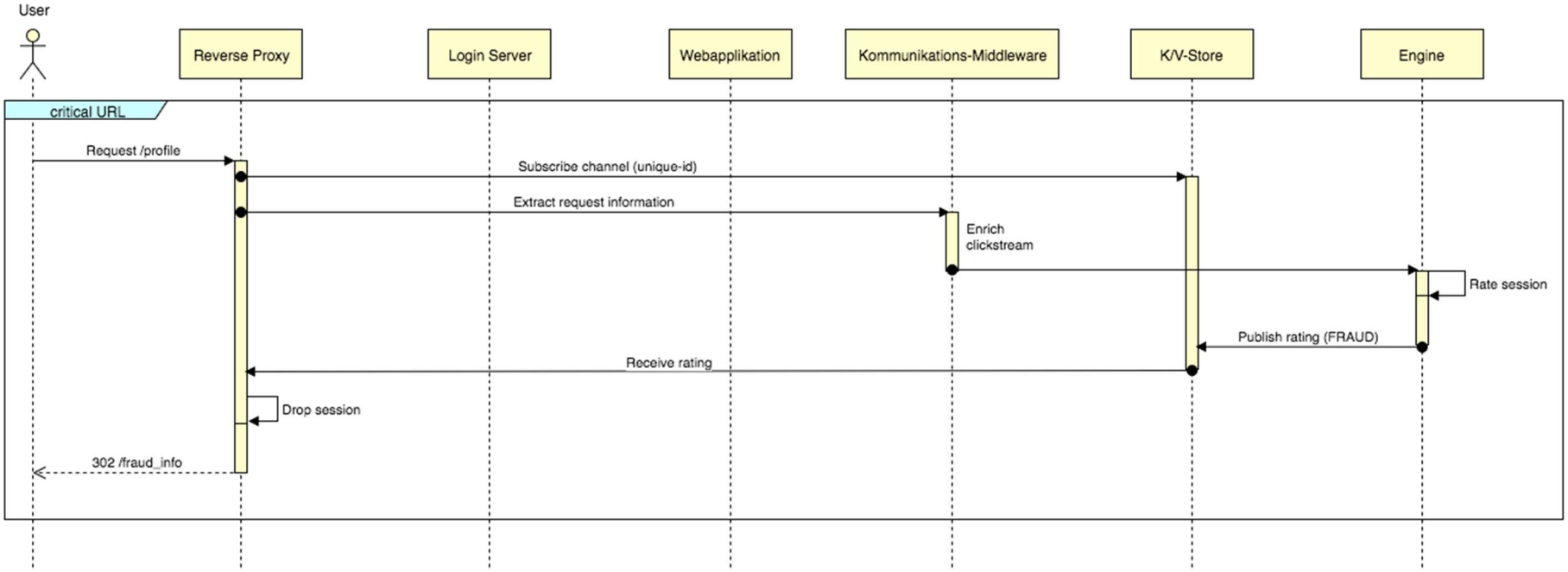


Diagramm 8 FRAUD Case

III.6. Analyse

III.6.1. Domain Model

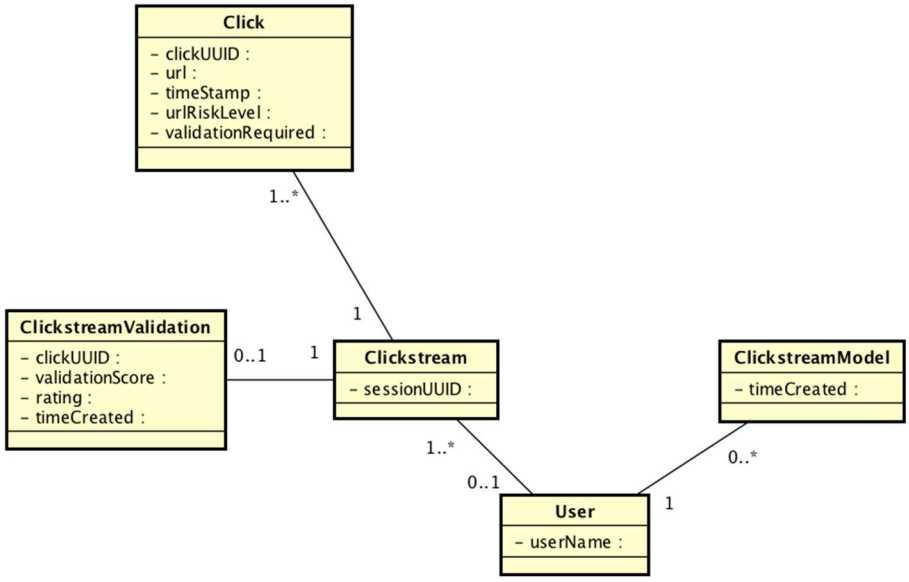


Diagramm 9 Domain Model

Das Domain Model von MarkovShield besteht aus den folgenden Entitäten:

Entität	**Beschreibung**
Click	Ein Click repräsentiert genau einen Seitenaufruf der Webapplikation und wird durch eine eindeutige Identifikationsnummer identifiziert *(clickUUID)*. Ein Click besitzt neben der aufgerufenen URL *(url)* auch einen genauen Zeitpunkt, an welchem dieser Click ausgeführt wurde *(timeStamp)*. Zudem ist für jeden Click festgehalten, wie hoch das Risiko dieser URL eingeschätzt wird *(urlRiskLevel)* und ob eine Validierung des bisherigen Clickstreams benötigt wird *(validationRequired)*. Diese zusätzlichen Informationen werden durch die Konfiguration von mod_mshield festgelegt.
Clickstream	Ein Clickstream besteht aus einem oder mehreren Clicks, welche im Laufe einer Session ausgeführt wurden. Dabei spielt es keine Rolle, ob diese vor oder nach der Anmeldung eines Benutzers ausgeübt wurden. Erst mit dem Logout wird ein neuer Session-Identifier *(sessionUUID)* generiert und der bestehende Clickstream beendet.

User	Der User ist ein Benutzer der Webapplikation, welcher sich erfolgreich authentifizieren konnte. Ein User kann losgelöst voneinander mehrere Clickstreams durchführen. Ein Benutzer wird durch einen Benutzernamen *(userName)* von anderen Benutzern unterschieden.
ClickstreamModel	Neben Clickstreams können einem User auch sogenannte ClickstreamModels zugeordnet werden. Diese definieren das erwartete Benutzerverhalten und werden aufgrund der vergangenen Clickstreams dieses Benutzers berechnet.
ClickstreamValidation	Die ClickstreamValidation legt den Zustand eines Clickstreams zu einem bestimmten Zeitpunkt fest *(timeCreated)*. Sie wird dann neu berechnet, wenn ein neuer Click *(clickUUID)* innerhalb des Clickstreams eine Validierung benötigt. Dafür wird der bisherige Clickstream mit den ClickstreamModels des Benutzers abgeglichen und ein Wert berechnet *(validationScore).* Dieser Wert bestimmt dann darüber, ob dieser Clickstream eine allfällige Gefahr darstellt und hält dies fest *(rating)*. Es ist jeweils nur die aktuellste ClickstreamValidation relevant.

Tabelle 6 Domain Model Entitäten

III.6.2. Persistenz

Das im letzten Kapitel aufgezeigte Domain Model bildet lediglich die logischen Verbindungen und Attribute ab. Für die Implementation wurden teilweise Änderungen am Modell vorgenommen, da keine klassische Datenbank eingesetzt wird und dadurch zum Teil weitere Datenelemente abgespeichert werden müssen.

III.6.2.1. Apache Kafka Persistierungs-Basics

Wichtig für das Verständnis der Persistierung ist dabei die Funktionsweise von Topics in Apache Kafka. Ein Topic ist eine Kategorie, in welche die Records veröffentlicht werden. Aus der Sicht der Persistenz ist ein Topic deshalb mit einer Table in einer Datenbank vergleichbar. Ein Topic kann jedoch in mehrere Partitionen unterteilt und somit auf mehreren Kafka Brokern verteilt werden. Dies ist dann notwendig, wenn ein Broker nicht mehr genügend Performance zur Verfügung stellen kann. Die Partition besteht aus mehreren Segments, in welchen die Daten physisch abgelegt werden. Die maximale Grösse eines Segments kann konfiguriert werden. Alternativ kann auch die maximale Zeit konfiguriert werden, nach welcher ein neues Segment erstellt wird.

Für jedes Topic speichert der Kafka Broker alle publizierten Records für eine bestimmte Zeit.[50] Diese Zeit wird Retention Period genannt und kann für jedes einzelne Topic konfiguriert werden.[51][52] Wichtig ist dabei, dass diese Retention nicht auf das aktuellste Segment greift und immer nur gesamte Segments gelöscht werden.

The log retention time is no longer based on last modified time of the log segments. Instead it will be based on the largest timestamp of the messages in a log segment.[53]

Zusätzlich zur Log Retention gibt es noch das Konzept der Log Compaction, auch dieses betrifft nur alte Segmente.[54] Log Compaction löscht im Gegensatz zur Log Retention die betroffenen Records nicht komplett, sondern behält lediglich den aktuellsten Record pro Key. Dies ist dann sinnvoll, wenn kein vollständiges Reprocessing aller Records benötigt wird und der letzte Stand jedoch noch von Bedeutung ist. Somit sind zum Teil erhebliche Speichereinsparungen möglich.

III.6.2.2. Zu persistierende Daten

In MarkovShield sind, wie in den meisten Applikationen, verschiedenste Arten von Daten vorhanden. Diese Daten haben verschiedene Quellen, Verwendungszwecke sowie auch eine unterschiedliche Lebensdauer. In diesem Kapitel werden diese Punkte für die wichtigsten Daten festgehalten.

[50] (Apache Kafka)
[51] (Kumar, 2016)
[52] (Confluent, 2016)
[53] (Apache Kafka)
[54] (Apache Kafka)

Click

Bezeichnung	**Beschreibung**
Entstehung	Ein Click entsteht, für jeden Seitenaufruf eines Benutzers auf die Webapplikation.
Verwendungszweck	Die Clicks werden benötigt um den Verlauf der Benutzersession abzubilden.
Historie	Die einzelnen Clicks müssen nicht für eine längere Zeit persistiert werden, da die relevanten Daten in den aggregierten Clickstreams vorhanden bleiben.
Persistierung	Mod_mshield persistiert die benötigten Daten jedes Clicks in das entsprechende Topic. Die Log Retention kann relativ tief definiert werden. Die Default Einstellung von 7 Tagen sollte genügen. Die Log Compaction sollte deaktiviert sein, da die einzelnen Clicks nicht auf die vorhergehenden Clicks schliessen lassen und der Verlauf relevant ist.
Bemerkung	Der Click entspricht genau dem logischen Click aus dem Domain Model.

Tabelle 7 Persistierung Click

Session

Bezeichnung	**Beschreibung**
Entstehung	Eine Session entsteht, wenn sich ein Benutzer bei mod_mshield einloggt.
Verwendungszweck	Die Session wird benötigt um einem Clickstream den Kontext eines Benutzers zu geben und ihm somit auch ein allfällig vorhandenes User Modell hinzuzufügen.
Historie	Die einzelnen Sessions müssen nicht für eine längere Zeit persistiert werden, da die relevanten Daten in den aggregierten Clickstreams vorhanden bleiben.
Persistierung	Mod_mshield persistiert die benötigten Daten jeder Session in das entsprechende Topic. Die Log Retention kann relativ tief definiert werden. Die Default Einstellung von 7 Tagen sollte genügen. Die Log Compaction hat hier im Normalfall keinen Einfluss, da sich der Benutzer nur einmalig einloggt. Die Log Compaction sollte deshalb deaktiviert sein.
Bemerkung	Die Session entspricht der Verknüpfung eines Clickstreams mit dem entsprechenden Benutzer.

Tabelle 8 Persistierung Session

ValidationClickstream

Bezeichnung	**Beschreibung**
Entstehung	Ein ValidationClickstream entsteht durch die Aggregation der verschiedenen Clicks einer Session.
Verwendungszweck	Der ValidationClickstream wird für die Analyse des momentanen Risk Levels einer Session benötigt.
Historie	Jeder ValidationClickstream wird von der Engine validiert und in einen ValidatedClickstream übergeführt. Deshalb wird keine langfristige Persistierung benötigt.
Persistierung	Die Middleware persistiert die aggregierten Validation-Clickstreams in ein eigenes Topic. Die Log Retention Policy sollte auf eine mittelmässige Dauer gesetzt werden. Damit kann eine Analyse des Validation-Clickstreams mit dem entsprechenden User Model nochmals durchgeführt werden, um allfällige Unregelmässigkeiten bei der Validierung zu untersuchen. Es empfiehlt sich deshalb eine Log Retention im Bereich von zwei Wochen bis einem Monat zu setzen. Die Log Compaction kann dabei helfen, die Grösse der Daten trotzdem auf einem vernünftigen Level zu halten, da normalerweise das User Model während einer Session nicht wechselt und somit alle vorhergehenden ValidationClickstreams aus dem aktuellsten Record abgeleitet werden können.
Bemerkung	Der ValidationClickstream stellt den Clickstream vor der Analyse dar und beinhaltet neben der kompletten Click-Historie auch ein allfälliges User Model.

Tabelle 9 Persistierung ValidationClickstream

Bezeichnung	Beschreibung
Entstehung	Ein Clickstream entsteht durch die Aggregation der verschiedenen Clicks einer Session. Der ValidatedClickstream wiederum entsteht nach der Validierung eines Clickstreams von der Engine.
Verwendungszweck	Der ValidatedClickstream wird einerseits für das Training der User Models benötigt, andererseits kann damit über Interactive Queries[55] der Status einer Session abgefragt werden.
Historie	Jeder ValidatedClickstream stellt einen kompletten Clickstream inklusive der ausschlaggebenden Validierung dar und bestimmt zusätzlich über die zukünftigen User Models. Eine langfristige Persistierung ist aus diesen Gründen empfehlenswert.
Persistierung	Die Engine persistiert die validierten Clickstreams in ein eigenes Topic. Die Log Retention Policy sollte auf eine lange Dauer gesetzt werden, um die berechneten Analyseresultate abrufen zu können. Es empfiehlt sich deshalb eine Log Retention im Bereich sechs bis zwölf Monaten. Es empfiehlt sich auf diesem Topic die Log Compaction zu aktivieren und dadurch die Datenmenge zu reduzieren. Somit ist nur das letzte berechnete Analyseresultat vorhanden. Wenn es nötig ist die volle Nachvollziehbarkeit über einen längeren Zeitraum zu gewährleisten, kann dies entsprechend konfiguriert werden.[56]
Bemerkung	Der ValidationClickstream stellt den Clickstream nach der Analyse dar und beinhaltet neben der kompletten Click-Historie auch eine ClickstreamValidation und damit den aktuellen Risk Level der Session.

Tabelle 10 Persistierung ValidatedClickstream

[55] Siehe auch Kapitel III.8.1.10 Interactive Queries
[56] (Apache Kafka)

User Model

Bezeichnung	**Beschreibung**
Entstehung	Ein User Model entsteht durch die Auswertung von vergangenen Clickstreams durch die Engine.
Verwendungszweck	Das User Model bildet anhand der vergangenen Clickstreams das erwartete Verhalten eines Benutzers ab. Mithilfe des User Models können die zukünftigen Sessions eines Benutzers ausgewertet werden.
Historie	Für MarkovShield sind die veralteten User Models nicht von Relevanz. Das aktuelle User Model sollte jedoch über eine lange Zeit verfügbar sein.
Persistierung	Die Engine persistiert das User Model nach dem Erstellen in ein eigenes Topic. Die Log Retention Policy sollte auf eine lange Dauer gesetzt werden, um immer ein User Model zur Verfügung zu haben. Es empfiehlt sich deshalb eine Log Retention im Bereich sechs bis zwölf Monaten. Es empfiehlt sich auf diesem Topic die Log Compaction zu aktivieren und somit die Datenmenge zu reduzieren. Somit ist nur das letzte berechnete User Model vorhanden. Wenn es nötig ist die volle Nachvollziehbarkeit über einen längeren Zeitraum zu gewährleisten, kann dies entsprechend konfiguriert werden.[57]
Bemerkung	Das User Model beinhaltet alle ClickstreamModels eines Benutzers.

Tabelle 11 Persistierung User Model

[57] (Apache Kafka)

III.7. Design

Die grundsätzliche Systemarchitektur wurde im Kapitel III.3 System Architektur ausführlich beschrieben. Im folgenden Kapitel liegt der Fokus auf der Software-Architektur der einzelnen Komponenten.

III.7.1. mod_mshield

III.7.1.1. Abhängigkeiten

Innerhalb des mod_mshield Apache Modules wurde der Aufbau der Header Files von MOD_BUT übernommen. Ergänzend sind durch die MarkovShield Funktionalität drei zusätzliche Abhängigkeiten auf andere Libraries hinzugekommen. Einerseits sind dies die Libaries, librdkafka[58] und hiredis[59], welche die Kommunikation mit Kafka und Redis implementieren. Andererseits wurde cJSON[60] für die JSON-Serialisierung der Daten genutzt. Des Weiteren bestehen, wie für Apache Module üblich, Abhängigkeiten zu spezifischen Header Files von HTTPD und APR[61]. Innerhalb des Modules wird für das RegEx Matching die Perl Compatible Regular Expressions Library PCRE[62] verwendet, welche bereits in MOD_BUT eingesetzt wurde.

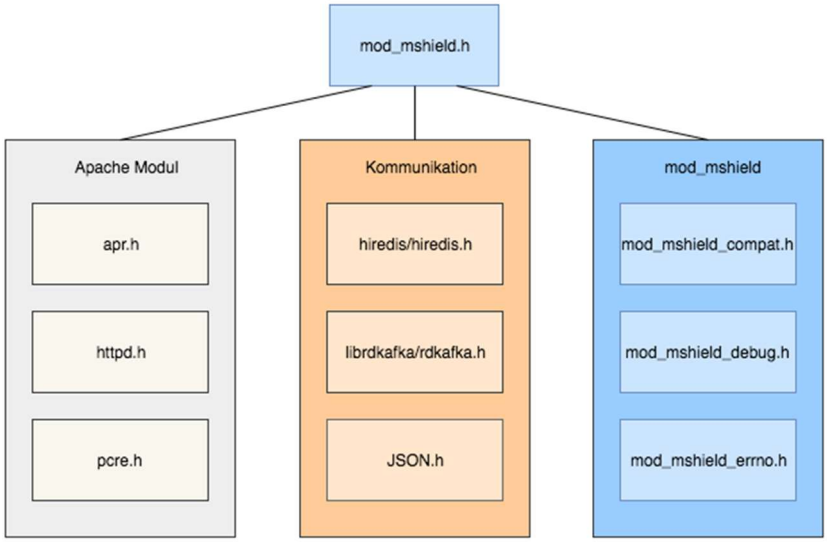


Abbildung 10 Abhängigkeiten von mod_mshield

[58] (Edenhill, github.com/edenhill, 2017)
[59] (redislabs, 2017)
[60] (DaveGamble, 2017)
[61] (The Apache Software Foundation, 2017)
[62] (Hazel, 2017)

III.7.2. Middleware und Engine

Im Maven Projekt «architecture_prototype» ist der gesamte Java-Code enthalten, welcher für den Betrieb von MarkovShield benötigt wird. Davon sind momentan die zwei Komponenten Middleware und Engine betroffen. Das Maven Projekt ist dabei in vier verschiedene Module unterteilt, welche im Diagramm 10 Maven Modules Übersicht aufgeführt sind.

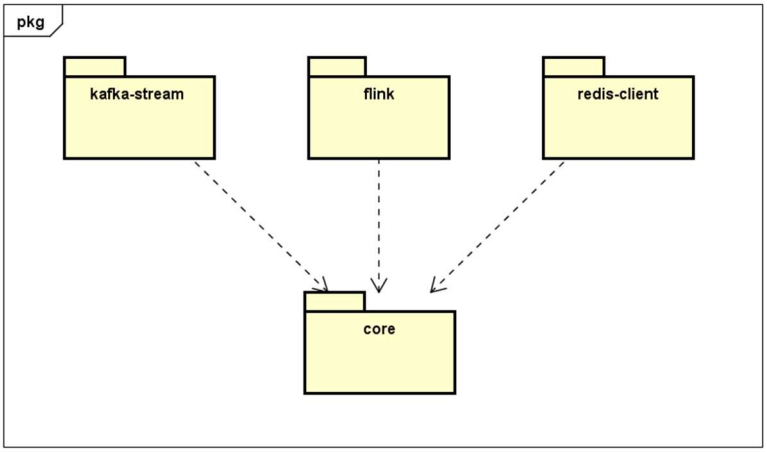


Diagramm 10 Maven Modules Übersicht

Dies führt zu einer besseren Struktur und Unterscheidung zwischen der Middleware und der Engine. Ebenfalls werden dadurch Dependency-Probleme behoben, welche im Verlauf des Projekts aufgetreten sind.

III.7.2.1. core

Im «core»-Modul sind hauptsächlich bearbeitungsunabhängige Klassen, wie zum Beispiel die Domain-Klassen aufzufinden. Diese haben häufig die Eigenschaft in verschiedenen Modulen verwendet zu werden und sind grundsätzlich unabhängig vom eingesetzten Framework.

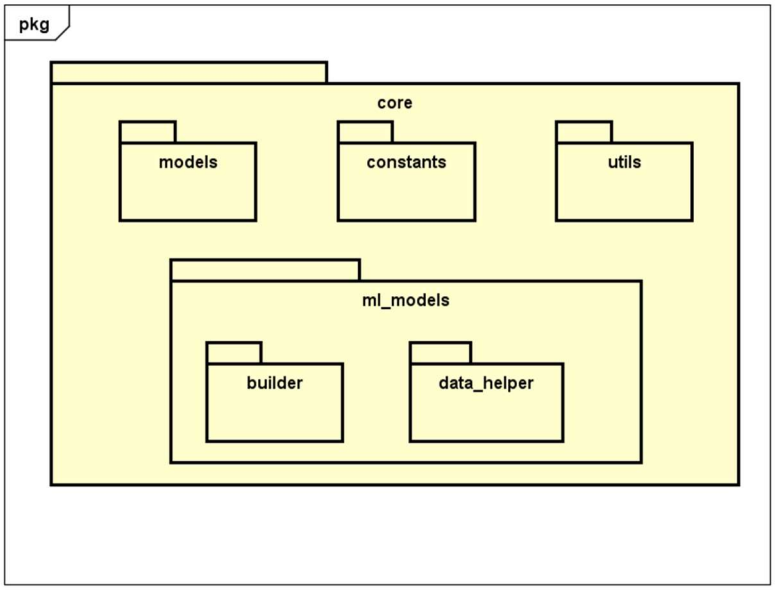


Diagramm 11 Package-Diagramm Core

Zusätzlich zu den Domain-Klassen, welche im Package «models» platziert sind, gibt es ein zweites sehr wichtiges Package «ml_models», mit zwei Subpackages. Darin enthalten sind jegliche Klassen, welche für die Models benötigt werden. Direkt darin enthalten sind die verschiedenen implementieren Models. Im Subpackage «builder» sind die verschiedenen Methoden implementiert um die Models zu berechnen und im Package «data_helper» sind einfache Datenklassen abgelegt.

Zudem gibt es die Packages «constants» und «utils», welche globale Helferklassen beinhalten. Zum Beispiel sind darin die Namen der benötigten Kafka Topics hinterlegt, sodass bei einer Änderung direkt die Middleware und die Engine auf das neue Topic zugreifen. Im Package «utils» ist unter anderem eine Helferklasse verfügbar, welche die Verwendung von Parametern vereinfacht.

III.7.2.2. flink

Im Modul «flink» sind alle Flink-spezifischen Klassen enthalten. Unter anderem sind darin die Flink-Jobs «MarkovShieldAnalyser» und «MarkovShieldModelUpdater» definiert. Das Modul «flink» besitzt lediglich ein Subpackage «serialization», in welchem die benötigten Serialisierungsschemas enthalten sind.

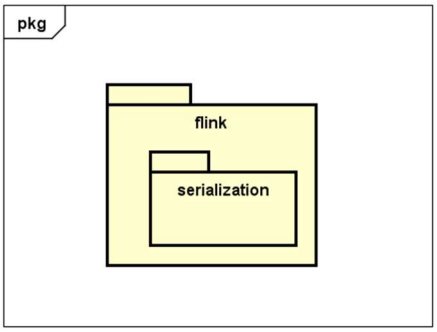


Diagramm 12 Package-Diagramm flink

III.7.2.3. kafka-stream

Das «kafka-stream»-Modul ist für die Aggregierung der Clicks zu Clickstreams, sowie der Anreicherung deren mit zusätzlichen Informationen zuständig. Zusätzlich dazu enthält das Modul ein Package «interactive_query», welches ausgewählte Daten über eine REST-API zur Verfügung stellt.

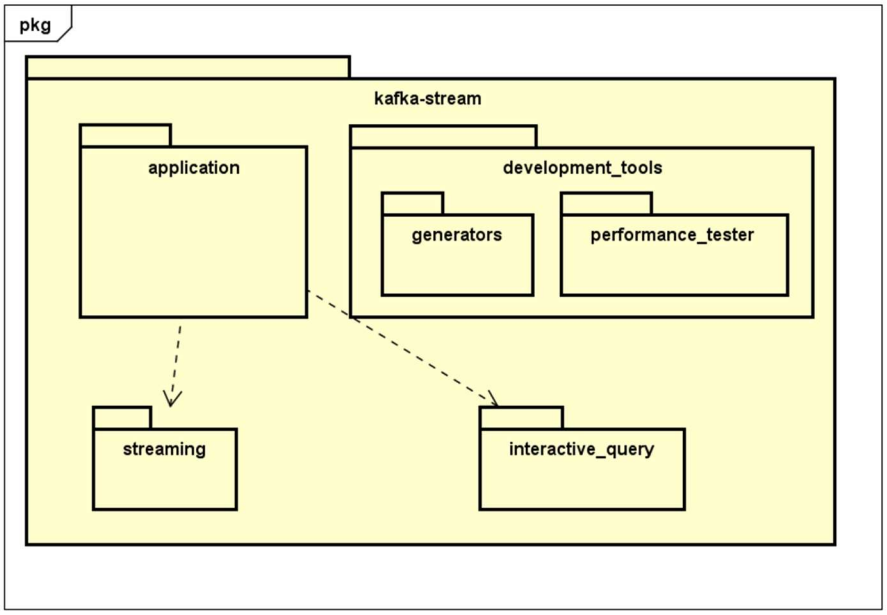


Diagramm 13 Package Diagramm kafka-stream

Die Hauptteile von «kafka-stream» sind in den Packages «application» sowie «streaming» enthalten. Im «application»-Package sind mehrheitlich Details zur Konfiguration von Apache Kafka Streams aufzufinden, während im «streaming»-Package die Logik für die Aggregation und die Anreicherung festgelegt wird.

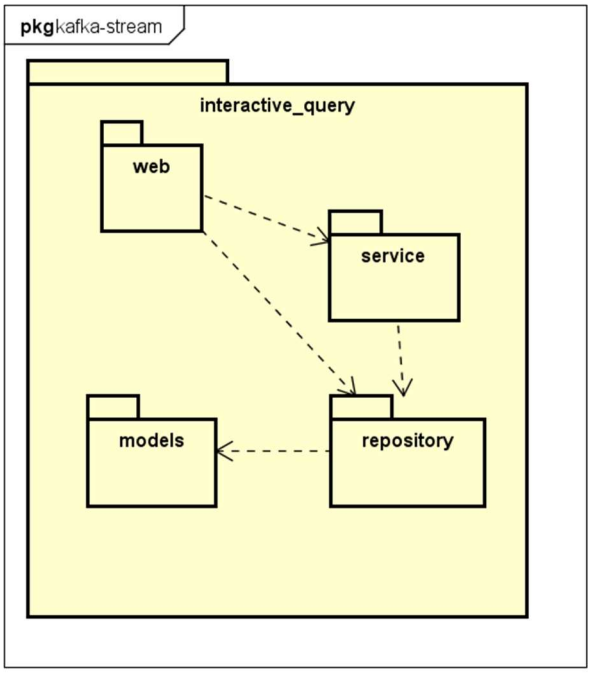


Diagramm 14 Package-Diagramm interactive_query

Im Subpackage «interactive_query» sind alle Klassen enthalten, welche für die Interactive Queries benötigt werden. Die Interactive Queries ermöglichen es gewisse aktuelle Daten direkt aus Apache Kafka zur Verfügung zu stellen. Im Fall von MarkovShield werden diese Daten über eine einfache REST-API zur Verfügung gestellt. Das Package «web» ist dabei für diese Schnittstelle zuständig und bezieht seine Informationen normalerweise aus den entsprechenden Services. Die Services beinhalten die Businesslogik und beziehen die grundliegenden Daten dafür aus einem Repository. Das Repository ist dabei für die Abfragen der Daten zuständig. Mehr Informationen zu Interactive Queries und zur spezifischen Implementation in MarkovShield sind im Kapitel III.8.1.10 Interactive Queries zu finden.

III.7.2.4. redis-client

Das Modul «redis-client» ist das mit Abstand kleinste Modul. Es ist erst im späteren Verlauf der Arbeit entstanden und beinhaltet lediglich ein Programm, welches für Performance-Analysen von MarkovShield genutzt werden kann. Aufgrund anderer genutzter Dependencies und der guten Trennbarkeit von anderen Funktionalitäten wurde entschieden ein zusätzliches Modul dafür zu erstellen.

III.8. Implementation und Testing

III.8.1. Implementation

Das folgende Kapitel enthält genauere und mehrheitlich technische Information über die Implementation von MarkovShield. Unter anderem sind hier wichtige Überlegungen sowie einige herausstehende Herausforderungen festgehalten.

III.8.1.1. Kommunikation des Ergebnisses

Während der Implementation der Kafka Consume Funktionalität im mod_mshield Apache Modul wurde festgestellt, dass die Kommunikation des Analyseergebnisses nicht wie angedacht bewerkstelligt werden kann. Ursprünglich sollten über Kafka Consume die Engine Resultate von den kritischen Request zu mod_mshield zurückgelangen. Es stellte sich jedoch heraus, dass es nicht ohne weiteres möglich ist, eine Korrelierung zwischen den gesendeten Informationen eines Requests und einem erhaltenen Resultat zu erreichen. Somit war mod_mshield nicht in der Lage, ein über Kafka Consume erhaltenes Resultat eindeutig einem zurückgehaltenem und wartendem Request zuzuordnen. Der Grund darin liegt, dass Kafka von Haus aus keine Funktionalität an anbietet, um nur einen spezifischen Message Key zu consumen und die anderen Messages zu ignorieren. Zudem wird beim Lesen einer Message das Offset der Consumer Gruppe inkrementiert. Dies bedeutet, dass andere Kafka Consumer, welche der gleichen Kafka Consumer Group angehören, nur Messages mit einem grösseren Offset konsumieren werden. Somit sind diese Kafka Consumer dann nicht mehr in der Lage, die zuvor erhaltenen Messages zu lesen ohne dass sie das ganze Topic neu lesen müssten.

Aus dem erwähnten Grund wurde deshalb während der Implementation entschieden, auf einen Key-/Value-Store für die Übertragung der Engine Resultate zu wechseln. Wie sich herausstellte, bietet Redis genau für einen solchen Zweck optimal passende Funktionalitäten an - nämlich eine Publish-/Subscribe-Unterstützung über Channels.

III.8.1.2. Verbindungsaufbau

Während der Implementation der Verbindungssetups in mod_mshield zu Apache Kafka und dem Key-/Value-Store Redis, kam die folgende Frage auf:

«Was passiert, wenn die Engine bereits ein Analyse Resultat liefert, bevor mod_mshield eine Verbindung zu Redis aufgebaut hat?»

Diese Frage kam auf, weil für Kafka aufgrund der sehr gut ausgebauten Library librdkafka nur einmalig eine Verbindung zum Kafka Broker selbst aufgebaut werden muss. Die librdkafka Library arbeitet intern mit einer Queue und mehreren Threads um optimale Antwortzeiten garantieren zu können. Bei Redis wiederum muss aufgrund der Library-Vorgaben für jedes Resultat eine eigene Verbindung aufgebaut werden, da falls es zu einem «*redisReply*»-Fehler kommt, der Redis Verbindungscontext nicht mehr verwendet werden kann.

Ein denkbarer Fall wäre nun, dass die Request Informationen längst via Kafka zur Engine gesendet und dort ausgewertet wurden, während mod_mshield immer noch versucht eine Verbindung zu Redis aufzubauen. In einem solchen Fall würde das Analyseergebnis mod_mshield nie erreichen. Der Grund liegt darin, dass ein Redis Channel nur existiert während mindestens ein Client drauf eine Subscribe-Operation offen hat.

Als Lösung dieses Problems wurde die Redis Funktionalität in zwei Teile aufgeteilt. Im ersten Teil wird für kritische URLs eine Verbindung zu Redis aufgebaut und mit der Subscribe-Operation auf die ClickUUID des Requests wird ein Redis Channel aufgebaut. Unmittelbar anschliessend werden die zuvor als JSON serialisierten Request Informationen an Kafka gesendet. Diese Operation wird sowohl für kritische als unkritische Requests durchgeführt. In einem anschliessenden Schritt wird für kritische URLs der zweite Teilschritt der Redis Funktionalität ausgeführt. Dabei wird auf eine Antwort der Engine auf dem offenen Chanel gewartet. Kann innerhalb eines frei definierbaren Timeouts kein Resultat von der Engine empfangen werden, wird der Request auf eine Error Seite weitergeleitet.

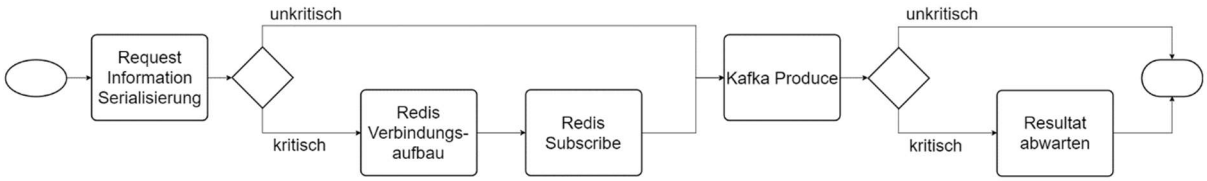


Abbildung 11 Ablauf eines Requests

Trotz Performanceoptimierungen wird eine gewisse Zeit benötigt, bis die Verbindungen aufgebaut sind und bis das Resultat von der Engine beim mod_mshield Modul angekommen ist. In der übernommenen Implementation von MOD_BUT wird beim Eintreten in die Hauptfunktion «*mshield_access_checker*» ein globaler Mutex («mshield_mutex») gelockt und erst vor jedem «return»-Statement wieder freigegeben. Dies verhindert zum einen, dass die Zugriffe auf das Shared-Memory voreinander geschützt sind und das es somit nicht zu Speicherverletzungen kommen sollte. Zum anderen jedoch verhindert dies auch, dass das Modul nicht wirklich durch mehrere parallel laufende Apache Worker verwendet werden kann, da sie sich gegenseitig aussperren.

Durch die Integration der MarkovShield spezifischen Funktionalitäten und die damit verbundenen benötigten Verbindungssetups zu den anderen Komponenten, würde dieser globale Mutex im Vergleich enorm lange blockiert werden und die Gesamtperformance des Moduls massiv vermindern. Aus diesem Grund wurde die Implementation entsprechend abgeändert, sodass der globale Mutex vor dem Warten auf eine Engine Antwort freigegeben wird. Der Mutex wird neu angefordert, sobald eine Antwort von der Engine erhalten wurde. Damit wird sichergestellt, dass mit der restlichen Funktionalität von mod_mshield sicher fortgefahren werden kann. Diese Implementation ist immer noch stark verbesserungsfähig, während der Arbeit wurde jedoch bewusst auf Refactorings von MOD_BUT verzichtet.[63]

[63] Siehe Kapitel III.12.2 mod_mshield Refactorings

III.8.1.3. Serialisierung

In einem System ist es von grossem Vorteil, wenn ein einheitliches Serialisierungsformat eingesetzt wird. Für Apache Kafka ist es empfohlen auf Apache Avro zu setzen, wobei Apache Kafka generell formatunabhängig realisiert ist.

Nach mehreren Versuchen wurde vom Einsatz von Apache Avro abgesehen, da diverse Probleme aufgetreten sind. Unter anderem ist es mit Apache Avro nicht möglich Vererbung abzubilden. Als Alternative Einsatzmöglichkeit wurde JSON gewählt.

Innerhalb des mod_mshield Apache Modules wurde cJSON[64] für die Serialisierung der Request Informationen und der User-UUID-Mappings verwendet. cJSON ist ein einfacher und leichtgewichtiger JSON Parser für ANSI C. Es wird lediglich ein C File und ein Header File benötigt, um damit JSON in beide Richtungen parsen zu können.

```
1   cJSON *click_json;
2   click_json = cJSON_CreateObject();
3   cJSON_AddItemToObject(click_json, "sessionUUID", cJSON_CreateString(uuid));
4   cJSON_AddItemToObject(click_json, "clickUUID", cJSON_CreateString(clickUUID));
5   cJSON_AddItemToObject(click_json, "timeStamp", cJSON_CreateNumber(r->request_time / 1000));
6   cJSON_AddItemToObject(click_json, "url", cJSON_CreateString(url));
7   cJSON_AddItemToObject(click_json, "urlRiskLevel", cJSON_CreateNumber(risk_level));
8   cJSON_AddItemToObject(click_json, "validationRequired", cJSON_CreateBool(validationRequired));
9   kafka_produce(cJSON_Print(click_json), ...);
10  cJSON_Delete(click_json);
```

Abbildung 12 ClickEntry Serialisierung mittels cJSON

[64] (DaveGamble, 2017)

In Java wird dabei Jackson[65] als Serializer verwendet, wobei aus Performancegründen zusätzlich das Module Afterburner[66] eingebunden wird. Ebenfalls aus Performancegründen wurde das binäre Format Smile evaluier, welches von Jackson unterstützt wird. Leider gilt die C- Implementation von Smile als experimentell.[67] Aus diesem Grund wurde zwischen der Serialisierung zwischen mod_mshield und dem Rest von MarkovShield unterscheiden. Für beide Teile kann unabhängig zwischen der Serialisierung mit Smile und JSON ausgewählt werden.

Alle Modelklassen wurden mit entsprechenden Annotationen, insbesondere «@JsonProperty» versehen um zukünftige Änderungen am Code oder den JSON-Format ohne gegenseitige Abhängigkeit vornehmen zu können.

```
public class ClickStream {

    private final List<Click> clicks;
    private String userName;
    private String sessionUUID;

    @JsonCreator
    public ClickStream(@JsonProperty ("userName") String userName,
                       @JsonProperty ("sessionUUID") String sessionUUID,
                       @JsonProperty ("clicks") List<Click> clicks) {
        this.userName = userName;
        this.sessionUUID = sessionUUID;
        this.clicks = new ArrayList<>(clicks);
    }
```

Abbildung 13 Jackson Property Annotationen

Um im User Model unabhängig vom Typen verschiedene ClickstreamModel abspeichern zu können, beziehungsweise für die Deserialisierung, müssen diese mit zusätzlichen Type-Informationen versehen werden. Jackson unterstützt bietet dafür eine einfache Annotation auf Interface-Ebene an.

```
@JsonTypeInfo (use = JsonTypeInfo.Id.CLASS, include = JsonTypeInfo.As.PROPERTY, property = "@class")
public interface ClickStreamModel extends Serializable {

    Date getTimeCreated();

    double clickStreamScore(ClickStream clickStream);
}
```

Abbildung 14 Jackson Type Annotation

[65] (Jackson Project Home @github)
[66] (Jackson Modules Base Afterburner)
[67] (Libsmile)

III.8.1.4. Konfiguration

Für die Konfiguration der Java Programme wird Apache Commons CLI benutzt.[68] Dies ist eine Java Library, welche das Parsen von Command-Line-Argumenten übernimmt. Zudem erlaubt uns der Einsatz von Commons CLI eine einfache Hilfefunktion anzubieten, um einen Überblick über die verfügbaren Konfigurationsparameter zu geben. In MarkovShield wird das «GNU like long options»-Format verwendet.

```
usage: gnu
    --bootstrap <arg>       address of the kafka bootstrap, it's default
                            is: broker:9092
 -h,--help                  print this message
    --numthreads <arg>      the number of threads that kafka-streams will
                            use, it's default is: 1
    --resthostname <arg>    hostname of the REST endpoint, it's default is:
                            localhost
    --restport <arg>        port of the REST endpoint, it's default is:
                            7777
    --zookeeper <arg>       address of the zookeeper, it's default is:
                            zookeeper:2181
```

Abbildung 15 Hilfetext von MarkovShieldClickstreams

Die verschiedenen Parameter werden beim Programmstart angegeben und können beliebig kombiniert werden.

Eine vollständige Übersicht über die verfügbaren Konfigurationsparameter ist im Kapitel III.9.4.1 Konfiguration aufgeführt. Alternativ sind alle Parameter auch im entsprechenden Repository aufgeführt.

Für eine einfachere Benutzung, wurde die Klasse OptionHelper entwickelt, welche eine einfache Verwendung von optionalen Konfigurationsoptionen erlaubt. Durch die Rückgabe eines Java Optionals ist es möglich direkt den Default-Wert anzugeben, was die Entwicklerfreundlichkeit und Lesbarkeit stark erhöht.

```
String redisHost = OptionHelper.getOption(commandLineArguments, REDIS_HOST_ARGUMENT_NAME)
    .orElse(DEFAULT_REDIS_HOST);
Integer redisPort = OptionHelper.getOption(commandLineArguments, REDIS_PORT_ARGUMENT_NAME)
    .map(Integer::valueOf)
    .orElse(DEFAULT_REDIS_PORT);
```

Abbildung 16 Verwendung des OptionHelpers

[68] (Commons CLI)

Für die Konfiguration des Apache Modules mod_mshield wurden, wie bei Apache Modulen üblich, Apache Konfigurationsdirektiven verwendet. Diese Apache Direktiven können im Normalfall in einem beliebigen «*.conf*»-File innerhalb eines Apache Konfigurationsverzeichnis verwendet werden. MOD_BUT stellte bereits diverse Konfigurationsmöglichkeiten zur Verfügung und durch die MarkovShield Erweiterung kamen allerdings einige Weitere dazu. Eine Liste der neuen Konfigurationsdirektiven kann unter Kapitel III.9.4.1 Konfiguration betrachtet werden. Diese Konfigurationsdirektiven werden beim Starten des Apache Services ausgelesen und anschliessend nicht mehr betrachtet, bis über das entsprechende Command ein Service Reload oder Restart veranlasst wird (Command: «*service httpd reload/restart*»).

In der aktuellen Implementation von MarkovShield werden die Risk Levels zu den jeweiligen URLs mittels Apache Direktiven definiert. Das Format ist dabei wie folgend:

«*MOD_MSHIELD_URL «^/URL/to/match*» 5*»

Der erste Teil «*MOD_MSHIELD_URL*» besagt Apache, dass es sich bei dieser Konfigurationszeile um eine MarkovShield URL Risk Level Definition handelt. Anschliessend an die URL Direktive muss ein RegEx Pattern definiert werden, auf welches die entsprechende URL matchen soll. Es kann grundsätzlich ein beliebiges Pattern nach Perl RegEx Syntax definiert werden. Dabei muss jedoch selbständig sichergestellt werden, dass sich mehrere Patterns nicht überschreiben. Zu guter Letzt, wird der Risk Level definiert, welcher für die URLs gelten soll, auf welche das definierte RegEx Pattern zu trifft. Wichtig ist hierbei, dass ein Level zwischen 0 und 1000 gewählt wird. Mit der Konfigurationsdirektive *MOD_MSHIELD_FRAUD_ VALIDATION_THRESHOLD* wird definiert, ab wann eine URL als kritische URL gelten soll.

III.8.1.5. Models generelle Struktur

Um das Hinzufügen von neuen Analyse-Algorithmen möglichst einfach zu gestalten, wurde der Grossteil der Implementation unabhängig der Algorithmen implementiert. Bereits zu Beginn wurde die Generierung der Models in separate Klassen ausgelagert. Diese Klassen, welche das Interface Model Builder implementieren, wurden vom MarkovModelUpdater verwendet um die konkreten Models zu instanziieren. Dem entsprechenden Model Builder müssen dabei lediglich die gewünschten Clickstreams übergeben werden um ein generiertes Model zu erhalten.

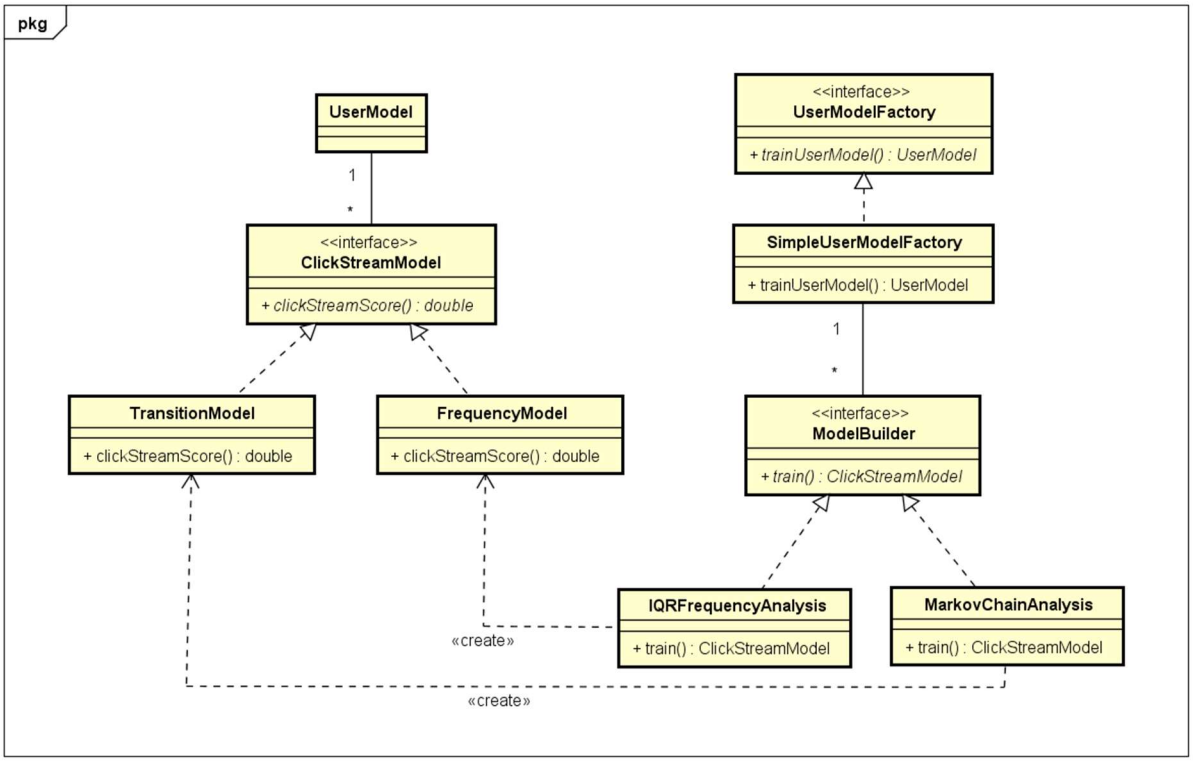


Diagramm 15 Klassendiagramm User Model Generation

Im Verlauf der Entwicklung wurde ein zusätzliches Abstraktionslevel hinzugefügt um einfach neue Model Builder in den Prozess einzubinden. Die UserModelFactory stellt dabei ein Interface zur Verfügung um direkt ein gesamtes User Model zu berechnen, in welchem im Normalfall mehrere Models enthalten sind. In der verwendeten Implementation SimpleUserModelFactory können die gewünschten Model Builder dem Konstruktor übergeben werden. Die Factory kümmert sich danach darum, dass jeder Model Builder aufgerufen wird und die berechneten ClickstreamModel im User Model enthalten sind.

Der MarkovShieldAnalyser verwendet bei der Berechnung des Gesamt-Scores alle im User Model enthaltenen Models unabhängig der spezifischen Implementation, sodass eine Erweiterung gewährleistet ist.

III.8.1.6. Model Builder Implementationsdetails

Wie im vorhergehenden Kapitel beschrieben sind die Model Builder für die Berechnung der verschiedenen Models zuständig. Momentan sind vier verschiedene Model Builder implementiert. In diesem Kapitel wird auf die wichtigsten Eigenschaften jedes Model Builders eingegangen. Für einen besseren Kontext empfiehlt es sich zuerst das Kapitel II.3.6 Berechnung der Models zu lesen.

```
public interface ModelBuilder extends Serializable {

    ClickStreamModel train(Iterable<ClickStream> stream);

}
```

Abbildung 17 Interface ModelBuilder

Das Interface besitzt lediglich eine Methode, welche ein Iterable von Clickstream entgegennimmt und daraus ein Model berechnet. Die Annahme ist, dass aus diesen Clickstreams nicht erwünschte Clickstreams bereits entfernt sind. In der momentanen Implementation betrifft das alle Clickstreams, welche den Status SUSPICIOUS oder FRAUD besitzen. Diese Clickstreams werden nicht an die Model Builder übergeben.

MarkovChainAnalysis

Die MarkovChainAnalysis widmet sich den Übergängen zwischen den verschiedenen URLs. Als erstes werden alle Übergänge zwischen den verschiedenen URLs gezählt. Dafür wird jeder Click innerhalb eines Clickstream mit seinem nachfolgenden Click verglichen. Eine solche Matrix kann man sich wie Abbildung 18 vorstellen. Dabei ist jede Spalte die ursprüngliche URL und jede Zeile das Ziel des ausgeführten Clicks. Insbesondere zu beachten ist dabei die letzte Zeile mit dem Eintrag endOfClickstream, welche das Ende eines Clickstreams darstellt.

	start.html	overview.html	transaction.html	news.html	logout.html
start.html	0	0	0	0	0
overview.html	5	2	4	1	0
transaction.html	0	4	0	0	0
news.html	0	2	0	1	0
logout.html	0	0	4	2	0
endOfClickStream	0	0	0	0	4

Abbildung 18 Logische Transition Matrix aufsummiert

Nachdem alle Clickstreams in die Matrix eingetragen wurden, müssen die einzelnen Spalten lediglich in prozentuale Einträge umgerechnet werden. Daraus folgt die finale Matrix Abbildung 19, welche im entsprechenden TransitionModel des Benutzers abgespeichert wird. Es

wurde entschieden, die Zuteilung einer URL zu einem Matrixindex separiert von den eigentlichen Daten abzulegen und somit eine verschachtelte Map-Struktur zu umgehen.

	start.html	overview.html	transaction.html	news.html	logout.html
start.html	0	0	0	0	0
overview.html	1	0.25	0.5	0.25	0
transaction.html	0	0.5	0	0	0
news.html	0	0.25	0	0.25	0
logout.html	0	0	0.5	0.5	0
endOfClickStream	0	0	0	0	1

Abbildung 19 Logische Transition Matrix prozentual

PDFFrequencyAnalysis

Die Klasse PDFFrequencyAnalysis setzt, wie bereits im Namen angetönt, auf die Probabilty-Density-Function. Eine kurze Beschreibung kann im Kapitel II.3.6 Berechnung der Models nachgelesen werden.

Die beiden Klassen PDFFrequencyAnalysis und IQFrequencyAnalysis sind sehr ähnlich aufgebaut, weshalb sie beide die abstrakte Klasse FrequencyAnalysis erweitern, welche einen grossen Teil der Logik beinhaltet.

Die Clickstreams müssen nun für die Analyse nach Session und URL aufsummiert werden, sodass eine Liste entsteht, welche URL in welcher Session wie oft angeklickt wurde.

	start.html	news.html	overview.html	transaction.html
Session1	1	3	2	2
Session2	1	2	1	1
Session3	1	0	1	1
Session4	1	3	1	2

Abbildung 20 Logische FrequencyMatrix

Für die weiteren Berechnungen ist die Zuteilung einer Frequency-Summe zu seiner Session nicht mehr von Relevanz, es ist lediglich notwendig diese voneinander zu trennen.

Anhand dieser Matrix kann danach für jede URL ein Intervall berechnet werden, in welches die erwarten Click-Frequencies in Zukunft fallen werden.

Für die Berechnung des zulässigen Intervalls wird die Klasse SummaryStatistics aus der Library Apache Commons Math3 verwendet.

IQRFrequencyAnalysis

Im Gegensatz zu PDFFrequencyAnalysis setzt IQRFrequencyAnalysis, wie bereits im Namen angetönt, auf Inter-Quartile-Range (IQR). Eine kurze Beschreibung von IQR kann im Kapitel II.3.6 Berechnung der Models nachgelesen werden.

Für die Berechnung des zulässigen Intervalls wird die Klasse DescriptiveStatistics aus der Library Apache Commons Math3 verwendet.

RandomModelBuilder

Der RandomModelBuilder dient lediglich zu Testzwecken. Er ignoriert die Clickstreams, welche als Argument übergeben werden und generiert ein RandomModel mit einer vorgegebenen Prozentzahl an SUSPICIOUS und FRAUD.

Durch die Einbindung des RandomModelBuilder kann relativ einfach das Verhalten von MarkovShield bei einem SUSPICOUS oder FRAUD-Case getestet werden.

III.8.1.7. Model Implementationsdetails

Die verschiedenen ClickstreamModels werden verwendet um einen einzelnen Clickstream nach gewissen Kriterien zu bewerten. Es gibt momentan drei verschiedene Implementationen eines ClickstreamModels.

FrequencyModel

Das FrequencyModel berechnet für einen Clickstream zuerst die Frequencies. Danach vergleicht es für jede aufgetretene URL die neue Frequency mit dem gespeicherten Intervall dieser URL. Wenn die Frequency nicht im Intervall liegt, wird der Score dieses Models erhöht. Die Erhöhung korreliert in der momentanen Implementation linear mit dem urlRiskLevel des Clicks, ist jedoch um den Wert 1 erhöht. Somit haben auch URLs mit einem tiefen urlRiskLevel einen Einfluss auf den Score, wichtigere URLs werden jedoch höher gewichtet.

```
if (isInIntervall(currentFrequency, currentURL)) {
    frequencyScore += urlRiskLevel + 1;
}
```

TransitionModel

Der Score des TransitionModel berechnet sich dadurch, dass für jede Transition des aktuellen Clickstreams Eins minus die gespeicherte Wahrscheinlichkeit mit dem urlRiskLevel multipliziert wird.

```
transitionScore += (1 - probabilityForClick) * (urlRiskLevelForClick + 1);
```

RandomModel

Das RandomModel ignoriert den übergebenen Clickstream und berechnet anstelle dessen einen Random-Wert zwischen 0 und 100. Aufgrund des berechneten Werts und den gespeicherten FRAUD und SUSPICIOUS-Anteilen gibt es einen entsprechenden Wert zurück.

```
if (random >= (100 - fraudShare)) {
    return 100;
}
if (random >= (100 - fraudShare - suspiciousShare)) {
    return 75;
} else {
    return 0;
}
```

III.8.1.8. GlobalKTable vs KTable

Auch der Aggregationsteil der Kommunikations-Middleware ist so implementiert, dass dieser auf verschiedenen Maschinen verteilt möglichst performant läuft. Momentan trifft dies im angedachten Deployment noch nicht zu, trotzdem wurde darauf geachtet, dass einem verteilten Deployment nichts im Wege steht. Ein Punkt der dies zum Ausdruck bringt ist die GlobalKTable. Um den Unterschied zwischen dem Einsatz der GlobalKTable und einer normalen KTable zu erklären, muss zuerst das Konzept einer KTable verstanden werden. Eine KTable entsteht durch einen Join von zwei KStreams, welcher zum Beispiel den kontinuierlichen Stream von Clicks wiederspiegelt. In der KTable wird jeder Record des Streams als Update der KTable betrachtet. Ebenfalls kann eine KTable durch die Aggregation eines KStreams erzeugt werden.[69]

Im Falle von MarkovShield soll der Stream von Clicks zuerst aggregiert und danach mit zusätzlichen Informationen, wie dem User Model versehen werden, bevor er zur Analyse weitergegeben wird.

[69] (Apache Kafka, kein Datum)

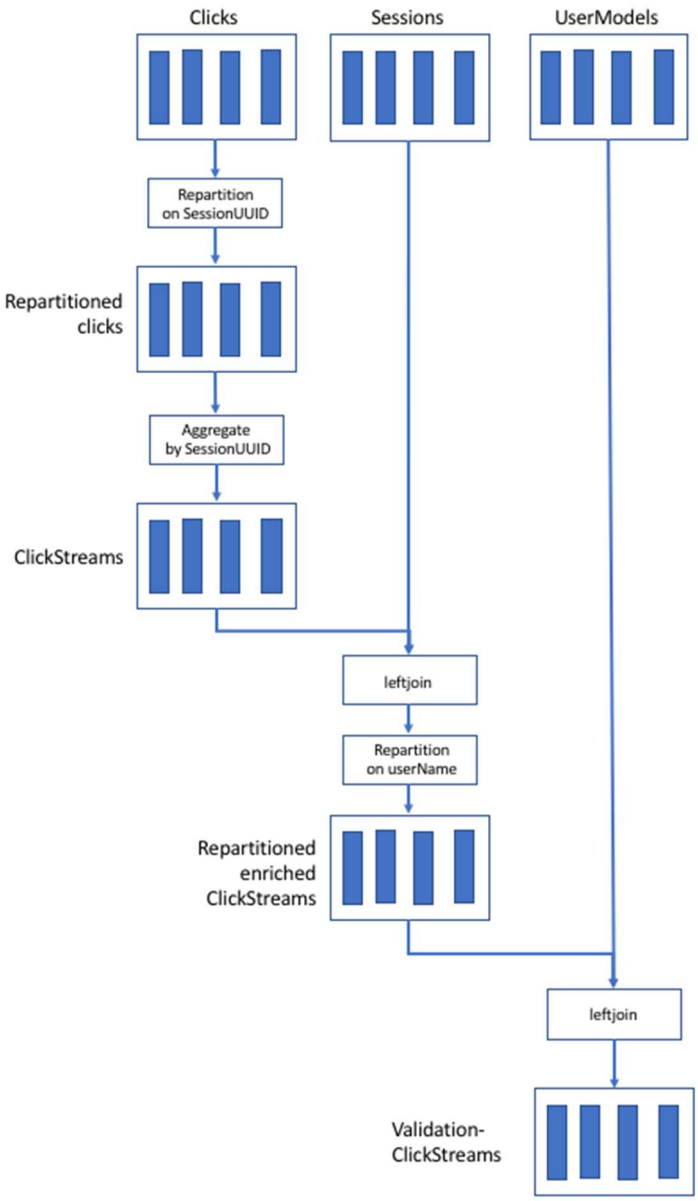


Abbildung 21 Aggregationsablauf ohne GlobalKTable

Beim Einsatz von normalen KTables für die Speicherung des aktuellsten Standes der Sessions sowie der User Models werden zwei Repartitionierungen und drei temporäre Topics benötigt.

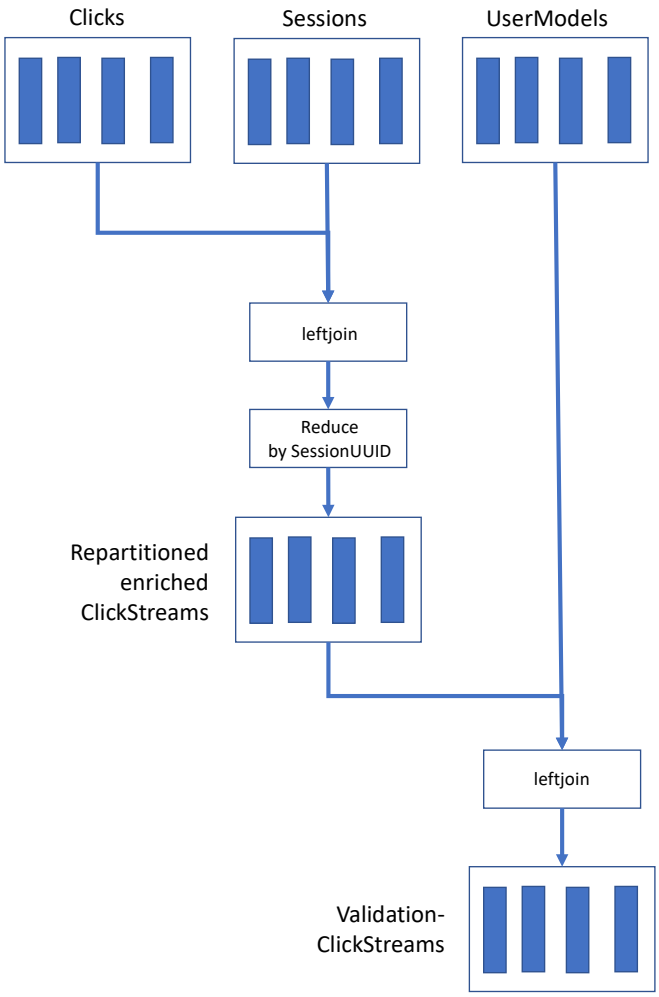


Abbildung 22 Aggregationsablauf mit GlobalKTable

Durch den Einsatz von GlobalKTables, welche über alle laufendenden Instanzen synchronisiert werden, wird nur noch ein temporäres Topic benötigt. Zudem fallen jegliche benötigte Repartitionierungen weg, was zu einer grossen Performanceeinsparung führen kann.[70]

III.8.1.9. MarkovShieldClickstreams Configuration Parameters

In der Klasse «MarkovShieldClickstreams» werden diverse Settings für den Aggregationsteil der Middleware definiert. Dabei gibt es einige wichtige und vielleicht etwas speziellere Parameter, auf welche in diesem Kapitel eingegangen werden.

Der wichtigste Parameter ist «CACHE_MAX_BYTES_BUFFERING_CONFIG». Er muss unbedingt auf 0 gesetzt sein, sodass die Joins keinen Cache benutzen und somit schnellst möglichst be-

[70] (Guy & et. al, KIP-99: Add Global Tables to Kafka Streams, 2017)

arbeitet werden. Zudem ist es für die Analyse notwendig, dass für jeden Click ein Validation-Clickstream veröffentlicht wird, sonst kann es theoretisch dazu führen, dass für einen Click keine Validierung durchgeführt wird und er somit abgelehnt wird. Zusammen mit diesem Parameter sollte jedoch unbedingt der Parameter «ROCKSDB_CONFIG_SETTER_CLASS_CONFIG» gesetzt werden, welcher es erlaubt das Caching der internen Rocksdb zu spezifizieren. Momentan ist bei der Rocksdb-Konfiguration der Write Buffer auf 32Mb gesetzt.

Indirekt damit verbunden ist auch der Parameter «STATE_DIR_CONFIG», über welchen die Lokation der lokalen Rocksdb spezifiziert werden kann. Er ist bei MarkovShield auf den relativen Pfad «kafka-store» gesetzt.

Der Parameter «APPLICATION_SERVER_CONFIG» wird für die Interactive Queries benötigt. Er definiert unter welcher Adresse andere Instanzen aufzufinden sind und erlaubt es somit diese abzufragen um auch in einem verteilten Setup komplette Rückgaben zu ermöglichen.

Der Parameter «NUM_STREAM_THREADS_CONFIG», welcher über die Command-Line-Konfiguration geändert kann, ermöglicht es das Stream Processing mit mehreren Threads zu starten. Standardmässig ist dieser Wert auf 1 gesetzt, da sich der Load momentan in Grenzen hält.

III.8.1.10. Interactive Queries

Um einen aktuelles Abbild der in Kafka gespeicherten Daten abfragen zu können, wurde Kafka Streams Ende des letzten Jahres mit der Funktionalität Interactive Queries erweitert. Diese erlauben es den aktuellsten State einer Kafka Streams Applikation abzufragen. In MarkovShield werden die Interactive Queries dazu benutzt, die aktuellsten Informationen über die User Models, die Sessions sowie die validierten Clickstreams mittels einer REST-API zur Verfügung zu stellen. Diese ist momentan relativ rudimentär aufgebaut, zeigt jedoch das Potential auf, welches Interactive Queries im behandelten Use-Case besitzen. Die entwickelte Lösung baut auf einem Beispiel[71] von Confluent, dem Hauptmaintainer von Apache Kafka, auf. Sie wurde jedoch erweitert und besser strukturiert. Die Hauptstruktur wurde bereits im Kapitel III.7.2.3 kafka-stream festgehalten. Als Webserver wird dabei Jetty verwendet, welcher bereits mit wenig Konfigurationsaufwand gestartet werden kann.

[71] (Apache Kafka, 2017)

```
@GET
@Path ("/users/{user}/usermodel")
@Produces (MediaType.APPLICATION_JSON)
public UserModel getUserModelByUser(@PathParam ("user") final String user) {
    return userModelService.getUserModel(user);
}
```

Abbildung 23 Annotationen REST-Endpoint

Die gewünschten Funktionalitäten können mit einigen wenigen Annotationen zur Verfügung gestellt werden. Der MarkovRestEndpoint ruft lediglich den entsprechenden Service auf und ist für die Erreichbarkeit und Serialisierung zuständig. Der Service wiederum fragt ein entsprechendes Repository ab und wendet auf den erhaltenen Daten die Businesslogik an, bevor er sie zur Verfügung stellt. Momentan sind darin vor allem Filterungen von Daten implementiert. In den entsprechenden Repositories findet dann die Abfrage mittels Interactive Queries statt.

Apache Kafka stellt dafür drei verschiedene Abfragen zur Verfügung, wovon im Moment lediglich zwei genutzt werden. Als erstes können alle momentan vorhanden Daten abgefragt werden. Alternativ dazu kann nach einem bestimmten Eintrag mittels seinem Keys gesucht werden und als dritte und letzte Möglichkeit, kann auch ein Range übergeben werden, nach welchem gesucht werden soll. Diese Methode wird allerdings momentan in MarkovShield nicht genutzt.

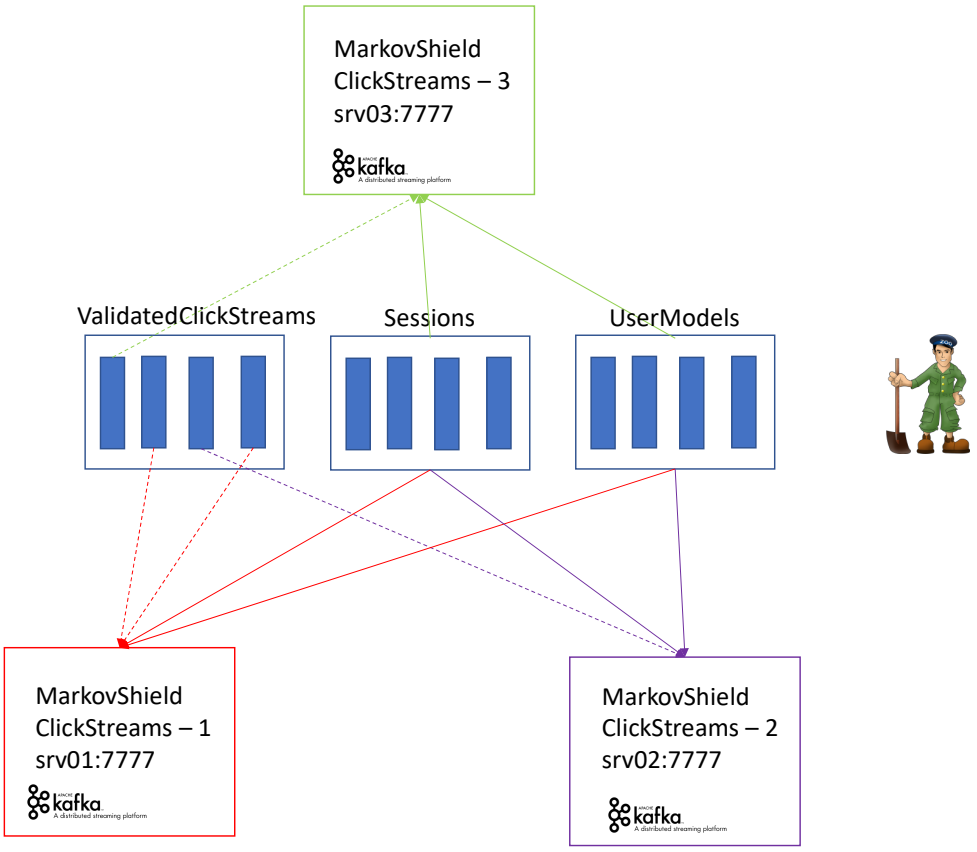


Abbildung 24 Interactive Query Verteilte System Architektur

Die potentielle Verteilung der Entries auf verschiedene Kafka-Streams Applikationen stellt besondere Anforderungen an die Repositories, da lediglich die lokalen Daten abgefragt werden können. Aus diesem Grund existiert neben KafkaStateRepository auch ein DistributedKafkaStateRepository. Das DistributedKafkaStateRepository ist dabei für die Abfrage von anderen MarkovShieldClickstreams zuständig. In der Abbildung 24 ist eine solche System Architektur grob aufgezeichnet. Dabei sind drei verschiedene Instanzen von MarkovShieldClickstreams am Stream Processing beteiligt. Die Zuteilung der einzelnen Partitionen zu einer Instanz wird dabei automatisch von Apache Kafka erledigt. Die Topics Sessions sowie User Models sind über jede Instanz synchronisiert, da hier eine GlobalKTable verwendet wird[72].

Wenn nun in eine Anfrage nach einem bestimmten ValidatedClickstream an srv01 gestellt wird, überprüft dieser, welche Instanz für diesen zuständig ist und leitet die Anfrage gegebenenfalls weiter. Um dies zu ermöglichen wird Zookeeper beim Start einer neuen Instanz deren

[72] III.8.1.8 GlobalKTable vs KTable

Endpoint mitgeteilt. Dieser wird wie im vorhergehenden Kapitel bereits erwähnt über den Parameter «APPLICATION_SERVER_CONFIG» festgelegt und kann MarkovShieldClickstreams beim Start der Applikation über Command-Line-Argument übergeben werden.

Alle API-Calls sind standardmässig unter der URI http://localhost:7777/MarkovShield/ aufzurufen.

Folgende API-Aufrufe können gemacht werden:

URI	Bezeichnung	Parameter
`/users/{user}` `/sessions`	Alle Sessions eines Benutzers zurückgegeben.	Der Benutzername des Benutzers. -case-sensitive
`/users/{user}` `/validatedClickstreams`	Alle Clickstreams eines Benutzers inklusive des letzten Validierungsresultates.	Der Benutzername des Benutzers. -case-sensitive
`/users/{user}` `/usermodel`	Das aktuellste User Model eines Benutzers.	Der Benutzername des Benutzers. -case-sensitive
`/validatedClickstreams`	Alle Clickstreams inklusive des letzten Validierungsresultats	-
`/validatedClickstreams` `/before/{timestamp}`	Alle Clickstreams inklusive des letzten Validierungsresultates vor einem bestimmten Zeitpunkt. Dabei zählt der letzte Click als Datum des Clickstreams.	Der Timestamp ist die Anzahl von Millisekunden seit Unix Epoch. -Exklusive des Timestamps
`/validatedClickstreams` `/after/{timestamp}`	Alle Clickstreams inklusive des letzten Validierungsresultates nach einem bestimmten Zeitpunkt. Dabei zählt der letzte Click als Datum des Clickstreams.	Der Timestamp ist die Anzahl von Millisekunden seit Unix Epoch. -Exklusive des Timestamps

`/validatedClickstreams /between/{timestampFirst} /{timestampLast}`	Alle Clickstreams inklusive des letzten Validierungsresultates zwischen zwei bestimmten Zeitpunkten. Dabei zählt der letzte Click als Datum des Clickstreams.	Der Timestamp ist die Anzahl von Millisekunden seit Unix Epoch. -Exklusive der Timestamps
`/validatedClickstreams /{sessionUUID}`	Der aktuellste Clickstream einer Session inklusive Validierungsresultat	Die SessionUUID des gewünschten Clickstreams -case-sensitive
`/usermodels`	Alle User Models	-
`/sessions`	Alle Sessions	-
`/sessions/{sessionUUID}`	Eine bestimmte Session	Die SessionUUID der Session -case-sensitive

Tabelle 12 Implementierte REST API Calls

III.8.1.11. Kafka Topic Creator

Beim Start von MarkovShieldClickstreams wird überprüft, ob alle benötigten Topics existieren. Andernfalls werden diese mithilfe des Kafka Topic Creators angelegt. Dabei werden auch die Topic-spezifischen Retention- sowie LogCompaction-Policies konfiguriert.

@mgabriel you can set the `cleanup.policy` to `compact,delete` and it will delete any segments older than `retention.ms`[73]

Während der Arbeit wurde festgestellt, dass es unter Umständen zu Problemen kommen kann, wenn genau gleichzeitig mehrere Instanzen von MarkovShieldClickstreams gestartet werden und noch keine Topics erstellt sind. Aus diesem Grund empfiehlt es sich in einem produktiven Einsatz auf den Topic Creator zu verzichten und die Topics vor dem Start von MarkovShieldClickstreams manuell anzulegen.

[73] (Guy, 2017)

III.8.1.12. Generators

Um die Funktionalität von MarkovShield unabhängig von mod_mshield zu testen, wurden im Verlaufe der Arbeit drei verschiedene Generatoren entwickelt, welche die Informationen direkt in Apache Kafka speichern. Unter anderem existiert ein Generator, welcher die User Sessions simulieren soll. Ein weiterer Generator kümmert sich um die Generierung von Models, sodass unabhängig vom ModelUpdater User Models angelegt werden können. Die Generatoren wurden bewusst sehr einfach gehalten und benutzen die Kafka-Producer-API.

III.8.1.13. Performance Testers

Da bei anfänglichen Performancetests die Gesamtlösung MarkovShield ungenügende Resultate erzielt hatte, wurden zusätzliche Tools entwickelt, welche an mehreren Teilpunkten in der Lösung die verbrauchte Zeit messen können. In der Abbildung 25 Systemarchitektur mit Zeitmessungspunkten ist die Systemarchitektur von MarkovShield abgebildet und zusätzlich wurde sie mit Zeitmessungspunkten ergänzt. An jedem der aufgezeigten Punkte wird dabei der exakte Zeitpunkt der Bearbeitung festgehalten, um damit bessere Aussagen über das Laufzeitverhalten treffen zu können. Für die Korrelation der einzelnen Messungen wird die jeweilige ClickUUID genutzt.

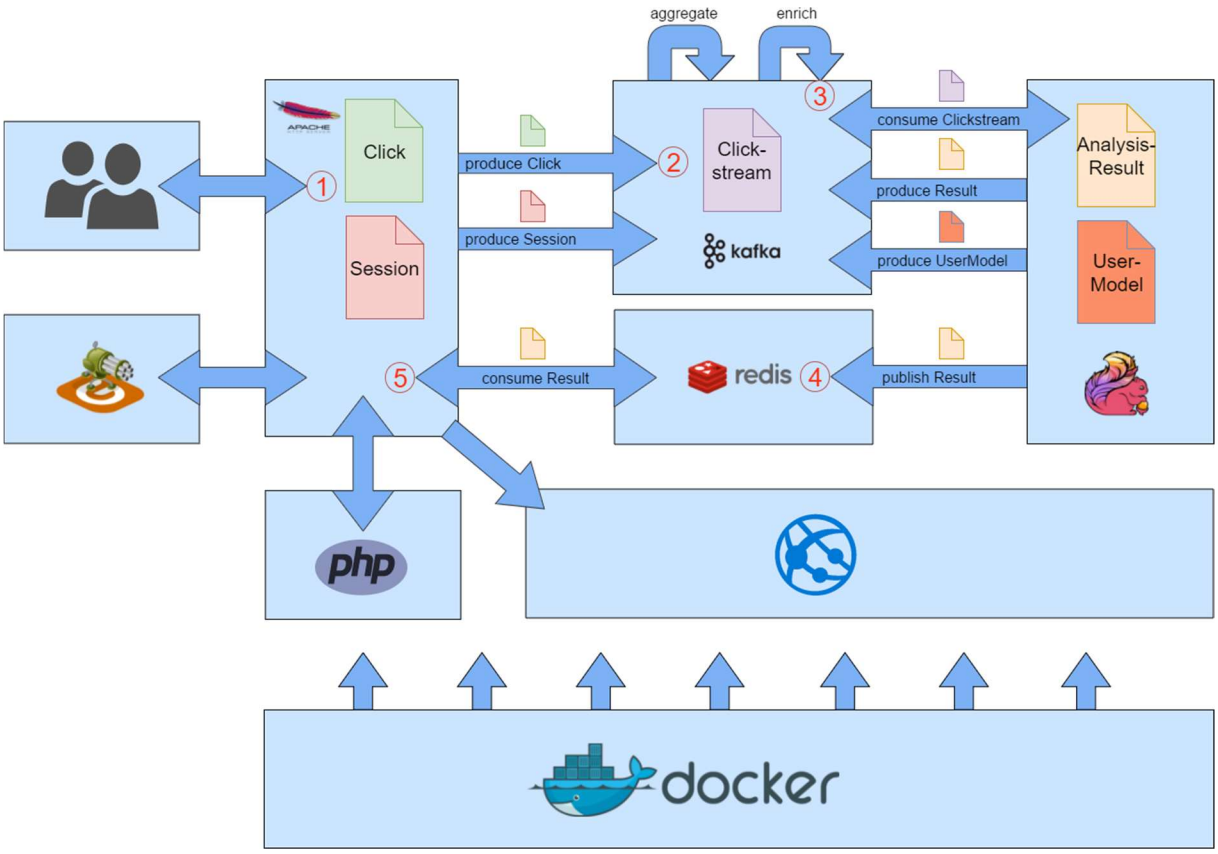


Abbildung 25 Systemarchitektur mit Zeitmessungspunkten

Der erste Timestamp (1) wird dabei von Apache HTTPD bei der Ankunft des Requests erstellt und zwischenzeitlich gespeichert. Dieser Timestamp wird zudem zusammen mit anderen Informationen an Apache Kafka übergeben. Für den zweiten Timestamp (2) registriert sich ein zusätzlicher Consumer beim Kafka Broker für das Click-Topic, in welches die Timestamps von mod_mshield veröffentlicht werden. Eine andere separate Applikation misst die aktuelle Zeit nach der Aggregation zu einem kompletten Clickstream (3).

Die letzte zusätzliche Zeitmessung (4) findet durch einen zusätzlichen Redis-Client statt, welcher auf allen Channels mithört und für die Korrelation mit den anderen Timestamps neben der aktuellen Zeit auch den Channel-Namen, also die ClickUUID, ausgibt.

Durch die Differenz der Timestamps zwei und vier kann die gesamte Durchlaufzeit durch den Analyseteil von MarkovShield berechnet werden. Eine grosse Differenz zwischen den Zeitpunkten eins und zwei, sowie vier und fünf lassen auf ein Problem mit mod_mshield schliessen.

III.8.1.14. Segmentation Faults

Während der Implementation von mod_mshield kam es mehrmals zu fehlerhaften Speicherzugriffen auf Werte (z.B. Structs), welche noch nicht initialisiert waren oder welche bereits freigegeben wurden. Diese Zugriffe resultierten dann jeweils in einem Segmentation Fault Fehler, welcher den Apache Worker Prozess zum Absturz brachte. Da meistens mittels sehr kleinen Schritten zwischen neuem Code schreiben und testen des neues Code gearbeitet wurde, konnten die Segmentation Faults relativ schnell mittels erneutem Überfliegen des Codes oder mit Hilfe von Logging Statements lokalisiert werden.

Gegen Ende der Arbeit als es um das ausführliche Testen der Funktionalität ging, kamen dann jedoch noch weitere Segmentation Fault Fehler zum Vorschein. Diese waren mit der vorhin erwähnten Methode nicht mehr zu lokalisieren, da überhaupt kein Ansatz vorhanden war, in welchem Codebereich diese Segmentation Faults in etwa aufgetreten sind. Um nun aber trotzdem die Segmentation Faults debuggen zu können, wurde der GNU Projekt Debugger (GDB)[74] verwendet.

Um mod_mshield mit dem GDB debuggen zu können, muss dieses mit den entsprechenden GCC[75] Debugging Informationen zusammen kompiliert werden (Parameter «*-g*»[76]). Konkret muss dieser GCC Parameter dem Tool APXS[77] («**AP**ache e**X**ten**S**ion tool») mitgegeben werden, da ein Apache Modul mittels APXS zu einem Dynamic Shared Object (DSO) kompiliert

[74] (GDB Developers, 2017)
[75] (Free Software Foundation, Inc., 2017)
[76] (Free Software Foundation, Inc., 2017)
[77] (The Apache Software Foundation, 2017)

werden muss. APXS wiederum verwendet GCC im Hintergrund und kann GCC Parameter mittels des Präfixes «*-Wc,*» direkt weitergeben. Somit wurde mod_mshield mit dem APXS Parameter «*-Wc,-g*» kompiliert, damit GDB die entsprechenden Debugging Informationen zur Verfügung hat.

Da der Apache Service standardmässig mittels mehreren Child Prozessen (Workers) läuft, muss der folgende Ablauf eingehalten werden, damit er mittels GDB debugged werden kann.

- Die Apache Umgebungsvariabeln müssen in der Bash Shell, in welcher anschliessend GDB laufen gelassen wird, gesourced werden.

  Command: «*source /etc/apache2/envvars*»

  Dies stellt sicher, dass verwendete Apache Umgebungsvariabeln im *httpd.conf* File aufgelöst werden können. Dieses File verwendet Apache standardmässig als Hauptkonfigurationsfile, wird dem Apache Process kein eigenes mitgegeben.
- Entgegen dem normalen Apache Startverhalten, muss mittels GDB nicht der Apache Service selbst gestartet werden, sondern nur ein einzelner Apache Prozess.

  Command: «*gdb apache2*»

- Innerhalb der nun geöffneten GDB Shell, muss als erstes ein Breakpoint gesetzt werden, damit die mod_mshield Funktionalität Schritt für Schritt durchlaufen werden kann. Im Rahmen des Debuggens der mod_mshield Segementation Faults, wurde die Hauptfunktion von mod_mshield als Breakpoint gesetzt.

  Commands (in der GDB Console): «*b mshield_access_checker*»

- Nachdem der Breakpoint gesetzt wurde, kann der Apache Prozess mit der Option «*-X*»[78] gestartet werden. Der Parameter «*-X*» bewirkt, dass Apache im Debug mode gestartet wird und im Vordergrund bleibt.

  Command (in der GDB Console): «*run -X*»

[78] (The Apache Software Foundation, 2017)

- Nun kann mittels des Browsers ein Request auf die Webapplikation gemacht und gleich wieder in die GDB Console gewechselt werden. In GDB kann mittels «*c*» oder «*continue*» bis zum nächsten Breakpoint vorgeschritten werden. Beim vorhin gesetzten Breakpoint angekommen, ist in der gdb Shell ersichtlich, auf welcher mod_mshield Zeile sich der Prozess aktuell befindet. Beispiel:

  (gdb) c
  Continuing.
  Breakpoint 1, mshield_access_checker (r=0x7ffff7e280a0) at mod_mshield.c:221

- Mittels «*n*» oder «*next*» kann nun Zeile für Zeile durchgegangen werden, bis der Segmentation Fault erneut auftritt. Beispiel:

  258 in mod_mshield.c
  (gdb) n

  Program received signal SIGSEGV, Segmentation fault.
  0x00007ffff771ec6c in apr_cpystrn (dst=0x2bf <error: Cannot access memory at address 0x2bf>, src=0x7ffff7e2c730 «H7KCoP8pMmj44fGd1piM+zOviUT0whXO», dst_size=<optimized out>) at /tmp/buildd/apr-1.5.1/strings/apr_cpystrn.c:59
  59 /tmp/buildd/apr-1.5.1/strings/apr_cpystrn.c: No such file or directory.

- Da nun die Stelle gefunden wurde, welche den Segmentation Fault verursacht, kann mittels «*bt*», «*backtrace*» oder «*where*» der aktuelle Stack angezeigt werden. Beispiel:

```
(gdb) bt
#0  0x00007ffff771ec6c in apr_cpystrn (dst=0x2bf <error: Cannot access memory at address 0x2bf>,
    src=0x7ffff7e2c730 "H7KCoP8pMmj44fGd1piM+zOviUT0whXO", dst_size=<optimized out>) at
    /tmp/buildd/apr-1.5.1/strings/apr_cpystrn.c:59
#1  0x00007ffff321e34a in mshield_session_create (session=0x7fffffffe6e0, is_new_session=true)
    at mod_mshield_session.c:129
#2  0x00007ffff322bad5 in mshield_access_checker (r=0x7ffff7e280a0) at mod_mshield.c:258
#3  0x00005555555a4800 in ap_run_access_checker (r=0x7ffff7e280a0) at request.c:87
#4  0x00005555555a76a8 in ap_process_request_internal (r=0x7ffff7e280a0) at request.c:265
#5  0x00005555555c4aa0 in ap_process_async_request (r=0x7ffff7e280a0) at http_request.c:315
#6  0x00005555555c4c50 in ap_process_request (r=0x7ffff7e280a0) at http_request.c:363
#7  0x00005555555c1552 in ap_process_http_sync_connection (c=0x7ffff7e4d290) at http_core.c:190
#8  ap_process_http_connection (c=0x7ffff7e4d290) at http_core.c:231
#9  0x00005555555b7f40 in ap_run_process_connection (c=0x7ffff7e4d290) at connection.c:41
#10 0x00007ffff343e7ba in child_main (child_num_arg=703) at prefork.c:704
#11 0x00007ffff343e9c7 in make_child (s=0x7ffff7fc1de0, slot=0) at prefork.c:746
#12 0x00007ffff343f70d in prefork_run (_pconf=0x5555557f6f38 <ap_server_conf>,
    plog=0x7ffff7fbb028, s=0x7ffff7fc1de0) at prefork.c:956
#13 0x00005555555927ee in ap_run_mpm (pconf=0x7ffff7fe7028, plog=0x7ffff7fbb028,
    s=0x7ffff7fc1de0) at mpm_common.c:94
#14 0x000055555558b5f3 in main (argc=2, argv=0x7fffffffec68) at main.c:777
```

Abbildung 26 Stack nach aufgetretenem Segmentation Fault

In diesem konkreten Beispiel, war der Fehler, dass in mod_mshield_session.c auf Zeile 129 (Funktion: «*mshield_session_create*») eine neue UUID generiert und diese nach «*session->data->uuid*» kopiert wurde.

```
1    apr_cpystrn(session->data->uuid, generate_uuid(session), sizeof(session->data->uuid));
```

Abbildung 27 apr_cpystrn auf nicht allozierten Pointer

«*session->data->uuid*» wiederum war jedoch fälschlicherweise noch nicht initialisiert, was zu einem Segmentation Fault führte und der entsprechende Apache Worker somit abstützte. Das Problem konnte dadurch behoben werden, indem die Session im Vorhinein korrekt initialisiert wird. In der Funktion «*mshield_session_init*», welche vor «*mshield_session_create*» in «mshield_access_checker» aufgefunden wird, musste für die Session der entsprechende Speicher alloziert werden (siehe Zeile 3 in Abbildung 28).

```
1  void
2  mshield_session_init(session_t *session, request_rec *r, mod_mshield_server_t *config) {
3      session_data_t *data = apr_pcalloc(r->pool, sizeof(session_data_t));
4      session->handle = INVALID_SESSION_HANDLE;
5      session->data = data;
6      session->request = r;
7      session->config = config;
8  }
```

Abbildung 28 Session Speicher Allozierung

Ein weiterer Segmentation Fault wurde mit Hilfe von Valgrind[79] gefunden. Valgrind ist ein Framework, welches erlaubt dynamische Analysetools laufen zu lassen. Dadurch können zum Beispiel Fehler im Memory Management oder Threading Bugs entdeckt werden. Im Fall von mod_mshield, wurde Valgrind ohne spezifisches Tool verwendet (Parameter: «*--tool=none*»).

Command: «*valgrind --trace-children=yes --trace-syscalls=yes --tool=none \ /usr/sbin/apachectl -DFOREGROUND -k start -e debug -X*»

Nachdem Apache im Debug Mode innerhalb von Valgrind gestartet wurde, konnte mit dem Browser auf die Webapplikation via Reverse Proxy (mod_mshield) normal zugegriffen werden. Das Interessante bei diesem Segmentation Fault war, dass er im normalen Betrieb von mod_mshield nicht aufgetaucht war, sondern stattdessen erst beim Beenden von Apache. Wurde der Apache Prozess innerhalb von Valgrind mittels «*Ctrl+C*» beendet, so kam der Segmentation Fault zum Vorschein.

[79] (Valgrind Developers, 2017)

```
==406== Process terminating with default action of signal 11 (SIGSEGV)
==406==  Access not within mapped region at address 0x100
==406==    at 0x95D3D73: kafka_cleanup (mod_mshield_kafka.c:476)
==406==    by 0x50D89BD: apr_pool_destroy (in /usr/lib/x86_64-linux-gnu/libapr-1.so.0.5.1)
==406==    by 0x93C020D: clean_child_exit (prefork.c:218)
==406==    by 0x93C0695: child_main (prefork.c:725)
==406==    by 0x93C09C6: make_child (prefork.c:746)
==406==    by 0x93C170C: prefork_run (prefork.c:956)
==406==    by 0x1467ED: ap_run_mpm (mpm_common.c:94)
==406==    by 0x13F5F2: main (main.c:777)
```

Abbildung 29 Kafka Cleanup Segmentation Fault

Verursacht wurde dieser Segmentation Fault durch einen Zugriff auf «*s->module_config*» innerhalb der «*kafka_cleanup*» Funktion, obwohl der Speicher von «*s*» (server_rec) bereits freigegeben wurde. Die «*kafka_cleanup*» Funktion wird mittels der APR Pools Funktion «*apr_pool_cleanup_register*» registriert, um bei einem Speicherpool Cleanup ausgeführt zu werden.

```
apr_status_t kafka_cleanup(void *arg) {
    server_rec *s = arg;
    mod_mshield_server_t *config;
    config = ap_get_module_config(s->module_config, &mshield_module);
    ...
```

Abbildung 30 Zugriff auf freigegebenen Struct (s->module_config)

Behoben werden konnte das Problem, indem nicht via «*s->module_config*» auf die Modul Konfiguration zugegriffen wird, sondern via «**arg*».

```
apr_status_t kafka_cleanup(void *arg) {
  mod_mshield_server_t *config = (mod_mshield_server_t *)arg;
  if (!config) {
    return APR_SUCCESS;
  }

  apr_pool_t *p = config->pool;
  if (!p) {
    return APR_SUCCESS;
  }
  MSHIELD_LOG_DEBUG(p, "kafka_cleanup");
  ...
```

Abbildung 31 Kafka Cleanup Segmentation Fault Fix

Dies wäre auch von Anfang an der richtige Weg gewesen, da die Server Konfiguration in «*apr_pool_cleanup_register*» bereits «*kafka_cleanup*» als «**arg*» beziehungsweise «**data*» mitgegeben wird (siehe Abbildung 33 Zeile 11).

```
1    apr_pool_cleanup_register(p, config, kafka_cleanup, kafka_cleanup);
```

Abbildung 32 Registrierung von kafka_cleanup

```
1   /**
2    * Register a function to be called when a pool is cleared or destroyed
3    * @param p The pool to register the cleanup with
4    * @param data The data to pass to the cleanup function.
5    * @param plain_cleanup The function to call when the pool is cleared
6    *                      or destroyed
7    * @param child_cleanup The function to call when a child process is about
8    *                      to exec - this function is called in the child, obviously!
9    */
10  APR_DECLARE(void) apr_pool_cleanup_register(
11                              apr_pool_t *p, const void *data,
12                              apr_status_t (*plain_cleanup)(void *),
13                              apr_status_t (*child_cleanup)(void *))
14                  __attribute__((nonnull(3,4)));
```

Abbildung 33 apr_pool_cleanup_register Dokumentation

III.8.2. Integration Tests

Sowohl für Kafka-Streams als auch für Flink existiert momentan leider keine einfache Möglichkeit um Unit-Tests zu erstellen. Für Kafka-Streams findet sich zwar ein Blogpost[80] mit einer Scala-Library, welche es unterstützt Producer und Consumer zu mocken. Es wurde jedoch entschieden vorerst auf den Einsatz dieser Library zu verzichten, da momentan vom Einsatz eines Scala Test Frameworks abgesehen werden wollte. In der Dokumentation von Apache Kafka Streams[81] finden sich dafür Beispiele für Integration-Tests, welche interne Instanzen von allen beteiligten Komponenten beinhalten. Aus diesem Grund wurde entschieden die wichtigsten Fälle der Aggregationen in Integrationstests abzubilden. Dafür konnte zu einem grossen Teil auf die bereits existierenden Klassen zurückgegriffen werden, jedoch musste zum Beispiel die Klasse IntegrationTestUtils angepasst werden, um die ganze benötigte Funktionalität zur Verfügung zu stellen.

III.8.3. Automatische Testverfahren (CI)

Um nach jedem Commit zu überprüfen, ob das Projekt noch gebuildet werden kann und allfällige Tests erfolgreich durchlaufen, wurde auf Continious Integration Testing gesetzt. Da der Jira-Stack mit BitBucket Pipelines[82] gleich eine CI-Testing Umgebung zur Verfügung stellt, wurde auch gleich diese verwendet. Im Hintergrund kann bei Pipelines mit beliebigen Docker Images gearbeitet werden, was das CI-Testing um einiges einfacher macht.

So wurde zum Beispiel für das Apache Reverse Proxy Modul mod_mshield ein eigenes Docker Image erstellt und auf den Docker Hub geladen. Innerhalb des Pipelines Konfigurations-YAML-Files, muss lediglich auf dieses Docker Image verwiesen und das Make Recipe angegeben werden, welches den Code builded.

```
1  image: pschmid/apache_module_compiler:latest
2
3  pipelines:
4    default:
5      - step:
6          script:
7            - make dev
```

Diagramm 16 mod_mshield Pipelines Konfiguration

Analog dazu wurde für den Architektur Prototyp ein Pipelines Konfigurationsfile angelegt, welches die Tests ausführt.

[80] (Poloczek, 2017)
[81] (Confluent Inc.)
[82] (Atlassian, 2017)

```
image: maven:3.3.9

pipelines:
  default:
    - step:
        script:
          - mvn test
```

Diagramm 17 Architektur Prototyp Pipelines Konfiguration

Um im Fall eines ausgetretenen Fehlers in der CI Pipeline informiert zu werden, wurden die CI Build Prozesse gleich in das MarkovShield Slack[83] Team integriert. So ist das MarkovShield Team stets über die aktuelle Kompilierungs- und korrekte Lauffähigkeit informiert.

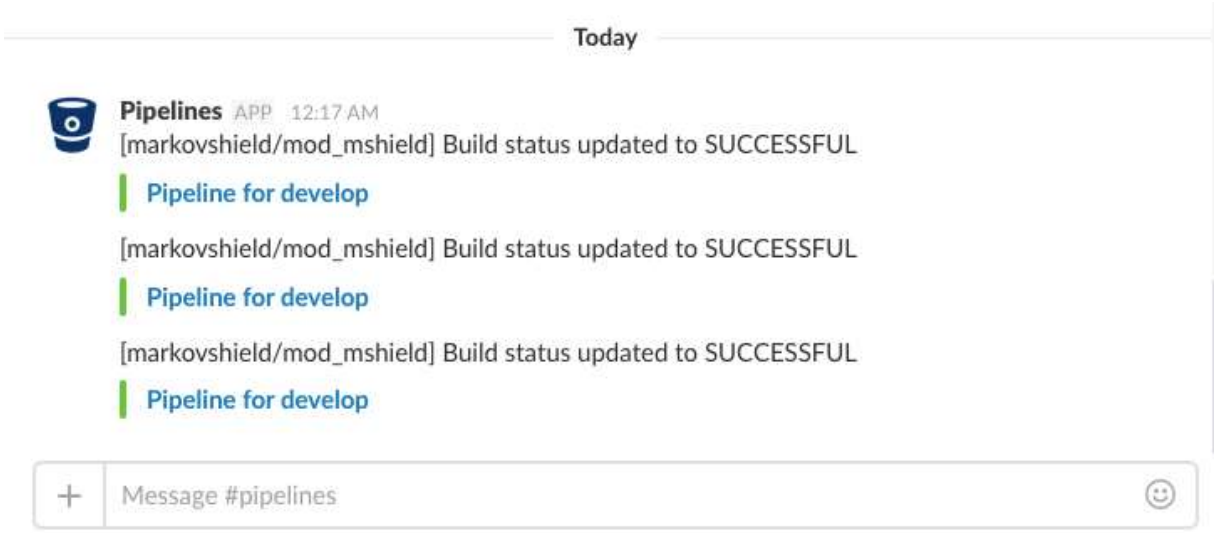


Diagramm 18 Pipelines Slack Integration

III.8.4. Manuelle Testverfahren (Systemtests etc.)

Neben den automatisierten Testverfahren wurden auch manuell durchzuführende Systemtests definiert und durchgeführt. Dies aus dem Grund, da die gesamte MarkovShield Funktionalität mittels automatisierten Integrationstests nur in einem sehr komplexen Zusammenspiel getestet werden könnte. Durch die Systemtests wird ermöglicht, dass die Funktionalität durch alle Architekturschichten durchdringend manuell getestet werden kann. Ein Systemtestprotokoll das während dem Durchführen der Systemtests ausgefüllt wird, ermöglicht zu einem späteren Zeitpunkt genau nachvollziehen zu können, wie und was zum Durchführungszeitpunkt funktioniert hatte und was nicht. Der Ablauf der Systemtests, die Systemtestspezifikation, wird im Kapitel III.10 Systemtestspezifikation behandelt.

[83] (Slack, 2017)

III.9. Deployment

Beim Deployment von MarkovShield gibt es grundsätzlich mehrere Punkte zu beachten, welche insbesondere für den produktiven Einsatz von grosser Relevanz sind.

III.9.1. Deployment-Komponenten

In der folgenden Tabelle soll aufgezeigt werden, welche Komponenten für ein vollständiges MarkovShield Deployment benötigt werden, was ihre Funktionen dabei sind und mit welchen anderen Komponenten sie kommunizieren.

Komponente	**Funktion**
Reverse Proxy	Damit ist ein Apache HTTPD Reverse Proxy mit mod_mshield und diversen weiteren Modulen aktiviert (mod_proxy, mod_rewrite, mod_headers, etc.) gemeint. Erster Auftreffpunkt für Requests, welche vom Benutzer auf die Webapplikation gesendet werden. Verantwortlich für das Extrahieren von Request Informationen und die Entscheidung, ob ein Request kritisch ist oder nicht. Behaltet kritische Requests so lange zurück, bis die Engine ein Resultat liefert oder ein Timeout auftritt. Kommuniziert direkt mit dem Benutzer, Apache Kafka und Redis.
Webapplikation (Backend)	Ist die durch MarkovShield zu schützende Komponente. Requests erreichen die Webapplikation erst, nachdem die vom Reverse Proxy durchgelassen wurden. Kommuniziert indirekt via Reverse Proxy mit dem Benutzer.
Kafka Broker	Verantwortlich für das Abspeichern der ClickEntires, Clickstreams und Auswertungsresultate. Unterteilt diese Daten in Topics. Kommuniziert mit dem Reverse Proxy, Zookeeper und Flink.

Zookeeper	Wird von Kafka zwingend gebraucht und ist sehr eng an Kafka gekoppelt. Kafka speichert diverse Konfigurationen in Zookeeper ab, um diese allen Kafka Brokers zur Verfügung zu stellen (Distributed Key-/Value-Store). Kommuniziert ausschliesslich mit Kafka.
Flink JobManager	Verantwortliche Komponente, um Flink Jobs über ein Cluster von TaskManager zu verteilen und zu verwalten. Kommuniziert mit dem Flink TaskManager.
Flink TaskManager	Flink Komponente, welche den eigentlichen Flink Job bearbeitet. Kommuniziert mit dem Flink JobManager, Kafka Broker und Redis.
Redis	Key-/Value-Store, über welchen die Resultate von Flink an mod_mshield im Reverse Proxy weitergegeben werden.
Kafka Streams Application	Java Applikation mit eingebundener Kafka Streams Library, welche die ganze Daten Aggregierung auf Kafka übernimmt und Clickstreams zur Auswertung an Flink sendet. Kommuniziert mit Kafka und Flink.
Flink Analyser Applikation	Auf Java basierender Flink Job, welcher die Clickstreams in Realtime auswertet. Kommuniziert mit Redis.
Flink ModelUpdater Applikation	Auf Java basierender Flink Job, welcher die User Models in definierbaren Abständen aktualisiert. Kommuniziert mit Kafka.

Tabelle 13 Deployment Komponenten

III.9.2. Demonstrations-Applikation

Für den einfachen Demonstrationszweck und als Einstiegspunkt in MarkovShield wurde eine Demo-Applikation erstellt, welche im Hintergrund das volle Setup laufen hat. Die gesamte Demo ist in Docker lauffähig und wird mittels einem einfachen Docker Compose Files gestartet. Die Applikation selbst, d.h. die Inhalte der Webseite, wurden von der Live CD des Hacking-Labs übernommen. Mittels dieser Inhalte wurde bereits ein Setup aufgebaut, welches die Pre-Authentication eines Benutzers über MOD_BUT demonstriert.

Eine Kurzanleitung zur Demo Applikation und das benötigte Docker Compose File können auf Github im «Standalone-Demo» Repository gefunden werden (https://github.com/MarkovShield/standalone-demo).

III.9.3. Apache Multi-Processing Modul

Der Apache Webserver bietet von Haus aus mehrere Modelle an, mit welchen Requests abgearbeitet werden können. Standardmässig stellt Apache für Linux drei dieser sogenannten Multi-Processing-Modules (MPMs)[84] zur Auswahl, wobei sich der Webserver Administrator im Betrieb dann für eines davon entscheiden muss. Diese drei MPMs sind Prefork, Worker und Event, wobei Prefork bei den meisten Linux Apache Webserver Packages standardmässig vorkonfiguriert ist.

Bei Apache Versionen vor 2.4, musste das gewählte MPM bereits beim Kompilieren des Webservers mitangegeben werden. Dies führte dazu, dass bei einem Wechsel des MPMs Apache neu kompiliert werden musste. Seit der Apache Version 2.4 ist dies jedoch nicht mehr nötig sollte das MPM gewechselt werden wollen, da die MPMs nun als Dynamic Shared Object (DSO) Modul beliebig aktiviert und deaktiviert werden können.

Bei der Wahl des richtigen MPM ist es relevant zu wissen, welche Module eingesetzt werden und welche Module besondere Anforderungen an das zu Grunde liegende MPM haben. So zum Beispiel gibt es Module wie mod_php[85], welche standardmässig nicht thread-safe sind und somit nicht ohne Weiteres mittels dem MPM Worker oder Event ausgeführt werden können.

[84] (The Apache Software Foundation, 2017)
[85] (webtatic.com, 2017)

Die Funktionsweise der 3 Haupt-MPMs unterscheidet sich wie folgt:

Prefork[86]: Die Worker Prozesse, welche die Requests abarbeiten, werden im Voraus geforkt. Während des Betriebes wird die Anzahl der Worker Prozesse dynamisch der aktuellen Last angepasst wobei über Konfigurationsdirektiven Limits gesetzt werden können. Jeder Worker Prozess kann nur ein Request gleichzeitig bearbeiten.

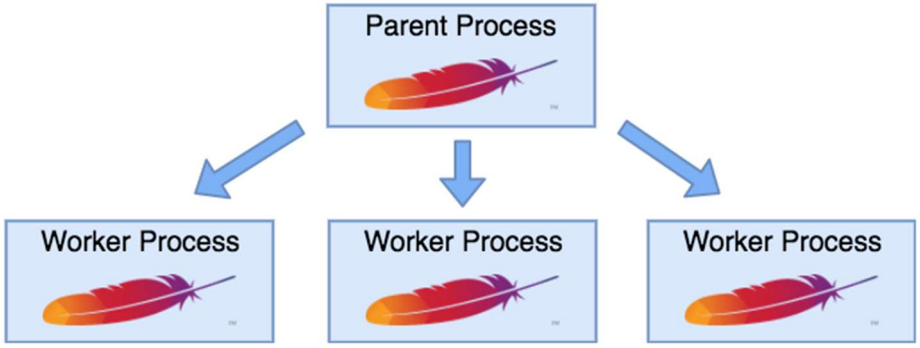


Abbildung 34 Apache Prefork MPM mit 3 Worker Prozessen

Worker[87]: Dieses MPM implementiert ein Hybrid-Server, welcher mittels Worker Prozessen und zusätzlich Server Threads arbeitet. Jeder Worker Prozess stellt eine fixe Anzahl Server Threads zur Verfügung. Zusätzlich erstellt jeder Worker Prozess ein Listener Thread (in Abbildung 35 rot abgebildet), welcher die ankommenden Verbindungen entgegennimmt und einem Server Thread weitergibt. Durch dieses Verfahren kann die Stabilität des Webservers sichergestellt und Systemressourcen können gespart werden.

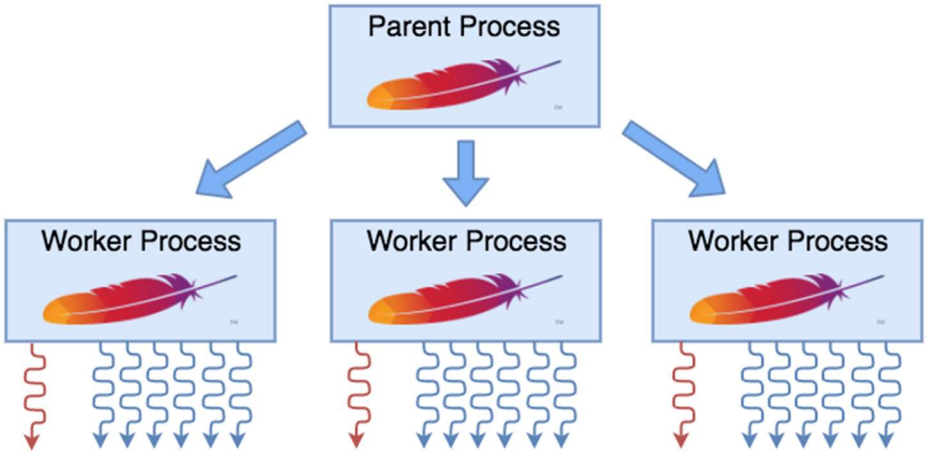


Abbildung 35 Apache Worker MPM mit 6 Server Threads und 1 Listener Thread

[86] (The Apache Software Foundation, 2017)
[87] (The Apache Software Foundation, 2017)

Event[88]: Das Event MPM basiert auf dem Worker MPM und unterscheidet sich aber zusätzlich dadurch, dass es probiert das «Keep alive»-Problem in HTTP zu lösen. Ein Client kann die Verbindung offenhalten, in dem er weitere Requests an den gleichen Socket sendet. Da Apache im Prefork MPM Modus pro aktiver Client Verbindung ein Prozess und im Worker MPM Modus ein Thread zuteilt, bleibt dieser Prozess respektive Thread im Waiting Mode und kann in dieser Zeit keine anderen Request abarbeiten. Das Event MPM löst dieses Problem nun indem es pro Worker Prozess einen dedizierten Thread (in Abbildung 36 grün abgebildet) erstellt, welcher für die «Keep alive»-Verbindungen verantwortlich ist. Somit sind die Server Threads nicht durch «Keep alive»-Verbindungen blockiert und können schnell wieder den nächsten Request abarbeiten. Der dedizierte «Keep alive»-Thread arbeitet mittels der neuen non-blocking Socket Architektur der Apache Portable Runtime (APR)[89] und ist somit in der Lage, viele «Keep alive»-Verbindungen gleichzeitig zu unterhalten.

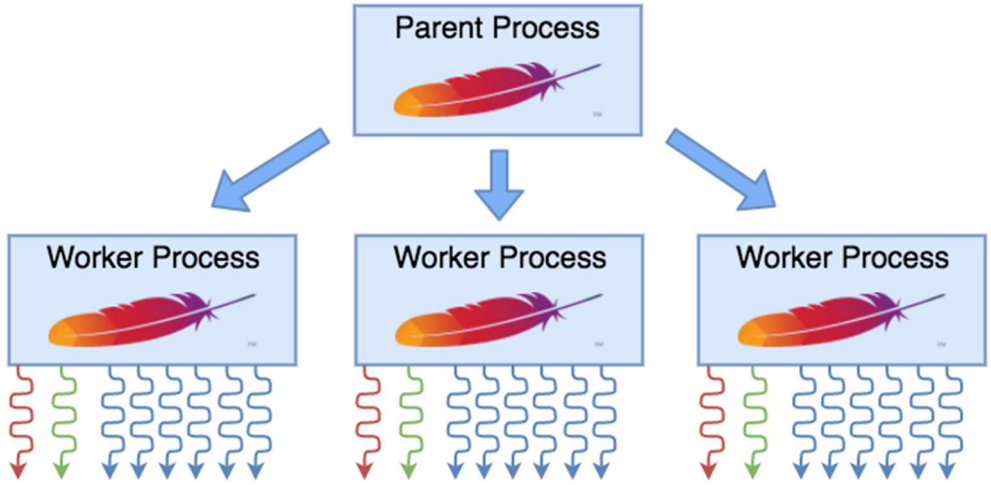


Abbildung 36 Apache Event MPM mit 6 Worker Threads, 1 Listener Thread und 1 «Keep alive»-Thread

Da mod_mshield insbesondere bei kritischen Requests eine gewisse Zeit für die Auswertung benötigt, sollte der Apache Server mittels des Event MPM betrieben werden. Würde mod_mshield in Kombination mit dem Apache Prefork MPM betrieben werden, so wäre die Requestanzahl pro Sekunde relativ stark beschränkt. Das Ganze würde zwar noch von den «Keep alive»-Zeiten der Client Browser abhängig sein, jedoch wenn MarkovShield für die Auswertung kritischer Requests in etwa 100-150ms benötigen würde, wären mit drei Apache Worker Prozessen nur rund 20 - 30 Requests pro Sekunde möglich.

[88] (The Apache Software Foundation, 2017)
[89] (The Apache Software Foundation, 2017)

III.9.4. Produktives Deployment

Sollte MarkovShield im produktiven Umfeld eingesetzt werden, so kann dies bis zu einer Grösse von rund 200 gleichzeitigen Benutzern ohne grössere Probleme und Umstellungen realisiert werden. Muss MarkovShield eine Webapplikation schützen, welche mehr als durchschnittlich 200 gleichzeitige Benutzer hat, so sollte zuerst evaluiert werden, ob es eine Skalierung der einzelnen MarkovShield Komponenten braucht oder nicht.

Grundsätzlich wird der Einfachheits- und Wartbarkeitshalber empfohlen, das MarkovShield Backend - die Engine - in einem Docker Setup zu deployen. Der Reverse Proxy sollte wiederum aus Performancegründen auf einem eigenen Server betrieben werden und nativ installiert sein. Zugleich ist beim Reverse Proxy wichtig, dass der Apache Service mit dem Event Multi-Processing Modul aktiviert ausgeführt wird.

III.9.4.1. Konfiguration

Die vollständigen Konfigurations-Anleitungen und -Einstellungen der MarkovShield Komponenten sind jeweils im Github Repository der jeweiligen Komponente als Markdown Files ersichtlich. So zum Beispiel wurden die einzelnen Parameter der Engine Applikationen und die Apache Konfigurationsdirektiven des mod_mshield Apache Modules direkt im entsprechenden Repository ausführlich beschrieben.

Im Rahmen der mod_mshield Entwicklung sind zusätzlich zu den bereits bestehenden Apache Direktiven von MOD_BUT die folgenden Konfigurationsdirektiven dazu gekommen:

Konfigurationsdirektive	**Beschreibung**
MOD_MSHIELD_FRAUD_DETECTION_ENABLED	Aktiviert oder deaktiviert die MarkovShield Funktionalitäten. Falls deaktiviert, ist die Funktion identisch wie die von MOD_BUT.
MOD_MSHIELD_FRAUD_LEARNING_MODE	Aktiviert oder deaktiviert den Learning Modus, in welchem nur Request Informationen an Apache Kafka gesendet werden, aber bei kritischen URLs keine Request Bewertung durchgeführt wird. Kritische Requests werden nicht zurückgehalten, sondern direkt an die Webapplikation durchgelassen.
MOD_MSHIELD_FRAUD_VALIDATION_THRESHOLD	Grenzwert, welcher zwischen 0 - 1000 frei gewählt werden kann. Bestimmt ab welchem Risk Level eine URL als kritisch gilt und somit ob ein Request auf diese URL zurückgehalten und bewertet werden muss.
MOD_MSHIELD_FRAUD_DETECTED_URL	URL auf welche ein Request weitergeleitet werden soll, sollte von MarkovShield ein FRAUD detektiert worden sein.
MOD_MSHIELD_FRAUD_ERROR_URL	URL auf welche der Request weitergeleitet werden soll, sollte es innerhalb der MarkovShield Engine zu einem Error kommen und somit keine Request Daten extrahiert werden könnten oder kein Resultat von der Engine zurückkommt.

MOD_MSHIELD_KAFKA_BROKER	Kafka Broker und Port, an welchen die Request Informationen gesendet werden sollen. Syntax: *Hostname:Port*
MOD_MSHIELD_KAFKA_TOPIC_ANALYSE	Kafka Topic an welches die einzelnen Click (Request) Informationen gesendet werden sollen.
MOD_MSHIELD_KAFKA_TOPIC_USER-MAPPING	Kafka Topic an welches beim Login des Benutzers das Mapping zwischen Benutzername und UUID gesendet wird.
MOD_MSHIELD_KAFKA_TOPIC_URL_CONFIG	Kafka Topic an welches beim Start vom Apache Server die aktuelle mod_mshield URL Konfiguration gesendet werden soll. Diese Direktive ist für künftige Erweiterungen vorgesehen.
MOD_MSHIELD_KAFKA_MSG_DELIVERY_TIMEOUT	Die Zeit in Sekunden, die maximal gewartet wird, bis der Kafka Broker den Erhalt der extrahierten Request Informationen bestätigt hat. Wird diese Zeit überschritten, wird der Benutzer an MOD_MSHIELD _FRAUD_ERROR_URL weitergeleitet.
MOD_MSHIELD_KAFKA_DELIVERY_CHECK_INTERVAL	Zeit in Nanosekunden, welche zwischen dem Überprüfen des Kafka Delivery Reports gewartet wird (sleep). Der Kafka Delivery Report vom Kafka Broker bestätigt den Erhalt der Message.
MOD_MSHIELD_REDIS_SERVER	Hostname oder IP Adresse der Redis Instanz, über welche das Engine Auswertungsresultat bezogen werden kann.
MOD_MSHIELD_REDIS_PORT	Port der Redis Instanz, über welche das Engine Auswertungsresultat bezogen werden kann.
MOD_MSHIELD_REDIS_RESPONSE_TIMEOUT	Zeit in Sekunden, wie lange maximal gewartet werden soll, bis von der Engine via Redis ein Auswertungsresultat eingetroffen sein muss.

MOD_MSHIELD_REDIS_CONNECTION_TIMEOUT	Zeit in Sekunden, welche maximal probiert wird, eine Verbindung zu Redis aufzubauen.
MOD_MSHIELD_URL	Definiert zu einer URL ein Risk Level. Kann beliebig viel in der Apache Konfiguration definiert werden. Es muss selbständig überprüft werden, dass sich RegEx Patterns nicht überschneiden. Syntax: *MOD_MSHIELD_URL «^/URL*» X* *X* steht für ein Risk Level zwischen 0 und 1000

Tabelle 14 Konfigurationsdirektiven mod_mshield

Für die verschiedenen Applikationsteile existieren unterschiedliche Konfigurationsparameter, welche über die Kommandozeile übergeben werden können. Ein Beispiel ist im Kapitel III.8.1.4 Konfiguration aufgeführt.

MarkovShieldClickstreams

Parameter	**Beschreibung**
help	Zeigt eine Hilfenachricht an
bootstrap	Der Hostname und der Port des Kafka Brokers
zookeeper	Der Hostname und der Port der Zookeeper Instanz
numthreads	Die Anzahl der Threads, welche Kafka Streams benutzt
resthostname	Der Hostname des REST-Endpoints
restport	Der Port des REST Endpoints

Tabelle 15 Konfigurationsparameter MarkovShieldClickstreams

MarkovShieldModelUpdater

Parameter	Beschreibung
help	Zeigt eine Hilfenachricht an
bootstrap	Der Hostname und der Port des Kafka Brokers
lookbackperiod	Die Lookbackperiod definiert, wie lange ein Clickstream in die Berechnungen der ClickstreamModels einfliesst.
updateinterval	Das Updateintervall definiert, wie oft die ClickstreamModels neu berechnet werden sollen.
sessiontimeout	Der Session Timeout definiert, nach wie langer Zeit eine Session bei Inaktivität eines Benutzers beendet wird.

Tabelle 16 Konfigurationsparameter MarkovShieldModelUpdater

MarkovShieldAnalyser

Parameter	Beschreibung
help	Zeigt eine Hilfenachricht an
bootstrap	Der Hostname und der Port des Kafka Brokers
redishost	Der Hostname der Redis-Instanz
redisport	Der Port der Redis-Instanz

Tabelle 17 Konfigurationsparameter MarkovShieldAnalyser

III.10. Systemtestspezifikation

Die nachfolgenden Systemtests prüfen die Richtigkeit der MarkovShield Funktionalitäten, welche in den vorgängigen Kapiteln der Projektdokumentation aufgezeigt wurden.

Für die Durchführung der Systemtests wurde zugleich ein Systemtestprotokoll erstellt, in welchem alle Zustände, abgesetzten Commands und aufgerufene Webseiten festgehalten werden sollten. Somit ist sichergestellt, dass zu einem späteren Zeitpunkt nachvollzogen werden kann, wie die getesteten Resultate damals ausgesehen haben und wie sie zustande gekommen sind.

III.10.1. Globale Voraussetzungen

Damit die Systemtests durchgeführt werden können, müssen die folgenden globalen Voraussetzungen im Vorhinein abgedeckt werden. Braucht ein nachfolgender Systemtest weitere Voraussetzungen, so sind diese vor dem eigentlichen Test erläutert und ergänzend zu den globalen Voraussetzungen zu betrachten.

- Die Systemtests können ausschliesslich auf macOS, Linux oder Unix Systemen durchgeführt werden. Windows wird nicht unterstützt.
- Wird in der Systemtestspezifikation auf eine Localhost URL verwiesen, so muss diese in einem Private Modus Tab im Browser aufgerufen werden. Dies stellt sicher, dass keine alten Cookie oder Caches verwendet werden.
- Werden von einem Systemtest Änderungen in einem Konfigurationsfile verlangt, müssen diese nach dem Test wieder rückgängig gemacht werden.
- Die beiden Hilfstools Kafkacat[90] und Redis-CLI[91] müssen lokal auf dem Rechner installiert sein. Mehr Informatioen zur jeweiligen Installation können unter https://github.com/edenhill/kafkacat und https://redis.io/download gefunden werden.
- Alle Systemtests sollen mit den zur Verfügung gestellten Dockerfiles und docker-compolse.yml Files durchgeführt werden. Die Dockerfiles sind im mod_mshield Git Repository unter «*examplesite/*/Dockerfile*» zu finden. Das Docker Compose File wird im mod_mshield Git Repository Root Verzeichnis zur Verfügung gestellt.
- Make, Docker[92] und Docker Compose[93] müssen lokal auf dem Rechner installiert und auf der neusten Version sein. Zudem muss der Docker Service im Hintergrund laufen.
- Es dürfen während der Testdurchläufe keine anderen Docker Container auf dem Testsystem laufen, da diese Einfluss auf die Testdurchführung haben könnten (z.B. durch bereits besetzte Ports).
- Das mod_mshield Git Repository muss lokal vorhanden sein und es muss der Develop Branch ausgecheckt sein (Command: «*git checkout develop*»). Es ist sicherzustellen, dass die neusten Änderungen vom Develop Branch lokal vorhanden sind (Command: «*git pull*»).
- Nach jedem Testablauf müssen alle Docker Container gestoppt werden damit für den nächsten Testablauf wieder saubere Voraussetzungen vorhanden sind. Die Docker Container können allesamt am Ende eines Testes mittels des Commands «*make shutdown-demo*», im mod_mshield Git Repository Root Verzeichnis ausgeführt, heruntergefahren und entfernt werden. Anschliessend ist jeweils mit «*docker ps*» zu überprüfen, ob wirklich alle Container gestoppt wurden. Falls doch noch ein Container läuft, so muss dieser manuell mittels «*docker stop <container-id>*« gestoppt werden.

[90] (Edenhill, 2017)
[91] (redislabs, 2017)
[92] (Docker Inc., 2017)
[93] (Docker Inc., 2017)

III.10.2. Demo Deployment

III.10.2.1. Voraussetzungen

Für die folgenden Systemtests sind ergänzend zu den globalen Voraussetzungen keine weiteren Voraussetzungen vorhanden.

III.10.2.2. Testablauf Make Demo Target Korrektheit

Nr.	Titel	Handlung	Reaktion
1	MarkovShield Demostack hochfahren	1. Im mod_mshield Verzeichnis muss «*make demo*» ausgeführt werden damit eine vollständig funktionierende Demo Applikation gestartet wird.	Nun sollten im Hintergrund alle Docker Container gestartet sein. Dies kann mittels «*docker ps*» überprüft werden.
2	Docker Container Vollständigkeit überprüfen	1. Mittels «*docker ps*» kann ausgegeben werden, welche Docker Container zurzeit gestartet sind.	Die folgenden Docker Container müssen aufgelistet sein: • mshielddemo_mshield_flink_modelupdater_1 • mshielddemo_mshield_flink_analyser_1 • mshielddemo_mshield_kafka_Clickstreams_1 • mshielddemo_taskmanager_1 • mshielddemo_jobmanager_1 • mshielddemo_broker_1 • mshielddemo_mshield_backend_1 • mshielddemo_mshield_reverse_proxy_1 • mshielddemo_zookeeper_1 • mshielddemo_redis_1

III.10.2.3. Testablauf Zugriff auf Header Informationen

Nr.	Titel	Handlung	Reaktion
1	MarkovShield Demostack hochfahren	1. Im mod_mshield Verzeichnis muss «*make demo*» ausgeführt werden damit eine vollständig funktionierende Demo Applikation gestartet wird.	Nun sollten im Hintergrund alle Docker Container gestartet sein.
2	Docker Container Vollständigkeit überprüfen	1. Mittels «*docker ps*» kann ausgegeben werden, welche Docker Container zurzeit alle gestartet sind.	Die folgenden Docker Container müssen aufgelistet sein: • mshielddemo_mshield_flink_modelupdater_1 • mshielddemo_mshield_flink_analyser_1 • mshielddemo_mshield_kafka_Clickstreams_1 • mshielddemo_taskmanager_1 • mshielddemo_jobmanager_1 • mshielddemo_broker_1 • mshielddemo_mshield_backend_1 • mshielddemo_mshield_reverse_proxy_1 • mshielddemo_zookeeper_1 • mshielddemo_redis_1
3	High Risk Level URL aufrufen	1. Im Browser muss https://localhost/ -> «Pre-Auth Demo» -> «Echo Request Header» aufgerufen und anschliessend eingeloggt werden. Nun muss auf den Menupunkt «Echo Request Header» geklickt werden.	1. Im Browser muss ein Request Header in tabellarischer Form dargestellt sein. Der Wert «Host» muss «mshield_backend:8888» lautet.

III.10.3. Systemtests Reverse Proxy Konfiguration

III.10.3.1. Voraussetzungen

Die folgenden Voraussetzungen müssen zusätzlich erfüllt sein, damit die Tests durchgeführt werden können.

- Die Reverse Proxy Konfiguration im mod_mshield Repository muss angepasst werden können.

III.10.3.2. Testablauf Free URLs

Nr.	Titel	Handlung	Reaktion
1	Free URLS in der Konfiguration setzen	1. Im File «examplesite/reverse-proxy/conf/mshield.conf» muss die Konfigurationsdirektive «*MOD_MSHIELD_SESSION_FREE_URL*» auf den Wert *'(^/error/refused_cookies\.html$)|(^/renew)|(^/img/)|(^/js/)|(^/fonts/)|(^/css/)'* (mit den einfachen Anführungszeichen) gesetzt werden. Änderung speichern.	-
2	MarkovShield Demostack hochfahren	1. Im mod_mshield Verzeichnis muss «*make demo*» ausgeführt werden damit eine vollständig funktionierende Demo Applikation gestartet wird.	Nun sollten im Hintergrund alle Docker Container gestartet sein.

3	Docker Container Vollständigkeit überprüfen	1. Mittels «*docker ps*» kann ausgegeben werden, welche Docker Container zurzeit alle gestartet sind.	Die folgenden Docker Container müssen aufgelistet sein: • mshielddemo_mshield_flink_modelupdater_1 • mshielddemo_mshield_flink_analyser_1 • mshielddemo_mshield_kafka_Clickstreams_1 • mshielddemo_taskmanager_1 • mshielddemo_jobmanager_1 • mshielddemo_broker_1 • mshielddemo_mshield_backend_1 • mshielddemo_mshield_reverse_proxy_1 • mshielddemo_zookeeper_1 • mshielddemo_redis_1
4	Free URLS testen	1. Mittels Kafkacat sollte im «*MarkovClicks*» Kafka Topic mitgehört werden. Der Befehl dazu lautet: «*kafkacat -C -b localhost -t MarkovClicks*» und sollte auf dem Testsystem ausgeführt werden.	Alle von mod_mshield extrahierten Klicks werden als JSON angezeigt.
		2. Im Browser muss https://localhost/ aufgerufen werden.	Es darf im Kafkacat **kein** neuer Click Entry mit der URL «*/img*» erschienen sein.

III.10.3.3. Testablauf URL Risk Level und Risk Threshold

Nr.	Titel	Handlung	Reaktion
1	Risk Threshold in der Konfiguration setzen	1. Im File «examplesite/reverse-proxy/conf/mshield.conf» muss die Konfigurationsdirektive «*MOD_MSHIELD_FRAUD_ VALIDATION_THRESHOLD*» (ohne Umbruch) auf den Wert «*3*» gesetzt werden. Änderung speichern.	-
2	URL Risk Levels definieren	1. Im File «examplesite/reverse-proxy/conf/*mod_mshield_URL_rating.conf*» das URL Risk Level der URL «*^/private**» auf «*4*» setzen. Änderung speichern.	-
		2. Im File «examplesite/reverseproxy/conf/*mod_mshield_URL_rating.conf*» das URL Risk Level der URL «*^/howto**» auf «*2*» setzen. Die Zeile muss anschliessend wie folgt aussehen: «*MOD_MSHIELD_URL «^/howto*» 2*». Änderung speichern.	-
3	MarkovShield Demostack hochfahren	1. Im mod_mshield Verzeichnis muss «*make demo*» ausgeführt werden damit eine vollständig funktionierende Demo Applikation gestartet wird.	Nun sollten im Hintergrund alle Docker Container gestartet sein.

4	Docker Container Vollständigkeit überprüfen	1. Mittels «*docker ps*» kann ausgegeben werden, welche Docker Container zurzeit alle gestartet sind.	Die folgenden Docker Container müssen aufgelistet sein: • mshielddemo_mshield_flink_modelupdater_1 • mshielddemo_mshield_flink_analyser_1 • mshielddemo_mshield_kafka_Clickstreams_1 • mshielddemo_taskmanager_1 • mshielddemo_jobmanager_1 • mshielddemo_broker_1 • mshielddemo_mshield_backend_1 • mshielddemo_mshield_reverse_proxy_1 • mshielddemo_zookeeper_1 • mshielddemo_redis_1
5	URL Risk Level über Grenzwert	1. Mittels Kafkacat sollte im «*MarkovClicks*» Kafka Topic mitgehört werden. Der Befehl dazu lautet: «*kafkacat -C -b localhost -t MarkovClicks*»	Alle von mod_mshield extrahierten Klicks werden als JSON angezeigt.
		2. Im Browser muss https://localhost/ -> «Pre-Auth Demo» -> «Echo Request Header» aufgerufen und anschliessend eingeloggt werden. Nun muss auf den Menupunkt «Echo Request Header» geklickt werden.	1. Im Kafkacat muss nun ein Click-Entry vorhanden sein, welcher auf die vorhin erwähnte URL zeigt und «*«validationRequired»: true*» besitzt. 2. Im Browser muss ein Request Header in tabellarischer Form dargestellt sein.

6	URL Risk Level unter Grenzwert	1. Mittels Kafkacat sollte im «*MarkovClicks*» Kafka Topic mitgehört werden. Der Befehl dazu lautet: «*kafkacat -C -b localhost -t MarkovClicks*»	Alle von mod_mshield extrahierten Klicks werden als JSON angezeigt.
		2. Im Browser muss https://localhost/howto/start-stop/ aufgerufen werden.	1. Im Kafkacat muss nun ein Click-Entry vorhanden sein, welcher auf die vorhin erwähnte URL zeigt und «*«validationRequired»: false*» besitzt. 2. Im Browser muss eine kurze Anleitung angezeigt werden, welche zeigt wie der Apache Web Server gestartet und gestoppt werden kann.

III.10.4. Systemtests Erreichbarkeit

III.10.4.1. Voraussetzungen

Zusätzlich zu den globalen Voraussetzungen müssen die folgenden spezifischen Voraussetzungen erfüllt sein, damit die Tests durchgeführt werden können.

- Die gesamte MarkovShield Stack muss bereits gestartet sein. Dazu kann im mod_mshield Git Repository Root Verzeichnis «*make demo*» ausgeführt werden. Hierbei wird ein kompletter MarkovShield Demostack hochgefahren. Zur Verifikation sollte mit «*docker ps*» überprüft werden, ob alle Container ordnungsgemäss gestartet wurden.

III.10.4.2. Testablauf Request Blockierung bei Erreichbarkeitsproblemen

Nr.	Titel	Handlung	Reaktion
1	Flink Job beenden	1. Der zuvor gestartete Flink Analyser Container soll gestoppt werden: «*docker stop mshielddemo_mshield_flink_analyser_1*»	Der Container wurde gestoppt und als Rückgabewert wurde der Container Name zurückgegeben.
		2. Nun muss der Flink Job auch noch auf dem JobManager beendet werden. Dazu kann http://localhost:48081/ aufgerufen werden und über «*Running Jobs*» -> «*Job ID*» -> «*Cancel*» muss der Job beendet werden.	Der Job hat nun den Status «*CANCELED*».
2	MarkovShield Funktionalität überprüfen	1. Im Browser muss https://localhost/ -> «Pre-Auth Demo» -> «Echo Request Header» aufgerufen werden.	Der Aufruf darf nicht erfolgreich gewesen sein und der Request muss auf «*/error/fraud_error.html*» weitergeleitet worden sein.

III.10.4.3. Testablauf Kafka nicht erreichbar

Nr.	Titel	Handlung	Reaktion
1	Kafka Docker Container beenden	1. Mittels «*docker stop mshield-demo_broker_1*» muss der Kafka Docker Container gestoppt werden, sodass die Extrahierung der Request Daten im mod_mshield fehlschlagen sollte.	Der Container wurde beendet und der Container Name wurde vom Command zurückgegeben.
2	MarkovShield Funktionalität überprüfen	1. Im Browser muss https://localhost/ aufgerufen werden.	Der Aufruf darf nicht erfolgreich gewesen sein und der Request muss auf «*/error/fraud_error.html*» weitergeleitet worden sein.

III.10.4.4. Testablauf Redis nicht erreichbar

Nr.	Titel	Handlung	Reaktion
1	Redis Docker Container beenden	1. Mittels «*docker stop mshield-demo_redis_1*» muss der Redis Docker Container gestoppt werden, sodass das Empfangen des Engine Resultates für mod_mshield fehlschlagen sollte.	Der Container wurde beendet und der Container Name wurde vom Command zurückgegeben.
2	MarkovShield Funktionalität überprüfen	1. Im Browser muss https://localhost/ -> «Pre-Auth Demo» -> «Echo Request Header» aufgerufen werden.	Der Aufruf darf nicht erfolgreich gewesen sein und der Request muss auf «*/error/fraud_error.html*» weitergeleitet worden sein.

III.10.5. Systemtests Run Modes

III.10.5.1. Voraussetzungen

Die folgenden Voraussetzungen müssen zusätzlich erfüllt sein, damit die Tests durchgeführt werden können.

- Die Reverse Proxy Konfiguration im mod_mshield Repository muss angepasst werden können.

III.10.5.2. Testablauf Fraud Detection deaktivieren

Nr.	Titel	Handlung	Reaktion
1	Fraud Detection mod_mshield Feature deaktivieren	1. Im File «examplesite/reverseproxy/conf/mshield.conf» muss die Konfigurationsdirektive «*MOD_MSHIELD_FRAUD_DETECTION_ENABLED*» auf den Wert «*Off*» gesetzt werden.	-
2	MarkovShield Demostack hochfahren	1. Im mod_mshield Git Repository Root Verzeichnis muss «*make demo*» ausgeführt werden damit eine vollständig funktionierende Demo Applikation gestartet wird. Die soeben veränderte Konfiguration wird automatisch miteinbezogen.	Nun sollten im Hintergrund alle Docker Container gestartet sein. Dies kann mittels «*docker ps*» überprüft werden.

3	Überflüssige Container stoppen	1. Die folgenden Docker Container sollten mittels «*docker stop <container-name>*« gestoppt werden, da sie ohne die Fraud Detection Funktionalität nicht mehr gebraucht werden: • mshielddemo_taskmanager_1 • mshielddemo_jobmanager_1 • mshielddemo_broker_1 • mshielddemo_redis_1 • mshielddemo_zookeeper_1 • mshielddemo_mshield_kafka_Clickstreams_1 • mshielddemo_mshield_flink_analyser_1 • mshielddemo_mshield_flink_modelupdater_1 Dies stellt zudem sicher, dass im Hintergrund von mod_mshield her ungewollte Verbindungen gemacht werden.	Es wird jeweils der Container Name als Rückgabewert auf das Command ausgegeben.
4	Demoseiten testen	1. Im Browser soll https://localhost/private/1/ aufgerufen und mittels «*Submit*» eingeloggt werden.	Es darf kein Internal Server Error auftauchen oder keine Weiterleitung auf eine Error Seite geben.
		2. Im Browser soll https://localhost/configuration/architecture/ aufgerufen werden.	Es darf kein Internal Server Error auftauchen oder keine Weiterleitung auf eine Error Seite geben.
		3. Im Browser soll https://localhost/howto/explorer/ aufgerufen werden.	Es darf kein Internal Server Error auftauchen oder keine Weiterleitung auf eine Error Seite geben.

III.10.5.3. Testablauf Learning Mode aktivieren

Nr.	Titel	Handlung	Reaktion
1	Learning Mode mod_mshield Feature aktivieren	1. Im File «examplesite/reverseproxy/conf/mshield.conf» muss die Konfigurationsdirektive «*MOD_MSHIELD_FRAUD_LEARNING_MODE*» auf den Wert «*On*» setzen.	-
2	MarkovShield Demostack hochfahren	1. Im mod_mshield Git Repository Root Verzeichnis muss «*make demo*» ausgeführt werden damit eine vollständig funktionierende Demo Applikation gestartet wird. Die soeben veränderte Konfiguration wird automatisch miteinbezogen.	Nun sollten im Hintergrund alle Docker Container gestartet sein. Dies kann mittels «*docker ps*» überprüft werden.
3	Click Entries mitlesen	1. Mittels Kafkacat sollte im «*MarkovClicks*» Kafka Topic mitgehört werden. Der Befehl dazu lautet: «*kafkacat -C -b localhost -t MarkovClicks*»	Alle von mod_mshield extrahierten Klicks werden als JSON angezeigt.
4	Demoseite mit hohem Risk Level testen	1. Im Browser muss https://localhost/ -> «Pre-Auth Demo» -> «Echo Request Header» aufgerufen werden. Bei der Login Abfrage mittels «*submit*» einloggen.	1. Im Kafkacat muss nun ein Click-Entry vorhanden sein, welcher auf die vorhin erwähnte URL zeigt und «*«validationRequired»: false*» besitzt. 2. Die aufgerufene Webseite muss geladen sein.

III.10.5.4. Testlauf Enforce Mode aktivieren

Nr.	Titel	Handlung	Reaktion
1	Fraud Detection mod_mshield Feature aktivieren	1. Im File «examplesite/reverseproxy/conf/mshield.conf» muss die Konfigurationsdirektive « «*MOD_MSHIELD_FRAUD_DETECTION_ENABLED*» auf «*On*» gesetzt werden.	-
2	MarkovShield Demostack hochfahren	1. Im mod_mshield Git Repository Root Verzeichnis muss «*make demo*» ausgeführt werden damit eine vollständig funktionierende Demo Applikation gestartet wird. Die soeben veränderte Konfiguration wird automatisch miteinbezogen.	Nun sollten im Hintergrund alle Docker Container gestartet sein. Dies kann mittels «*docker ps*» überprüft werden.
3	Click Entries mitlesen	1. Mittels Kafkacat sollte im «MarkovClicks» Kafka Topic mitgehört werden. Der Befehl dazu lautet: «*kafkacat -C -b localhost -t MarkovClicks*»	Alle von mod_mshield extrahierten Klicks werden als JSON angezeigt.
4	Demoseite mit hohem Risk Level testen	1. Im Browser muss https://localhost/ -> «Pre-Auth Demo» -> «Echo Request Header» aufgerufen werden.	1. Im Kafkacat muss nun ein Click-Entry vorhanden sein, welcher auf die vorhin erwähnte URL zeigt und «*«validationRequired»: true*» besitzt. 2. Mittels «*redis-cli psubscribe W**» können alle Engine Resultate betrachtet werden. Hier muss zum entsprechenden Request OK als Rating zurückgekommen sein. 3. Die aufgerufene Webseite muss geladen sein.

III.10.6. Systemtests Engine Resultat Rating

III.10.6.1. Voraussetzungen

Die folgenden zusätzlichen Voraussetzungen müssen erfüllt sein, damit die Tests durchgeführt werden können.

- Die Reverse Proxy Konfiguration im mod_mshield Repository muss angepasst werden können.
- Das Architektur Prototyp Git Repository muss ausgecheckt sein und auf den Develop Branch zeigen (Command: «*git checkout develop*»).

III.10.6.2. Testablauf Rating OK

Nr.	Titel	Handlung	Reaktion
1	MarkovShield Demostack hochfahren	1. Im mod_mshield Verzeichnis muss «*make demo*» ausgeführt werden damit eine vollständig funktionierende Demo Applikation gestartet wird.	Nun sollten im Hintergrund alle Docker Container gestartet sein. Dies kann mittels «*docker ps*» überprüft werden.
2	Demoseiten testen	1. Im Browser soll https://localhost/ -> «Pre-Auth Demo» -> «Echo Request Header» aufgerufen und wenn aufgefordert mittels «*Submit*» eingeloggt werden. Da aufgrund der fehlenden trainierten Modele in der Demoapplikation immer OK als Resultat von der Engine zurückkommt, wird bei diesem Test auch ein OK erwartet.	Es darf kein Internal Server Error angezeigt werden oder keine Weiterleitung auf eine Error Seite geben. Die Seite muss normal geladen werden.

III.10.6.3. Testablauf Rating SUSPICIOUS

Nr.	Titel	Handlung	Reaktion
1	MarkovShield Demostack hochfahren	1. Im mod_mshield Git Repository Root Verzeichnis muss «*make demo*» ausgeführt werden damit eine vollständig funktionierende Demo Applikation gestartet wird.	Nun sollten im Hintergrund alle Docker Container gestartet sein. Dies kann mittels «*docker ps*» überprüft werden.
2	Jobs Docker Containers beenden	1. Da im nachfolgenden Schritt zwei Jobs manuell gestartet werden, müssen vorgängig die folgenden Docker Container beendet werden. Der ModelUpdater Flink Job wird um eine Minute verzögert gestartet. Falls der ModelUpdater Container vor dem Ablauf dieser Zeit gestoppt wird, muss anschliessend im Teilschritt 2 bloss der Analyser Job gestoppt werden, da der ModelUpdater Job gar noch nicht gestartet wurde. Commands für das Beenden der Container: «*docker stop mshield-demo_mshield_flink_analyser_1*» und «*docker stop mshielddemo_mshield_flink_modelupdater_1*»	Es wird jeweils der Container Name als Rückgabewert auf das Command ausgegeben.
		2. Nun müssen die Flink Jobs auch noch auf dem JobManager beendet werden. Dazu kann http://localhost:48081/ aufgerufen werden und über «*Running Jobs*» -> «*Job ID*» -> «*Cancel*» muss jeder Job beendet werden. Hinweis: Job erscheint erst eine Minute nach dem starten der Container im Flink JobManager UI, da dieser absichtlich verzögert gestartet wird.	Die Jobs haben nun den Status «*CANCELED*».

3	SUSPICIOUS Resultat erzwingen	1. Im Architektur Prototyp Git Repository muss im Code in der Klasse «*ch.hsr.MarkovShield.flink.MarkovShieldAnalyser.java*» in der Methode «*calculateMarkovFraudLevel*» der bestehende Code durch ein «*return MarkovRating.SUSPICIOUS*» ersetzt werden.	Mit dem soeben veränderten Code wird in Zukunft immer SUSPICIOUS von der Engine zurückgegeben.
		2. Nun muss das MarkovShieldAnalyse Flink Jar erneut erstellt werden. Dazu muss im Root Verzeichnis des Architektur Prototyp Repositories «mvn clean install» ausgeführt werden.	Viel Output wird zusehen sein und alle Tests sollten durchlaufen.
4	Flink Job manuell starten	1. Anschliessend muss der Flink Job manuell gestartet werden. Dazu muss das folgende Command im Architecture Prototype Root Verzeichnis ausgeführt werden (für macOS/Linux/Unix): «*flink run -c ch.hsr.MarkovShield.flink.MarkovShieldAnalyser --jobmanager jobmanager:6123 flink/target/flink-1.0-SNAPSHOT-jar-with-dependencies.jar*»	Das Command sollte nun im Vordergrund es sollte zu sehen sein, wie der Job erfolgreich an Flink übertragen wurde.

5	Demoseiten testen	1. Unkritische URL aufrufen: https://localhost/howto/logfile/	Der Client darf nicht weitergeleitet werden und die Webseite muss normal angezeigt werden.
		2. Im Browser soll https://localhost/ aufgerufen und nach «Pre-Auth Demo» -> «Echo Response Header» manövriert werden. Beim Login muss mittels «Submit» eingeloggt werden.	1. Da aufgrund der vorherigen Änderung nun immer SUSPICIOUS von der Engine zurückkommt, wird der Client automatisch auf den Login Server 2 weitergeleitet. 2. Mittels «*redis-cli psubscribe W**» muss überprüft werden, dass für alle Requests SUSPICIOUS als Resultat zurückkommt.

III.10.6.4. Testablauf Rating FRAUD

Nr.	Titel	Handlung	Reaktion
1	MarkovShield Demostack hochfahren	1. Im mod_mshield Git Repository Root Verzeichnis muss «*make demo*» ausgeführt werden damit eine vollständig funktionierende Demo Applikation gestartet wird.	Nun sollten im Hintergrund alle Docker Container gestartet sein. Dies kann mittels «*docker ps*» überprüft werden.
2	Duplikate Docker beenden	1. Da im nachfolgenden Schritt zwei Jobs manuell gestartet werden, müssen vorgängig die folgenden Docker Container beendet werden. Der ModelUpdater Flink Job wird um eine Minute verzögert gestartet. Falls der ModelUpdater Container vor dem Ablauf dieser Zeit gestoppt wird, muss anschliessend im Teilschritt 2 bloss der Analyser Job gestoppt werden, da der ModelUpdater Job gar noch nicht gestartet wurde. Commands für das Beenden der Container: «*docker stop mshield-demo_mshield_flink_analyser_1*» und «*docker stop mshielddemo_mshield_flink_modelupdater_1*»	Es wird jeweils der Container Name als Rückgabewert auf das Command ausgegeben.
		2. Nun müssen die Flink Jobs auch noch auf dem JobManager beendet werden. Dazu kann http://localhost:48081/ aufgerufen werden und über «*Running Jobs*» -> «*Job ID*» -> «*Cancel*» muss jeder Job beendet werden. Hinweis: Job erscheint erst eine Minute nach dem starten der Container im Flink JobManager UI, da dieser absichtlich verzögert gestartet wird.	Die Jobs haben nun den Status «*CANCELED*».

3	FRAUD Resultat erzwingen	1. Im Architektur Prototyp Git Repository muss im Code in der Klasse «*ch.hsr.MarkovShield.flink.MarkovShieldAnalyser.java*» in der Methode «*calculateMarkovFraudLevel*» der bestehende Code durch ein «*return MarkovRating.FRAUD*» ersetzt werden.	Mit dem soeben veränderten Code wird in Zukunft immer FRAUD von der Engine zurückgegeben.
		2. Nun muss das MarkovShieldAnalyse Flink Jar erneut erstellt werden. Dazu muss im Root Verzeichnis des Architecture Prototyp Repositories «mvn clean install» ausgeführt werden.	Viel Output wird zusehen sein und alle Tests sollten durchlaufen.
4	Flink Job manuell starten	1. Jetzt muss der Flink Job manuell gestartet werden. Dazu muss das folgende Command im Architecture Prototype Root Verzeichnis ausgeführt werden (für macOS/Linux/Unix): «*flink run -c ch.hsr.MarkovShield.flink.MarkovShieldAnalyser --jobmanager jobmanager:6123 flink/target/flink-1.0-SNAPSHOT-jar-with-dependencies.jar*»	Das Command sollte nun im Vordergrund es sollte zu sehen sein, wie der Job erfolgreich an Flink übertragen wurde.

5	Demoseiten testen	1. Unkritische URL aufrufen: https://localhost/howto/logfile/	Der Client darf nicht weitergeleitet werden und die Webseite muss normal angezeigt werden.
		2. Im Browser soll https://localhost/ aufgerufen und nach «Pre-Auth Demo» -> «Echo Response Header» manövriert werden. Da aufgrund der vorherigen Einstellung nun immer FRAUD von der Engine zurückkommt, wird der Client auf eine Fraud Detected Error Seite umgeleitet.	1. Der Client muss auf eine Fraud Detected Error Seite weitergeleitet worden sein. 2. Mittels «*redis-cli psubscribe W**» muss überprüft werden, dass für alle Requests FRAUD als Resultat zurückkommt.
6	Neue Session erzeugen	1. Mittels des folgenden Kafkacat Befehles die «*SessionUUID*» im «*MarkovClicks*» Topic genau beobachten: «*kafkacat -C -b localhost -t MarkovClicks*». Nun auf eine kritisch URL zugreifen und auf die Fraud Detected Error Seite weitergeleitet werden. Anschliessend wieder eine neutrale URL aufrufen (z.B. https://localhost/howto/explorer/).	Die alte Session muss im Falle eines erkennten FRAUD gelöscht werden. Deshalb ist bei erneutem Zugriefen auf die unkritische URL im «*MarkovClicks*» Kafka Topic eine neue «*SessionUUID*» zusehen.

III.11. Laufzeitverhalten von MarkovShield

In diesem Kapitel geht es darum aufzuzeigen und zu beurteilen, wie das Laufzeitverhalten der MarkovShield Gesamtlösung aussieht.

III.11.1. Rahmenbedingungen

III.11.1.1. Testsystem

Der Reverse Proxy (Apache mit mod_mshield) sowohl als auch das gesamte MarkovShield Backend wurden auf dem gleichen System laufen gelassen. Das Testsystem hatte dabei die folgenden Hardware Eigenschaften.

System Typ	Workstation
Model	Fujitsu Celsius W530
CPU	Quad-Core mit Hyper-Threading aktiviert (entspricht 8 Cores) (Intel(R) Xeon(R) CPU E3-1245 v3 @ 3.40GHz)
RAM	16GB @ DDR3 1600 MHz
HDD	256GB SAMSUNG MZ7TD256

Tabelle 18 Load Test System Hardwaremerkmale

III.11.1.2. Software Version

Auf dem Testsystem wurde ein CentOS 7.3 Core mit einer Standardinstallationsrolle installiert.

Betriebssystem	CentOS Linux release 7.3.1611 (Core)
Kernel	3.10.0-514.21.1.el7.x86_64
Docker	Docker version 17.03.1-ce, build c6d412e
Docker Compose	docker-compose version 1.13.0, build 1719ceb
MarkovShield Engine	2.0, build b0d32ec
MarkovShield mod_mshield	2.0, build 79183be
Apache	Apache/2.4.6 (CentOS) mit Event MPM

Tabelle 19 Eingesetzte Software Versionen

Der Einfachheit halber wurde die Systemfirewall und SELinux[94] deaktiviert, um mögliche Einflüsse minimieren zu können Der Apache Webserver selbst wurde nativ auf dem System installiert betrieben und die restlichen MarkovShield Komponenten in Docker Container - auch das Backend.

Die MarkovShield Komponenten wurden über das Docker Compose File von der MarkovShield One-Command standalone Demoapplikation gestartet. Die einzigen zwei Anpassungen am Docker Compose File waren, dass der Reverse Proxy auskommentiert wurde und der Port 8888 vom Backend nicht bloss Docker intern «exposed» wurde, sondern dass dieser auf das Hostsystem freigegeben wurde (siehe Zeile 14 in Abbildung 37). Die Änderung am Backend Port war nötig, weil nur so von extern her darauf zugegriffen werden konnte (vom nativ installieren Apache Webserver).

```
1   version: '3'
2   services:
3     #mshield_reverse_proxy:
4     #  image: markovshield/mshield-demo-reverse-proxy:2.0
5     #  hostname: mshield_reverse_proxy
6     #  ports:
7     #    - "80:80"
8     #    - "443:443"
9
10    mshield_backend:
11      image: markovshield/mshield-demo-backend:2.0
12      hostname: mshield_backend
13      ports:
14        - "127.0.0.1:8888:8888"
15  ...
```

Abbildung 37 Docker Compose File Anpassungen für den Load Test

Als die zu testende Webseite wurde die bestehende MarkovShield Demonstration-Webapplikation verwendet.

[94] (Wikipedia.org, 2017)

III.11.1.3. Tools

Für die Load Tests wurde das Load & Performance Testing Tool Gatling eingesetzt. Gatling ist in Scala geschrieben und erlaubt es auf eine einfache Art Performance Tests zu schreiben oder gar mittels eines mitgelieferten Recorders generieren zu lassen. Dieser Recorder ist im Prinzip nichts anderes als ein Proxy, welcher im Browser eingerichtet werden muss. Um anschliessend einen Load Test selbständig aufzustellen, muss die zu testende Webseite so besucht werden, wie sie später auch getestet werden sollte. Der Recorder zeichnet im Hintergrund alle Requests und zusätzlichen Informationen wie Request Headers auf und generiert daraus beim beenden des Aufzeichnen den entsprechenden Scala Code. Dieser generierte Load Test Code kann anschliessend nach Belieben abgeändert und erweitert werden.

Für die MarkovShield Load Tests wurde Gatling Version 2.2.5 eingesetzt.

III.11.1.4. Testablauf

Im Rahmen dieses Load Testes wurde ein möglichst realistisches Testscenario erstellt, welches berücksichtigt, dass Benutzer Webseiten mit viel relevanten Informationen tendenziell länger besuchen. Zudem wurde der Test so gestaltet, dass die angegebene Benutzeranzahl jeweils über 60 Sekunden hinweg linear gesteigert wurde.

Der Load Test selbst wurde jeweils von einem Entwickler Notebook aus gestartet. Dies bedeutet, dass die einzelnen Requests über WLAN gestartet und über LAN beim Testsystem ankamen und verarbeitet wurden. Parallel zu den Load Tests wurden jeweils Dauer-Pings laufen gelassen um eruieren zu können, wie gross die Round-Trip-Time (RTT) bis zum Server selbst ist. Aus diesen Resultaten konnte ausfindig gemacht werden, dass die RTT zwischen Entwickler Notebook und Server im Durchschnitt stets zwischen 2ms bis 4ms war.

Der Vollständigkeit halber wird das Testscenario hier aufzeigt. Dabei ist wichtig zu verstehen, dass die Werte in den «.*pause(X)*»-Aufrufen in Sekunden angegeben werden.

```
val scn: ScenarioBuilder = scenario("NormalUsageSimulation1")
  .exec(http("Req 0 /")
    .get("/")
    .headers(headers_0))
  .pause(3)
  .exec(http("Req 1 /configuration/pre-auth/")
    .get("/configuration/pre-auth/")
    .headers(headers_0))
  .pause(14)
  .exec(http("Req 2 /howto/explorer/")
    .get("/howto/explorer/")
    .headers(headers_0))
  .pause(10)
  .exec(http("Req 3 /howto/start-stop/")
    .get("/howto/start-stop/")
    .headers(headers_0))
  .pause(6)
  .exec(http("Req 4 /configuration/backend-apache/")
    .get("/configuration/backend-apache/")
    .headers(headers_0))
  .pause(9)
  .exec(http("Req 5 CRIT /private/request-header/")
    .get("/private/request-header/")
    .headers(headers_0))
  .pause(3)
  .exec(http("Req 6 /login/login.php")
    .post("/login/login.php")
    .headers(headers_0)
    .formParam("user", "hacker")
    .formParam("password", "compass")
    .formParam("appid", "0"))
  .pause(8)
  .exec(http("Req 7 CRIT /private/1/")
    .get("/private/1/")
    .headers(headers_0)
    .resources(http("Req 8 CRIT /private/1/printheader.php")
    .get("/private/1/printheader.php")
    .headers(headers_0)))
  .pause(17)
  .exec(http("Req 9 CRIT /private/2/")
    .get("/private/2/")
    .headers(headers_0)
    .resources(http("Req 10 CRIT /private/2/chat.php")
    .get("/private/2/chat.php")
    .headers(headers_0)))
  .pause(10)
```

Abbildung 38 Load Test Ablauf Teil 1

```
46      .pause(10)
47      .exec(http("Req 11 CRIT /private/2/chat.php")
48        .post("/private/2/chat.php")
49        .headers(headers_0)
50        .formParam("message", "Test"))
51      .pause(13)
52      .exec(http("Req 12 /howto/edit/")
53        .get("/howto/edit/")
54        .headers(headers_0))
55      .pause(10)
56      .exec(http("Req 13 /configuration/pre-auth/")
57        .get("/configuration/pre-auth/")
58        .headers(headers_0))
59      .pause(9)
60      .exec(http("Req 14 CRIT /private/request-header/")
61        .get("/private/request-header/")
62        .headers(headers_0))
63      .pause(11)
64      .exec(http("Req 15 /howto/start-stop/")
65        .get("/howto/start-stop/")
66        .headers(headers_0))
67      .pause(17)
68      .exec(http("Req 16 /configuration/backend-apache/")
69        .get("/configuration/backend-apache/")
70        .headers(headers_0))
71      .pause(2)
72      .exec(http("Req 17 /configuration/client-certificates/")
73        .get("/configuration/client-certificates/")
74        .headers(headers_0))
75      .pause(6)
76      .exec(http("Req 18 /admin")
77        .get("/admin")
78        .headers(headers_0)
79        .check(status.is(404)))
80      .pause(18)
81      .exec(http("Req 19 /configuration/frontend-apache/")
82        .get("/configuration/frontend-apache/")
83        .headers(headers_0))
84      .pause(10)
85      .exec(http("Req 20 /configuration/pre-auth/")
86        .get("/configuration/pre-auth/")
87        .headers(headers_0))
88
89    setUp(scn.inject(rampUsers(100) over (60 seconds))).protocols(httpProtocol)
```

Abbildung 39 Load Test Ablauf Teil 2

III.11.2. Performance Tests mit Gatling

Für die nachfolgenden Load Test wurden jeweils 100 gleichzeitige Benutzer simuliert.

III.11.2.1. Enforce Mode

Die nachfolgende Abbildung 40 zeigt mittels der blauen Fläche «*Numbers of requests*» an, wie viele Requests pro Sekunde betrachtet über den Load Test Zeitverlauf abgesetzt wurden. Anhand der orangen Kurve «*Active Users*» kann erkannt werden, dass die 100 simulierten Benutzer über eine Zeit von 60 Sekunden linear hochskaliert und gegen Ende auch wieder linear heruntergefahren wurden. Die Benutzer beenden sich automatisch mehr oder weniger wieder linear, da alle den exakt gleichen Ablauf mit den gleichen Wartezeiten zwischen den Requests absolvieren.

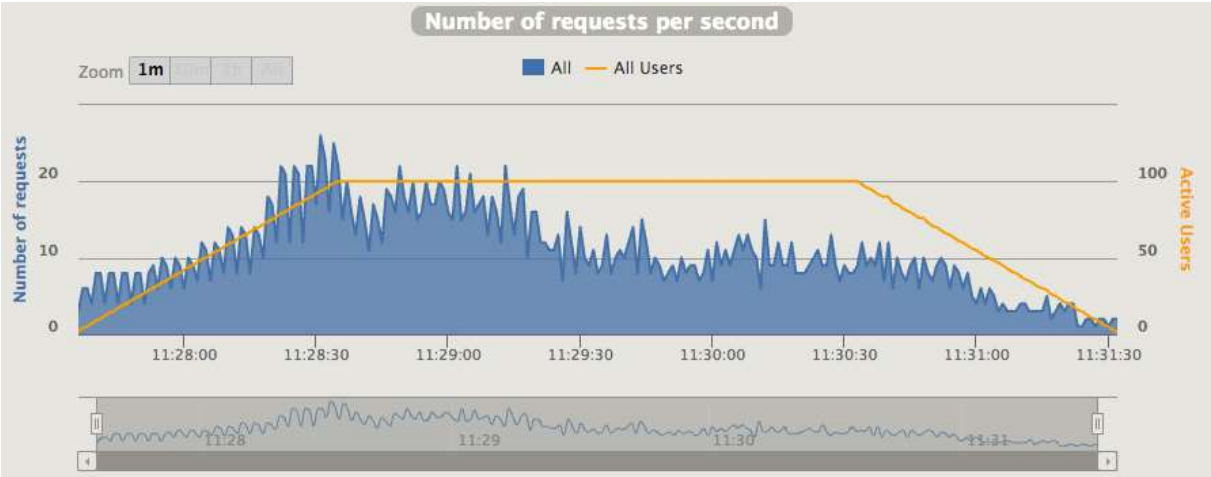


Abbildung 40 Enforce Mode: Anzahl Requests (blau) und Anzahl User (orange) pro Sekunde

In der Abbildung 41 ist ersichtlich, wie sich die Response Times unterteilt auf die farblich unterschiedenen Quantile verhalten. Die Zeit auf der X-Achse kann verwendet werden, um eine Korrelation zwischen der Anzahl der Requests und der Response Time herzustellen.

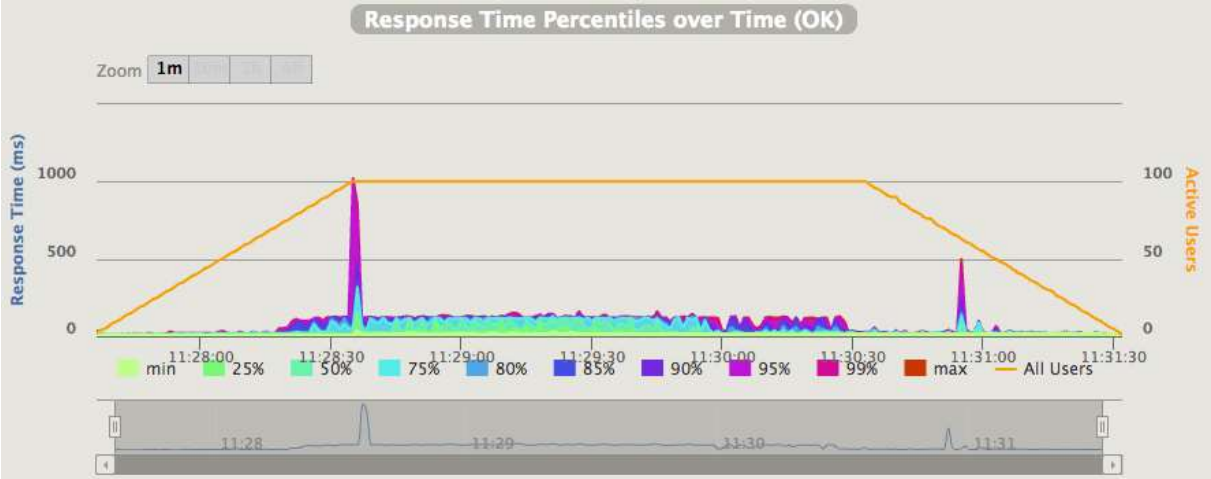


Abbildung 41 Enforce Mode: Response Time (in ms) pro Quantil

Zum Zeitpunkt um etwa 11:28:35 Uhr herum ist ein Peak zusehen, welcher nur etwa die obersten 1-5% der Requests betrifft (siehe Abbildung 41).

In der Abbildung 42 ist eine genauere Übersicht über alle Requests aufgeführt. Es ist gut zu sehen, dass lediglich einzelne kritische Requests eine hohe Antwortzeit besitzen. Kritische Requests sind anhand des Präfixes «*Req X C*» oder «*Req X CR*» erkennbar.

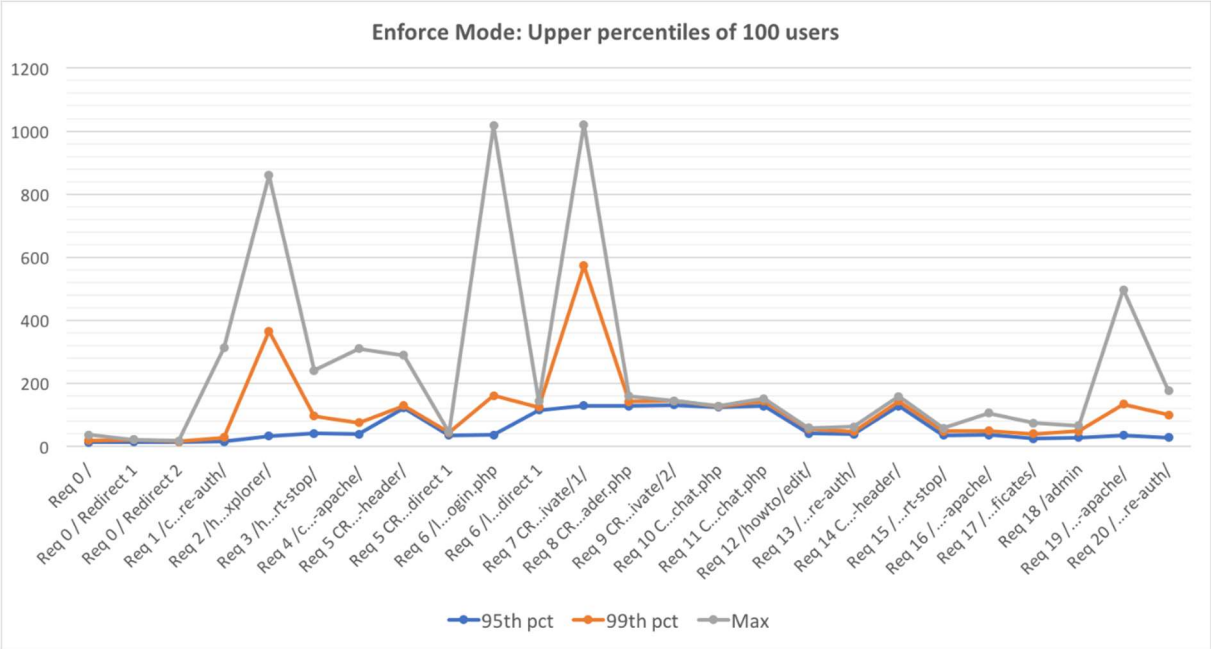


Abbildung 42 Enforce Mode: 95th, 99th Quantile und die maximale Response Time

III.11.2.2. Learning Mode

Im Learning Mode sieht der Aufbau der Requests pro Sekunde sehr ähnlich wie der entsprechende Aufbau im Enforce Mode aus.

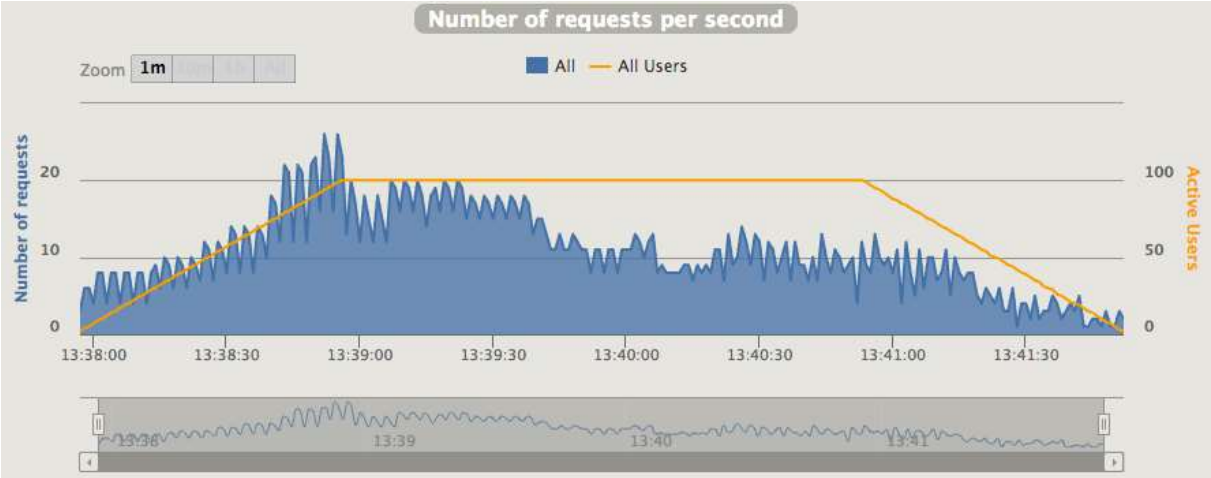


Abbildung 43 Learning Mode: Anzahl Requests (blau) und Anzahl User (orange) pro Sekunde

Bei den Response Times kann jedoch erkannt werden, dass sich diese mehrheitlich um circa 30ms herumbewegen und nur wenige kleinere Ausschläge bis auf 150ms aufweisen.

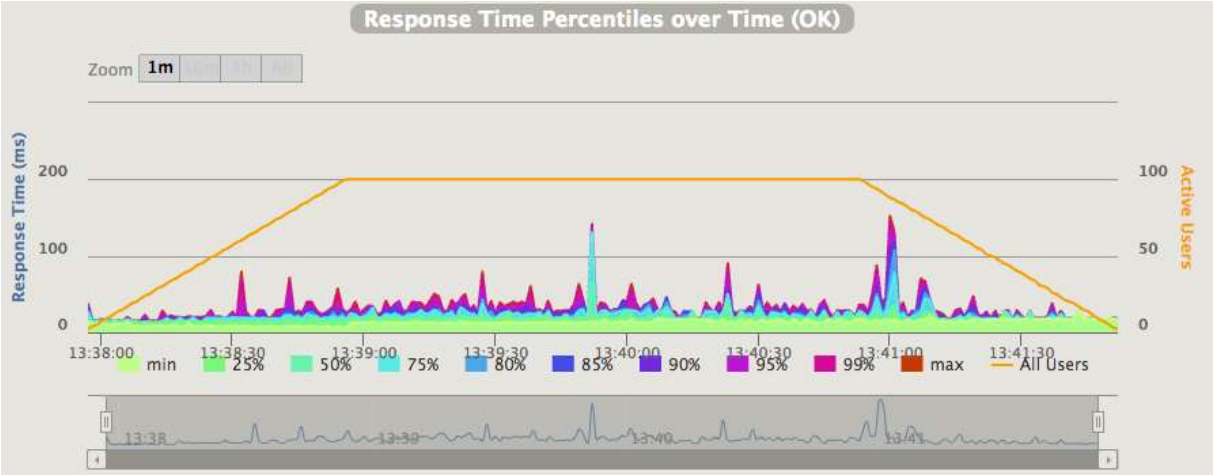


Abbildung 44 Learning Mode: Response Time (in ms) pro Quantil

III.11.2.3. MOD_BUT

Der Aufbau der Requests pro Sekunde ist auch bei MOD_BUT in etwa der gleiche. Dies ist logisch, da Gatling keine Informationen über das verwendete Apache HTTPD Modul besitzt und sich im Idealfall immer gleich verhält.

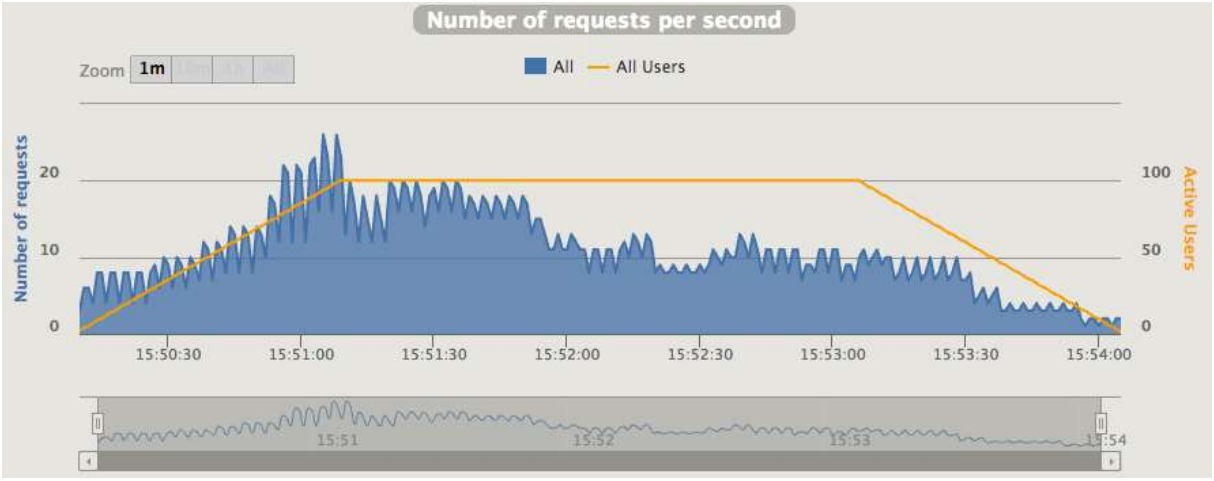


Abbildung 45 MOD_BUT: Anzahl Requests (blau) und Anzahl User (orange) pro Sekunde

In der Abbildung 46 kann erkannt werden, dass die Response Time für die Mehrheit der Requests bei rund 20ms liegt. Auch hier, wie schon beim Learning Mode, sind mehrere kleine Peaks bis zu 100ms aufgetreten.

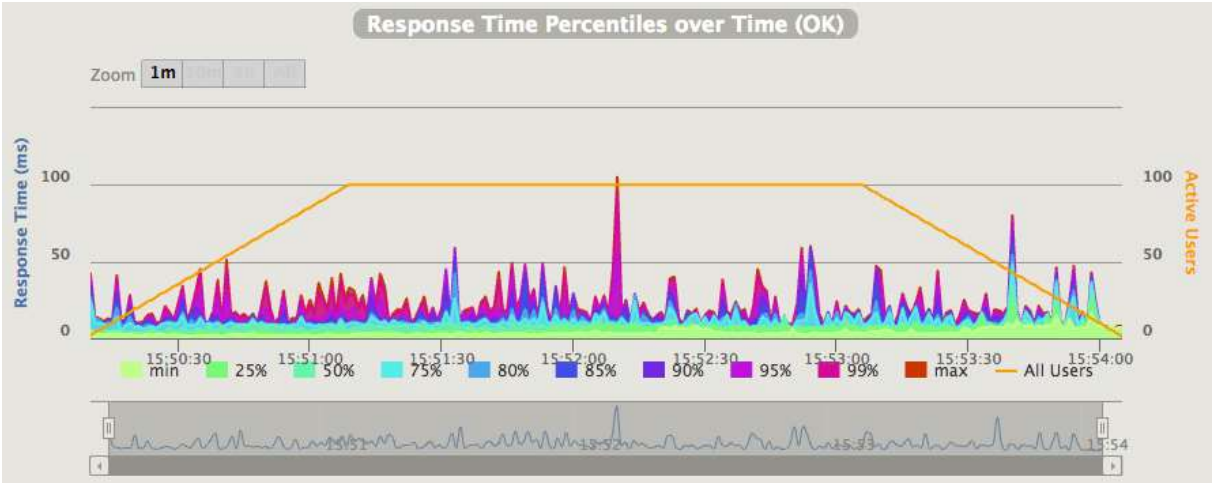


Abbildung 46 MOD_BUT: Response Time (in ms) pro Quantil

III.11.3. Performance Test Schlussfolgerung

Anhand der Abbildung 47 wird ersichtlich, wie sich die einzelnen mod_mshield Modi im Vergleich zu MOD_BUT schlagen. Dabei kann erkannt werden, dass mod_mshield im Learning Mode rund doppelt so lange für die Request-Bearbeitung braucht als MOD_BUT. Der Enforce Mode von mod_mshield braucht wiederum rund nochmals doppelt so lange wie der Learning Mode, um die Requests im Durchschnitt abzuarbeiten.

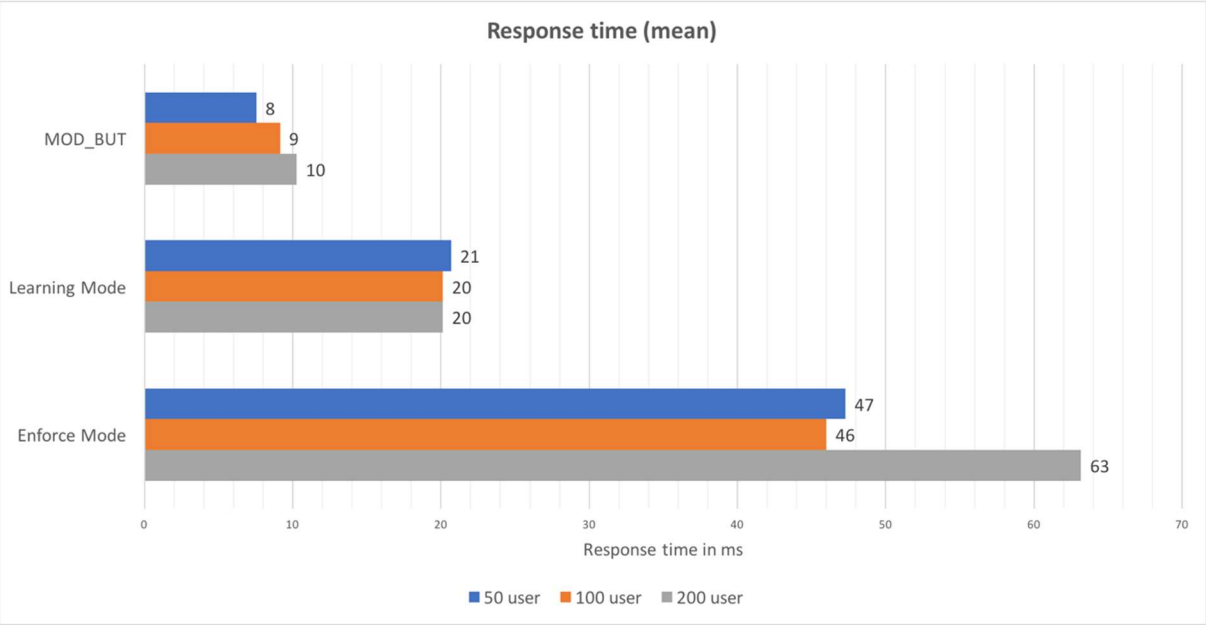


Abbildung 47 Durchschnittliche Response Time pro Mode/Modul

Bei MOD_BUT sowie dem Learning Mode von mod_mshield können zwischen den verschiedenen Simulationsszenarien keine wirklichen Unterschiede erkannt werden. Einzig beim Enforce Mode wird ersichtlich, dass bei 200 gleichzeitigen Benutzer die Response Time gegenüber 100 Benutzern um ca. einen Drittel ansteigt.

In der Abbildung 48 wird aufgezeigt, dass die 95th Quantile in einem akzeptierbaren Rahmen liegen. Auch hier ist erkennbar, dass MOD_BUT weiterhin rund nur die Hälfte der Zeit vom Learning Modus des mod_mshield Apache Modules braucht. Des Weiteren wird veranschaulicht, dass der Enforce Mode rund viermal so lange braucht wir der Learning Mode.

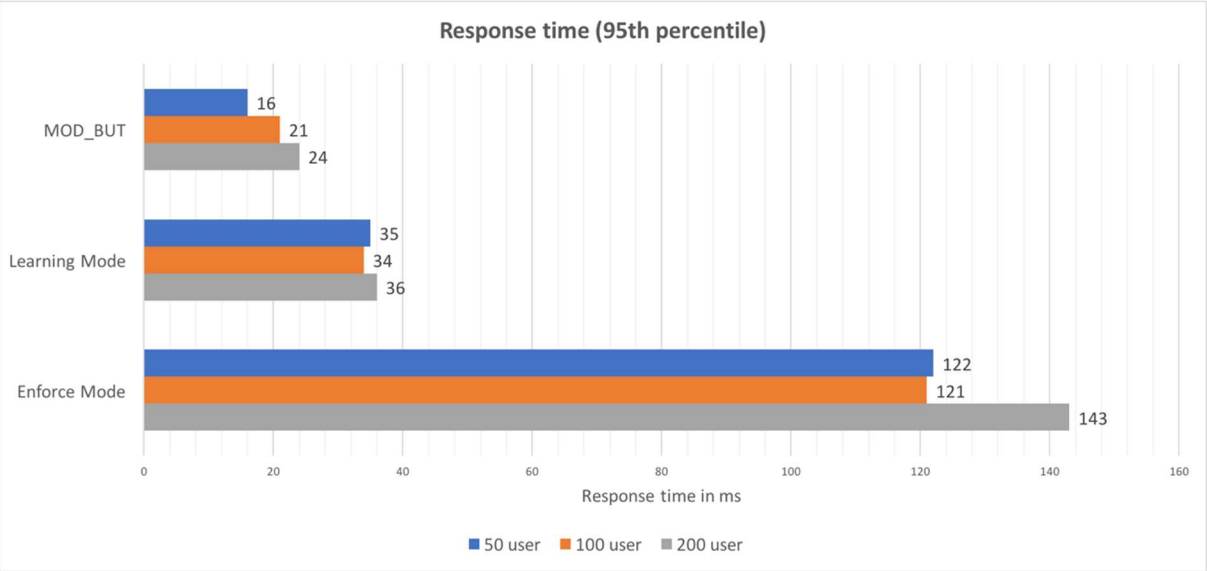


Abbildung 48 95th Quantil Response Time pro Mode/Modul

Als kurzgefasstes Fazit kann gesagt werden, dass MarkovShield im Learning Mode sowohl als auch im Enforce Mode für bis zu 200 gleichzeitige Benutzer gute Antwortzeiten liefern kann unter welchen die User Experience nicht merklich beeinflusst werden sollte.

III.12. Weiterentwicklung

III.12.1. Möglichkeiten der Weiterentwicklung

Im Kapitel II.4.2 Ausblick auf die Weiterentwicklung wurde bereits mit einer groben Übersicht auf die möglichen Erweiterungen von MarkovShield eingegangen. Im folgenden Kapitel werden diese bereits erwähnten und denkbaren Erweiterungen detaillierter beschrieben.

III.12.1.1. User-based Rating

In der aktuellen Implementierung ist es so, dass die Rating Resultate und entsprechenden Schutzmassnahmen nur auf eine aktive Session angewandt werden können.

Nach dem Start einer neuen Session, wird diese bisher unabhängig des im Voraus erreichten Session-Status behandelt. Zum Beispiel wird nach einer Detektion eines FRAUD die momentan vorhandene Session des Benutzers terminiert, es steht dem Benutzer allerdings frei eine neue Session zu beginnen, welche unabhängig des vorher erreichten Resultats bewertet wird.

Die hier denkbare Erweiterung von MarkovShield besteht darin, das Rating Resultat User-based zu realisieren. Dies bedeutet zum Beispiel, dass nach der Erkennung einer Session als FRAUD diese Erkenntnis dem Benutzer hinterlegt wird. Versucht sich nun der Benutzer mit einer neuen Session einzuloggen, könnte dies auf dem Login Server unterbunden werden, da dieser vom erkannten FRAUD des Benutzers Kenntnis hat.

Um ein User-based Rating zu realisieren, müssten mindestens die folgenden Anpassungen gemacht werden:

- Der Login Server müsste über das Rating Resultat informiert werden. Denkbare Ansätze wären hier, dass der Login Server selbst über ein eigenes Kafka Topic informiert würde. Dazu bräuchte der Login Server mindestens Angaben zum Benutzer und zum Rating Resultat. Alternativ könnte der Login Server ein API zur Verfügung stellen, über welches die Engine dem Login Server die gleichen Informationen mitteilt.
- Auf dem Login Server oder in der optional angebundenen Benutzerverwaltung (z.B. ein LDAP) müsste der Status für den Benutzer hinterlegt werden. Würde ein LDAP am Login Server angeschlossen sein, so könnte der Benutzer relativ einfach LDAP-mässig gesperrt werden.
- Die MarkovShield Engine müsste so erweitert werden, dass sie die Benutzerkennung mit dem jeweiligen Rating Resultat in ein eigenes Kafka Topic schreibt oder aber einen API POST Call auf den Login Server mit den gleichen Daten sendet.
- Alternativ ist es denkbar, dass der Login Server den Status des Benutzers beim Login aktiv auf einer andere Komponte, zum Beispiel Apache Kafka via Interactive Querie, abfragt.

III.12.1.2. Dashboard zur Visualisierung der Models und Analyseresultate

Die Analyseresultate und die User Models, welche in der aktuellen Implementierung über ein REST-API zur Verfügung gestellt werden, könnten durch ein entsprechendes Dashboard visuell ansprechend dargestellt werden. Des Weiteren könnte über das Dashboard den negativen Resultaten wie SUSPICIOUS und FRAUD eine spezielle Beachtung geschenkt werden.

Die benötigten Anpassungen für diesen Schritt wären minimal, da lediglich noch eine weitere kleine Komponente dazu käme, welche auf eine MarkovShield bestehende Funktionalität, die REST-API, zugreifen würde. Die effektiv benötigten Schritte wären:

- In einem ersten Schritt, müsste eine kleiner Webserver ausgewählt und entsprechend konfiguriert werden, sodass dieser in der Lage ist, eine Single-page Application (SPA)[95] zu hosten.
- Das Dashboard könnte am einfachsten als Single-page Application realisiert werden, welche direkt mittels API Calls mit der MarkovShield REST-API kommuniziert.

Grundsätzlich spielt es keine grosse Rolle, was für Technologien hier gewählt werden. Ein Beispiel eines solchen Dashboard-Stacks könnte aus NodeJS[96] als Webserver bestehen, welcher eine Angular[97] Single Page Application Seite hostet. Die Schwierigkeit besteht vor allem darin die erhaltenen Daten für einen Benutzer verständlich darzustellen.

III.12.1.3. Web UI zur Konfiguration

Eine erweiterte zusätzliche Webkomponente könnte, neben dem vorhin erwähnten Dashboard, ein Web UI sein, über welches MarkovShield konfiguriert werden könnte. Hier wäre insbesondere interessant, den künftigen MarkovShield Administratoren eine Möglichkeit zu bieten, über welche sie komfortabel die Risk Levels der URLs konfigurieren könnten. Neben der einfachen Zuweisung zwischen URL und Risk Level, könnte das Konfigurations-UI auch noch die folgenden Features anbieten:

- Neue URLs könnten automatisch von mod_mshield erkannt und im Konfigurations-UI entsprechend dargestellt werden. Dies bedingt, dass auf diese neue URL zugegriffen wurde, da mod_mshield ansonsten nicht weiss, dass diese URL auf der Webapplikation überhaupt existiert ist. Diese neuen URLs können auf dem UI dann so angezeigt werden, sodass ein MarkovShield Administrator gleich ein Risk Level dazu definieren kann.

[95] (Wikipedia.org, 2017)
[96] (Node.js Foundation, 2017)
[97] (angular.io, 2017)

- Die Konfigurations-UI Komponente könnte die MarkovShield Administratoren darauf hinweisen, dass mehere URL-Pattern sich überschneiden und somit ein Konflikt besteht. Ein solcher Konflikt müssten dann manuell aufgelöst werden.

Das Konfigurations-UI könnte ohne grössere Bemühungen auch direkt im Dashboard integriert werden, welches Analyseresultate und User Models darstellt. Dabei müsste jedoch beachtet werden, dass gegebenenfalls noch eine Dashboard-interne Benutzerverwaltung mit Rollenverteilung implementiert werden müsste. Dies aus dem Grund, da gewisse Benutzer vielleicht nur in der Lage sein sollten, die Analyseresultate anzuschauen aber zugleich nicht berechtigt sein sollten, ein Risk Level einer URL zu verändern.

Um die Konfigurations-UI im vorhin beschriebenen Umfang realisieren zu können, sind die nachfolgenden Änderungen an den bestehenden MarkovShield Komponenten nötig:

- Die URL Risk Level Konfiguration muss an einem Ort gespeichert werden. Da bereits Kafka zur Persistierung der Sessiondaten im Einsatz ist, könnte ein weiteres Topic erstellt werden, in welchem die URLs mit dem jeweils dazugehörigen Risk Level Werten persistiert werden.
- Im mod_mshield Apache Modul müsste die aktuelle URL Risk Level Konfiguration über die Apache Konfigurationsdirektiven durch die Konfigurationsmöglichkeit über ein Kafka Topic ersetzt werden. Mod_mshield muss dazu in der Lage sein bestehende URL Konfigurationen aus diesem Topic auszulesen. Gleichzeitig müssen Änderungen in diesem Topic regelmässig überprüft und in die Konfiguration von mod_mshield nachgeladen werden.

III.12.1.4. Erweiterung des Login Servers

Der aktuell im Einsatz stehende Login Server ist im Grunde nichts anderes als ein einfaches PHP-Skript. Dieses prüft die eingegebenen Credentials, zeigt gegebenfalls ein statischen Two-Way-Authentication Token an und fügt anschliessend dem Request ein Cookie hinzu. Dieses Cookie nutzt mod_mshield anschliessend um zu bestimmen, ob der Login erfolgreich war oder nicht (Wert im Cookie: «*LOGON=ok*»).

Zu Beginn der Login Server Weiterentwicklung sollte die Technologie des Login Servers evaluiert werden. In der aktuellen MarkovShield Implementation ist es lediglich ein PHP-Skript. Zur Auswahl steht dieses PHP-Skript zu erweitern oder aber eine komplett andere Technologie wie NodeJS zu setzen. Die Grundanforderungen sind die, dass mit der gewählten Technologie mindestens die folgend aufgelisteten Funktionalitäten abgedeckt werden können:

- Ein Request muss entgegengenommen werden und HTTP POST Parameter müssen ausgelesen werden können.
- Die eingegebenen Credentials müssen überprüft werden können.
- Unter Umständen muss eine Benutzerverwaltung wie ein LDAP angesprochen werden können.

- Schlussendlich muss ein Cookie mit dem Login Status dem Request angehängt werden.
- Optional muss das Cookie signiert werden können.

Um in einem weiteren Schritt die volle Login Server Funktionalität mit realem Two-Way-Authentication zu erreichen, ist es notwendig, einen SMS- und/oder SMTP-Gateway einzubinden. Die Token Versandmöglichkeit kann alternativ auch über einen Online Service realisiert werden. Mittels dieses Schrittes ist sichergestellt, dass ein Token verwendet wird, sollte ein Step-Up initialisiert werden. Des Weiteren ist es denkbar, dass der Login Server an ein LDAP oder ein vergleichbares Benutzerverwaltungssystem angebunden wird. Ist im initialen Login Server Weiterentwicklungsschritt entschieden worden auf NodeJS zu setzen, kann die LDAP Funktionalität ohne grössere Aufwände realisiert werden, da schon diverse NPM Module öffentlich verfügbar sind, welche die Kommunikation mit LDAP übernehmen.

Zu guter Letzt sollte noch an die Sicherheit beim Login Prozess gedacht werden. Da aktuell mod_mshield über ein am Request angehängtes Cookie den Login Status des Benutzers austauschen, wäre es ratsam das Cookie auf Seite Login Server mit einem Timestamp zu ergänzen und anschliessend zu signieren. Mod_mshield wiederum müsste dann die Login Server Cookie Signatur auf die Richtigkeit und den Timestamp auf die Aktualität überprüfen. Nur wenn beides erfolgreich war, dürfte der Benutzer vom mod_mshield her als eingeloggt erachtet werden.

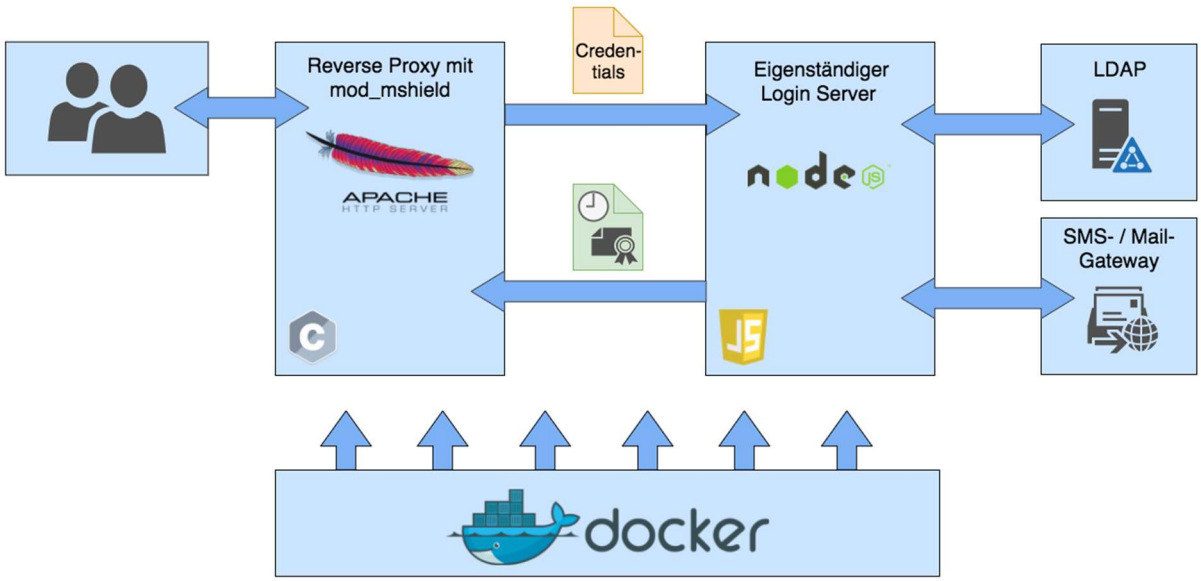


Abbildung 49 Login Server Erweiterung

III.12.1.5. Erweiterung um weitere Models

Die MarkovShield Engine könnte in Zukunft um weitere Machine-Learning Models erweitert werden. Die Schwierigkeit bestünde in einer solchen Erweiterung hauptsächlich darin, dass passende mathematische Models gefunden und richtig implementiert werden. Die Integration in die Engine selbst wäre eine einfachere Sache, da in der Engine der einfachen Erweiterbarkeit grosse Beachtung geschenkt wurde.[98]

III.12.1.6. Anbindung an eine bestehende Risk Engine

An vielen potentiellen Einsatzorten von MarkovShield wird es bereits eine bestehende Risk Engine Lösung geben. Sollte dies so sein, wäre es absolut denkbar, dass MarkovShield so erweitert wird, sodass mit einer solchen bestehenden Risk Engine kommuniziert werden kann. Dabei würden die Risk Analyseresultate der bestehende Risk Engine weitergegeben und nicht mehr über den Key-Value Store zum mod_mshield Apache Modul kommuniziert werden.

Abhängig von der bestehenden Risk Engine wären unteranderem die folgenden Schritte notwendig:

- Die MarkovShield Engine informiert die Risk Engine über den jeweils aktuellen Stand der Session. Zum Beispiel über eine von der bestehenden Risk Engine bereitgestellte API.
- Mod_mshield wird so angepasst, dass keine Requests mehr blockiert werden. Stattdessen werden die Requests der Webapplikation weitergegeben und diese wiederum übernimmt das Handling aufgrund der Risk Engine Auswertung.
- Der gesamte Key-Value Store Teil innerhalb von mod_mshield kann entfernt werden, da dieser nicht mehr benötigt wird.

[98] Siehe Kapitel III.8.1.5 Models generelle Struktur

III.12.1.7. Absicherung der Kommunikation

In der aktuellen Konfiguration der MarkovShield Gesamtlösung ist die gesamte Kommunikation zwischen den Komponenten nicht explizit verschlüsselt oder anderweitig geschützt. Dies wurde auf Grund der begrenzten Zeit absichtlich weggelassen.

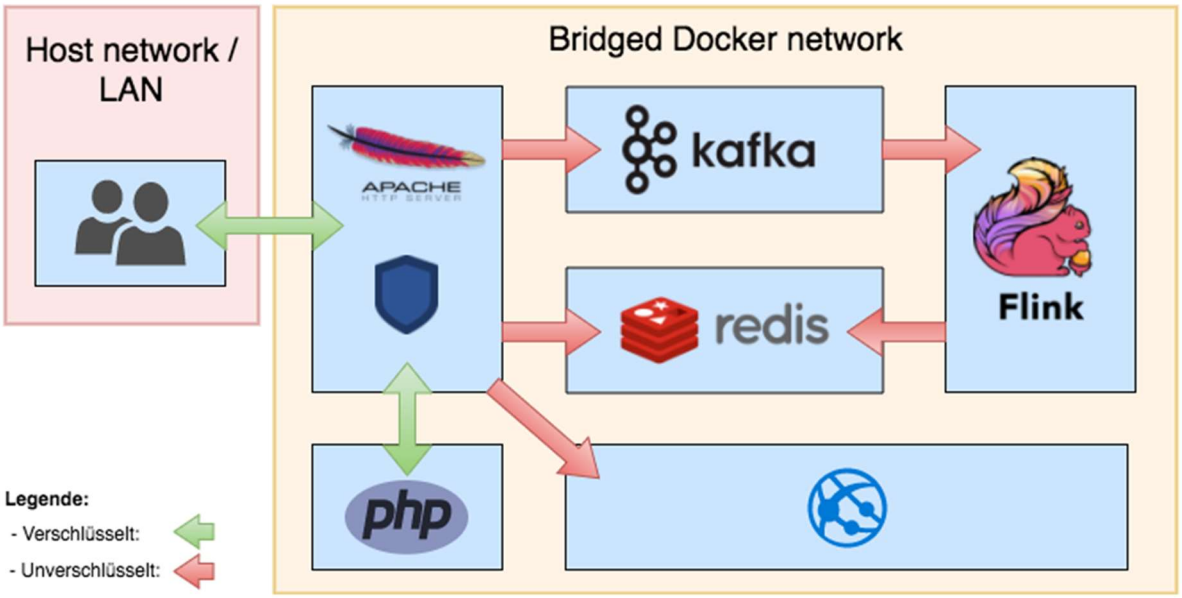


Abbildung 50 Ist-Stand Verschlüsselung der Kommunikation und Netzwerksegmentierung

Für den produktiven Einsatz ist es jedoch unabdingbar, die Kommunikation zwischen den einzelnen Komponenten abzusichern. Grundsätzlich ist es so, dass jede der verwendeten Komponenten eine verschlüsselte Kommunikation von Haus aus unterstützt.[99][100][101][102]

Neben der normalen Verschlüsselung der Kommunikation sollte auch darauf geachtet werden, dass MarkovShield Control Daten nicht über das normale Daten Netz verschickt werden. So sollte zum Beispiel darauf geachtet werden, dass die Request Informationen via Kafka an die Engine und von der Engine via Redis zurück zu mod_mshield ausschliesslich über ein separates und zugleich speziell abgesichertes Netzwerksegment übertragen werden. Mittels dieser Netzwerksegmentierung kann eine zusätzliche Hürde für einen allfälligen Angreifer realisiert werden.

[99] (Juma, 2016)
[100] (librdkafka, 14)
[101] (redis, kein Datum)
[102] (Flink, kein Datum)

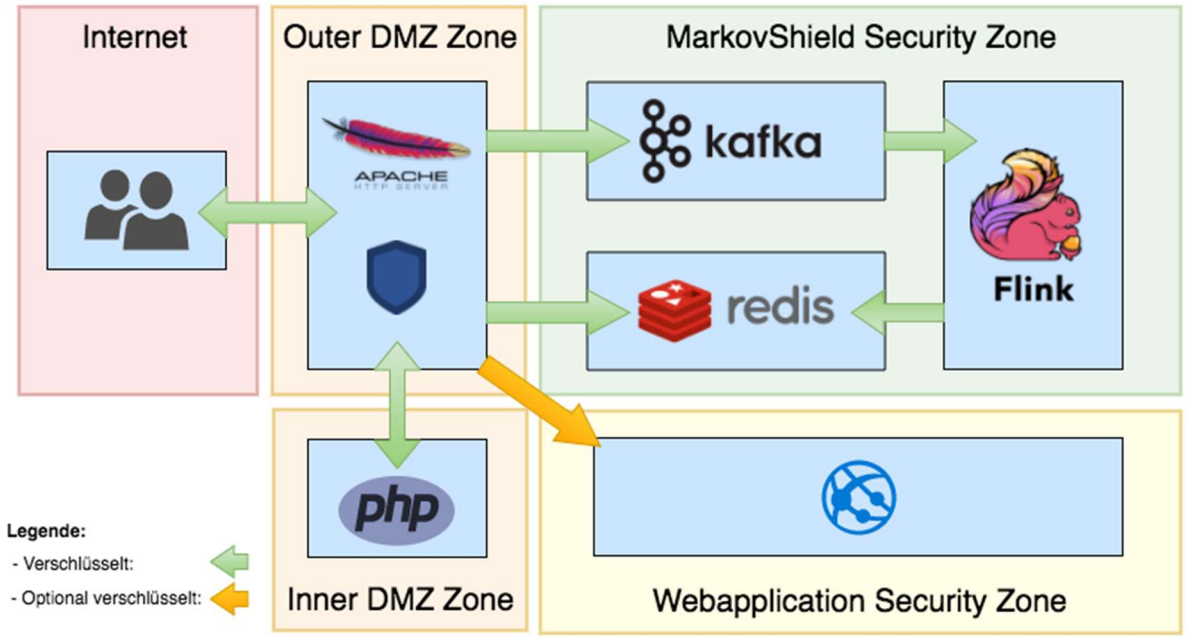


Abbildung 51 Empfohlene Verschlüsselung und Netzwerksegmentierung für produktiven Einsatz

III.12.1.8. Information einer Kontaktperson

Sollte ein FRAUD erkannt werden, wird in der aktuellen Implementation die momentan aktive Session terminiert und der Benutzer weitergeleitet. In einem weiteren Ausbauschritt wäre es denkbar, dass nicht nur die Session terminiert wird, sondern dass zugleich eine oder mehrere Kontaktpersonen benachrichtigt werden. Diese Kontaktperson wiederum kann daraufhin entsprechende Schritte unternehmen. Zum Beispiel kann sie den Benutzer kontaktieren und ihn über den möglichen Maleware-Befall seines Rechners aufklären. Selbstverständlich muss hierfür erst ein seriöser Prozess erarbeitet werden, da solche Anrufe von angeblichen Maleware-Befällen in der vergangenen Zeit von Betrügern ausgenutzt wurden.

Des Weiteren existiert hierbei vor allem grosses Potential, um eine Service Center API anzubinden oder ein Mail zu versenden, um eine zeitnahe Kontaktaufnahme zu ermöglichen. Über eine Service Center API könnte gleich ein Incident mit entsprechenden Informationen zum Benutzer und zum Vorfall angelegt werden. Es wäre auch denkbar, diese solche FRAUD Information lediglich oder zudem in einem Dashboard anzeigen zu lassen, um den MarkovShield Administratoren den aktuellen Stand aufzuzeigen.

III.12.2. mod_mshield Refactorings

Historisch bedingt sind im mod_mshield Apache Modul diverse Refactorings als sinnvoll zu erachten.

So ist zum Beispiel die Datenstruktur hinter dem Cookie Store eine selbst implementierte Linked List. Der Cookie Store wird dazu genutzt alle Cookies der Sessions zwischen zu speichern.

```
1   /**
2    * @brief The mod_mshield cookie data
3    */
4   typedef struct {
5       int slot_used;       /**< Bool if the cookie is used of not */
6       char name[100];      /**< Name of the cookie  */
7       char value[100];     /**< Cookie value */
8       int next;            /**< "Pointer" to the next cookie */
9       int prev;            /**< "Pointer" to the previous cookie*/
10      int location_id;     /**< ID from the mod_mshield directory level configuration location */
11  } cookie_t;
```

Abbildung 52 mod_mshield Cookie_t Datenstruktur

In diesem konkreten Beispiel könnte auf eine von der Apache Portable Runtime Project Library (APR)[103] zur Verfügung gestellte Datenstruktur, wie zum Beispiel eine «apr_table_t», zurückgegriffen werden.

Ein weiterer Punkt wäre zum Beispiel, die Funktion mit der mod_mshield Kernfunktionalität («*static int mshield_access_checker(request_rec *r)*») besser zu strukturieren. Aktuell umfasst diese Funktion alleine knapp 500 Zeilen und ist dadurch relativ unübersichtlich.

[103] (The Apache Software Foundation, 2017)

Neben den bereits erwähnten möglichen Refactorings, wäre es aus Sicht der Performance und der besseren parallelen Nutzung der mod_mshield Funktionalitäten durchaus sinnvoll, die aktuell bestehende Mutex Gebrauchsweise zu überarbeiten. Wie bereits im Kapitel III.8.1.2 Verbindungsaufbau beschrieben, existiert zurzeit noch ein suboptimales Locken des globalen Mutexes, da dieser von einem Worker mehr oder weniger während dem ganzen Abarbeiten eines Requests behalten wird. Die einzige Ausnahme ist, dass während dem Warten auf das Engine Resultat der Mutex freigegeben wird. In einem weiteren mod_mshield Weiterentwicklungsschritt wäre es deshalb sinnvoll zu überprüfen, ob der globale Mutex nicht jeweils bloss vor den Zugriffen auf das Shared-Memory gelockt werden sollte. Während der anderen Zeit wäre es somit problemlos möglich, dass parallellaufende Apache Worker die Requests abarbeiten könnten sofern sie keine Zugriffe auf das Shared-Memory benötigen.

Allgemein wäre es angebracht, als Erstes ein grobes Code Review mit einem C-Experten durchzuführen, um die grössten Fehler zu bestimmen. Anschliessend könnten die weiteren Massnahmen und Reviews schrittweise Top-Down bestimmt werden.

III.12.3. Vorgehen

III.12.3.1. Prioritäten

Bei der Umsetzung der unter dem Kapitel III.12.1 Möglichkeiten der Weiterentwicklung beschriebenen Punkten empfiehlt es sich die folgenden Prioritäten als grobe Richtlinie für die Weiterentwicklungsreihenfolge zu verwenden. Je nach Einsatzort und bestehende Umgebung kann und soll die folgend gegebene Prioritätenliste variieren.

Nr.	Punkt	Begründung
1	Absicherung der Kommunikation	Für den produktiven Einsatz ist es absolut empfohlen die Kommunikation zwischen den verschiedenen MarkovShield Komponenten entsprechend zu schützen.
2	Erweiterung des Login Servers	Der Login Server sollte als eigenständige Komponente implementiert werden, sodass dieser optimal von den anderen Komponenten getrennt und somit speziell geschützt werden kann.
3	Dashboard zur Visualisierung	Aktuell bietet MarkovShield nur sehr begrenzt eine Übersicht über den aktuellen Status der Sessions und Models. Aus diesem Grund empfiehlt es sich in naher Zukunft ein entsprechendes Dashboard zu entwickeln.
4	mod_mshield Refactorings	Die bereits erwähnten Refactorings am mod_mshield Apache Modul sind zwar nicht kritisch, jedoch der vereinfachten künftigen Codewartbarkeit halber doch von Relevanz. Zugleich könnte durch die Mutex-Umstellung ein Performancegewinn resultieren.

Tabelle 20 Weiterentwicklungsprioritäten

Die restlichen, hier in der Tabelle nicht aufgelisteten, möglichen Weiterentwicklungen, sind mehr optional zusehen und je nach eigenen Bedürfnissen spezifisch mit einer eigens definierten Priorität anzupacken.

III.12.3.2. Veröffentlichung von MarkovShield

Innerhalb des MarkovShield Entwicklungsteams wurde beschlossen, dass die Software gegen Ende der Arbeit als Open Source der Öffentlichkeit zur Verfügung gestellt werden soll. Zum einen wurde dieser Schritt beschlossen, da Open Source Code innerhalb MarkovShield selbst verwendet wird aber auch weil dadurch weitere Personen an der Entwicklung teilhaben können. Zudem wird so der Weg geebnet, um MarkovShield anderen Personen demonstrieren zu und ihnen die Gesamtlösung schmackhaft machen zu können. Ein weiterer Grund ist, dass geteiltes Wissen oftmals ein Gewinn für alle ist.

Der Source Code an sich wurde über eine neu gegründete Organisation auf Github.com veröffentlicht. Die einzelnen Repositories können unter folgendem Link gefunden werden:

https://github.com/MarkovShield

Die für die Demo Applikation benötigten Docker Images wurden auf dem Docker Hub der Öffentlichkeit bereitgestellt. Wie auch auf Github.com, wurde eine neue Organisation gegründet um gegen aussen ein einheitliches Bild präsentieren zu können:

https://hub.docker.com/u/MarkovShield/

Da nach der Arbeit an MarkovShield weiterentwickelt wird, wurde in den Git Repositories und bei den Docker Hub Images jeweils der Tag 2.0 für die Version verwendet, welche zum Zeitpunkt der Arbeitsabgabe vorhanden war und auch abgegeben wurde. Sollte nach einer gewissen Zeit das Bedürfnis vorhanden sein die neusten Weiterentwicklungen einzusehen oder zu testen, sollte ein neuerer Tag betrachtet werden.

III.13. Projektmanagement

III.13.1. Projektmeetings

Zu Beginn der Arbeit wurde vom Team festgelegt, dass grundsätzlich jede Woche ein Projektstatusmeeting mit dem BA Team und dem Betreuer gehalten werden soll. Sollte es gegen Mitte oder Ende der Arbeit Wochen geben in welchen es nahezu nichts zu besprechen gibt, kann ein Meeting nach gegenseitiger Absprache auch ausgelassen werden.

Es wurde zudem definiert, dass jeweils rund 24 Stunden vor einem Meeting eine Aufstellung über die zu besprechenden Themen dem Betreuer zugestellt wird, sodass genügend Zeit für die Vorbereitung des Meetings zur Verfügung steht. Im Anschluss an ein Meeting sollte innert 48 Stunden ein Meeting Protokoll erstellt werden, welches die diskutierten Themen und Entscheide wiederspiegelt.

III.13.2. Vorgehensmodell

Für diese Arbeit wurde primär nach dem Rational Unified Process (RUP)[104] Vorgehensmodell gearbeitet. Innerhalb der einzelnen Projektphasen wurde mittels agilen zwei wöchigen Sprints die Arbeit fortlaufend eingeteilt. So konnte stets gewährleistet werden, dass auf Veränderungen und neue Gegebenheiten optimal reagiert werden konnte.

[104] (Wikipedia.org, 2017)

III.13.3. Meilensteine

Innerhalb des Projektes wurden 5 Meilensteine geplant und erreicht.

Nr	Meilenstein	Datum
MS1	End of Inception	24.03.17
MS2	End of Elaboration	28.04.17
MS3	Feature Freeze	26.05.17
MS4	End of Construction	09.06.17
MS5	Completion of the project	14.06.17

Tabelle 21 Meilensteine des Projektes

Da die Inception Phase aufgrund der hohen Komplexität um eine Woche verlängert werden musste, mussten die nachfolgenden Meilensteine entsprechend auch verschoben werden. Diese Anpassungen sind in der folgenden Milestone-Trend-Analysis (MTA) zu entnehmen.

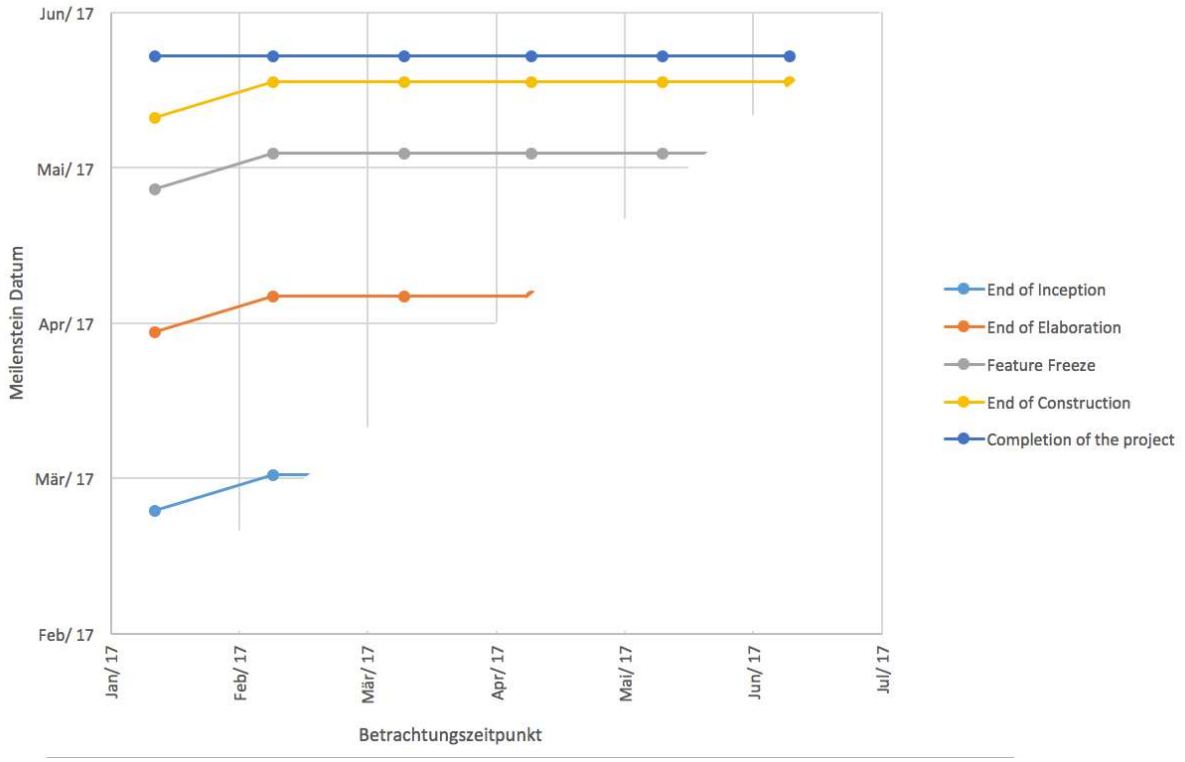


Diagramm 19 Milestone Trend Analyse

III.13.4. Stakeholder

Für die Arbeit wurde eine Stakeholder Analyse durchgeführt, um besser abschätzen zu können, wer wie viel Einfluss und Interesse am Projekt haben könnte.

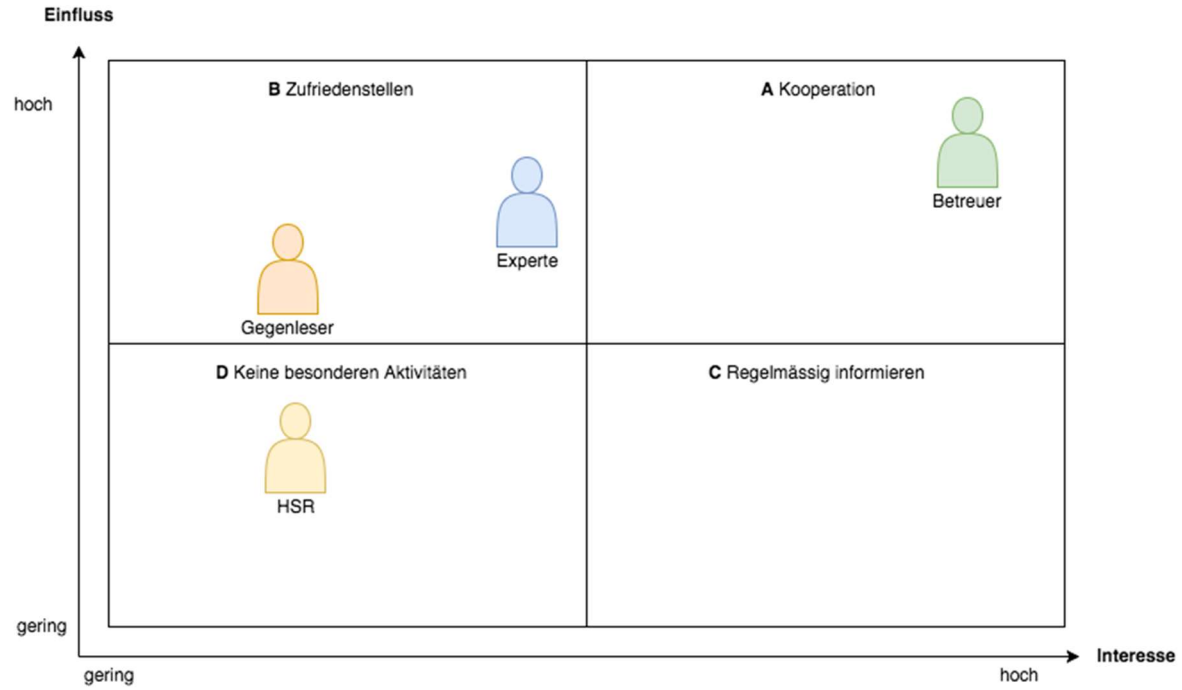


Abbildung 53 Stakeholder Analyse

Stakeholder	Beschreibung
Betreuer	Der Betreuer ist die wichtigste Ansprechperson und hat den grössten Einfluss auf die Arbeit. Es muss regelmässige Kommunikation geben und normale bis wichtige Entscheidungen müssen in Absprache mit ihm erfolgen.
Experte	Der Experte hat einen relativ hohen Einfluss auf die Bewertung des Endproduktes und sollte auch schon während der Arbeit mit Informationen entsprechend versorgt werden.
Gegenleser	Der Gegenleser hat auf die Schlussbewertung der Arbeit einen mittleren Einfluss und muss zu gegebener Zeit im richtigen Mass über den Projektstand informiert werden.
HSR	Die HSR bietet die Rahmenbedingungen für die Arbeit und ist am Erfolg des Projektes interessiert.

Tabelle 22 Beschreibung der Stakeholder

III.13.5. Team

Name	Rolle	Zuständigkeiten
Ivan Bütler	Bachelorarbeit-Betreuer	Verantwortlich für die Bachelorarbeit und die Betreuung der Entwickler
Matthias Gabriel	Entwickler	Zuständig für die Kommunikations-Middleware (Kafka Streams) und die Engine (Apache Flink)
Philip Schmid	Entwickler	Zuständig für das Apache Modul (mod_mshield) und das Deployment (Docker)

Tabelle 23 Auflistung der Projektbeteiligten und deren Verantwortung

Aktivität	Datum	Verantwortung
Inception		
Project initialization	24.02.17	Philip Schmid
Risk analysis	28.02.17	Matthias Gabriel
Requirements rough	08.03.17	Philip Schmid
Setup toolchain	10.03.17	Team
Evaluate communication flows	24.03.17	Team
Evaluate architecture	24.03.17	Team
MS 1 End of Inception	**24.03.17**	**Team**
Elaboration		
Knowhow building	07.04.17	Team
Domain modeling	07.04.17	Matthias Gabriel
Finalize requirements	14.07.17	Philip Schmid
Deployment concept	21.04.17	Philip Schmid
Architecture prototype	28.04.17	Matthias Gabriel
MS 2 End of Elaboration	**28.04.17**	**Team**
Construction		
Alpha	12.05.17	Matthias Gabriel
Beta	02.06.17	Matthias Gabriel
MS 3 Feature Freeze	**26.05.17**	**Team**
System tests	02.06.17	Philip Schmid
Load tests	09.06.17	Philip Schmid
Bug fixing	09.06.17	Team
MS 4 End of Construction	**09.06.17**	**Team**
Transition		
Product release	12.06.17	Team
Create poster	13.06.17	Philip Schmid
Finalize documentation	14.06.17	Matthias Gabriel
MS 5 Completion of the project	**14.06.17**	**Team**

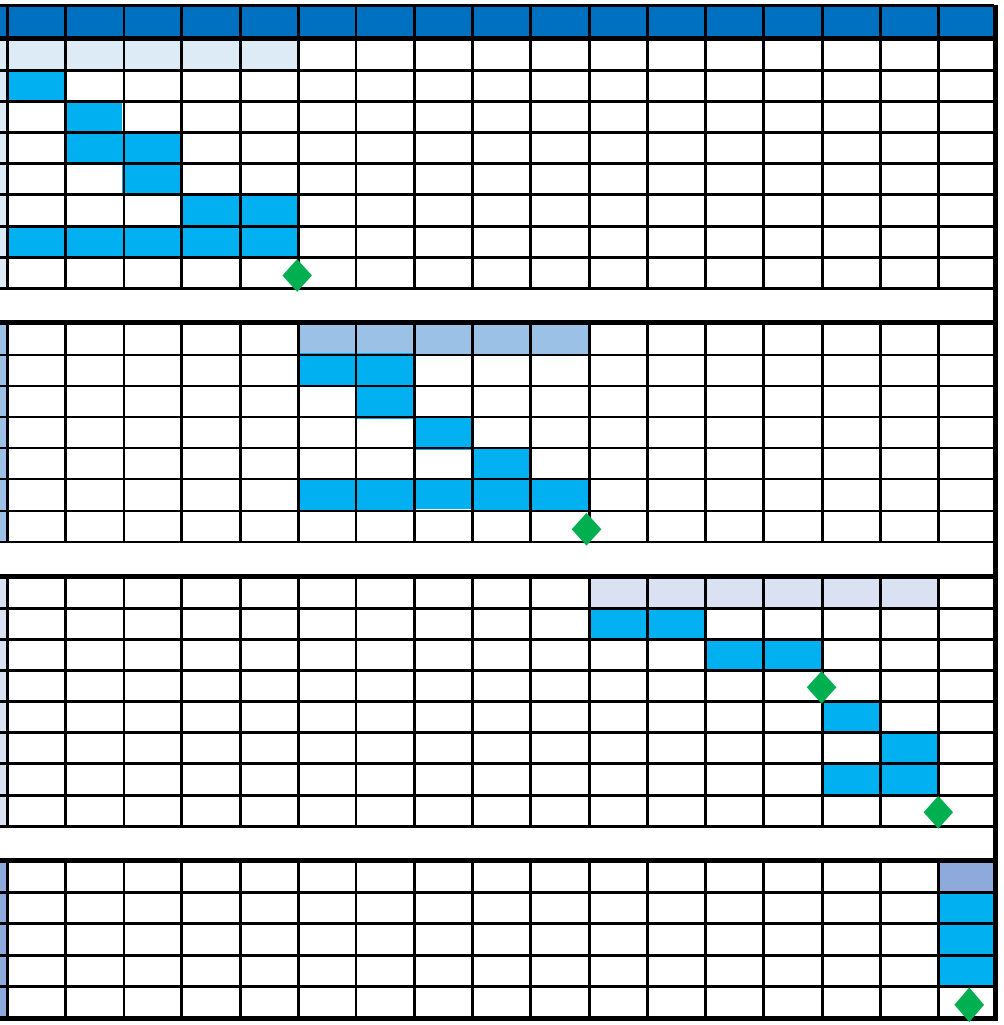


Tabelle 24 Projektplanung MarkovShield

Nr	Titel	Beschreibung	max. Schaden [h]	Eintrittswahr-scheinlichkeit	Gewichteter Schaden	Vorbeugung	Verhalten beim Eintreten	Risio eliminiert	Risiko eliminiert durch / Status
R1	JIRA nicht verfügbar	Der JIRA online Service steht wegen eines Problems nicht zur Verfügung.	4	5%	0.2	Keine Möglichkeit	Normal weiterarbeiten, alle Projektmanagement-Dinge manuell in einem Excel dokumentieren	OK	NICHT EINGETRETEN
R2	Software Versionen	Gewisse Software-Versionen werden sich während der Projektlaufzeit verändern.	0	99%	0	Die verwendete Software wird zu Projektbeginn mit der Version festgelegt und während der Projektdauer nicht angepasst.	-	OK	Software Versionen wurden festgelegt.
R3	Software Architektur Fehler	Die Software Architektur könnte Fehler enthalten und muss korrigiert werden.	16	25%	4	Bis End. of Elab. muss ein Prototyp bestehen, welcher alle wichtigen Bereiche der Software umfasst, um Designfehler frühstmöglich erkennen zu können.	Allfällige Designfehler müssen im Team besprochen und eine Gegenmassnahme bestimmt werden. Gegebenfalls wird externe Hilfe hinzugezogen.	OK	Das Kafka Consume stellte auf mod_mshield Seite ein grösseres Problem dar, da mittels diesem nicht spezifisch auf eine Nachricht gehört werden konnte. Als Lösung wurde auf Redis für den Empfang des Engine Resultates gewechselt.
R4	Git Repository	Git Repository wird aufgrund eines Fehlers korrupt.	15	10%	1.5	Einmal im Tag sollte das gesamte Git Repository durch einen automatischen Job gesichert werden.	Backup zurückspielen	OK	NICHT EINGETRETEN
R5	Bitbucket nicht verfügbar	Bitbucket ist nicht erreichbar, dadurch kann nicht auf das Repository zugegriffen werden	8	1%	0.04	Keine Möglichkeit	In der Zwischenzeit sollte mit lokalen Commits gearbeitet werden bis der Service wieder verfügbar ist. Sollte auch nach längerer Zeit BitBucket noch nicht wieder online sein, wird bei den lokalen Git Repositories auf eine neue Remote gewechselt (z.B. Github).	OK	NICHT EINGETRETEN
R6	Performance der Architektur	Die Performance der gewählten Architektur ist zu niedrig. Dies hat einen Einfluss auf die Requestbearbeitungszeit und somit auf das Benutzerfeeling.	24	10%	2.4	Während der Elaborationsphase müssen die Architekturkomponenten gut evaluiert werden. Die angegebenen theoretischen Leitstungsmerkmale der Komponenten müssen beachtet und aufgefundene Performanceerfahrungsbereichte mit einbezogen werden. Mitteles eines Architekturprotoypen und realistischen Tests muss die Performance der Architektur getestet werden.	Abhängig vom Projektfortschritt muss im BA Team entschieden werden, ob eine andere Komponente bessere Performanceeigenschaften bieten könnte. Falls dem so wäre, müsste diese eingebaut werden.	OK	Obwohl gegen Ende der Arbeit zuerst Performance Probleme vorhanden waren, konnten diese noch rechtzeitig eruiert und behoben werden. Die MarkovShield Performance ist nun im grünen Bereich.

Tabelle 25 Technische Risiken Teil 1

Nr	Titel	Beschreibung	max. Schaden [h]	Eintrittswahr-scheinlichkeit	Gewichteter Schaden	Vorbeugung	Verhalten beim Eintreten	Risio eliminiert	Risiko eliminiert durch / Status
R7	Performance der Auswertung	Die Performance der True Streaming Processing Engine lässt zu wünschen übrig. D.h. die Auswertung der aktuellen Clickstreams kann nicht genügend schnell abgehandelt werden.	24	5%	1.2	Es muss in der Elaborationsphase sehr sorgfältig evaluiert werden, welche Streaming Processing Lösung theoretisch die Performance Anforderungen abdecken kann. Zugleich muss dies mittels eines Architekturprototypen belegt bestätigt werden.	Je nach Fortschritt des Projektes muss evaluiert werden, ob noch auf eine alternative Streaming Processing Lösung gewechselt werden kann.	OK	Das MarkovShield Backend (Engine + Kommunikations-Middleware) konnte soweit optimiert werden, dass die verbrauchte Zeit im vernünftigen Rahmen liegt (unter 250ms, im Durchschnitt bei rund 120ms).
R8	Fehlende Reverse Proxy Funktionalität	Die zurzeit existierende Funktionalität des Apache Modul MOD_BUT kann die zusätzlichen Anforderungen seitens MarkovShield nicht abdecken.	8	70%	5.6	In der Inceptionphase sollte soweit als möglich evaluiert werden, welche Funktionalitäten seitens Reverse Proxy gebraucht werden.	Sofern die fehlende Funktionalität das Apache Modul MOD_BUT betrifft, soll mit Ivan Bütler (MOD_BUT Maintainer) zusammen geschaut werden, ob sie nachträglich noch im Modul eingebaut werden kann oder ob eine Eigenentwicklung die Funktionalität zur Verfügung stellen könnte.	OK	Relativ früh im Projekt wurde entschieden, ein eigenes Apache Modul auf MOD_BUT basierend zu entwickeln, um die MarkovShield spezifischen Funktiontlitäten dort zu integrieren.
R9	Zu wenig Testdaten	Es sind für das Model Training zu wenig Testdaten verfügbar.	6	25%	1.5	Bereits zu Beginn der BA sollten die Logdaten-Quellen (z.B. das Hacking-Lab) so konfiguriert werden, dass ab diesem Zeitpunkt alle Logs abgespeichert werden.	Es müssen weitere Testdaten besorgt werden. Einerseits kann dies durch Abwarten geschehen oder proaktiv durch das generieren von entsprechenden Testdaten.	OK	Die Testdaten wurden aufgrund des Projektverlaufes nicht gebraucht.
R10	Dokument korrupt	Ein Dokument auf OneDrive wird korrupt und kann nicht mehr geöffnet werden.	4	10%	0.4	Der gesammte OneDrive Ordner wird einmal pro Woche (Abend vor dem Meeting) manuell auf das Dokumenten Backup Repository kopiert und somit gesichert.	Dokument, sofern als möglich, aus der OneDrive History restoren oder im schlimmsten Fall aus dem Dokumenten Backup Repo holen.	OK	NICHT EINGETRETEN

Tabelle 26 Technische Risiken Teil 2

III.14. Projektmonitoring

III.14.1. Projektauswertung

In den nachstehenden Unterkapiteln wird auf diverse Auswertungen zum Projekt eingegangen.

III.14.1.1. Arbeitszeit

Das folgende Diagramm zeigt die aufgewendete Arbeitszeit pro Teammitglied und Projektphase.

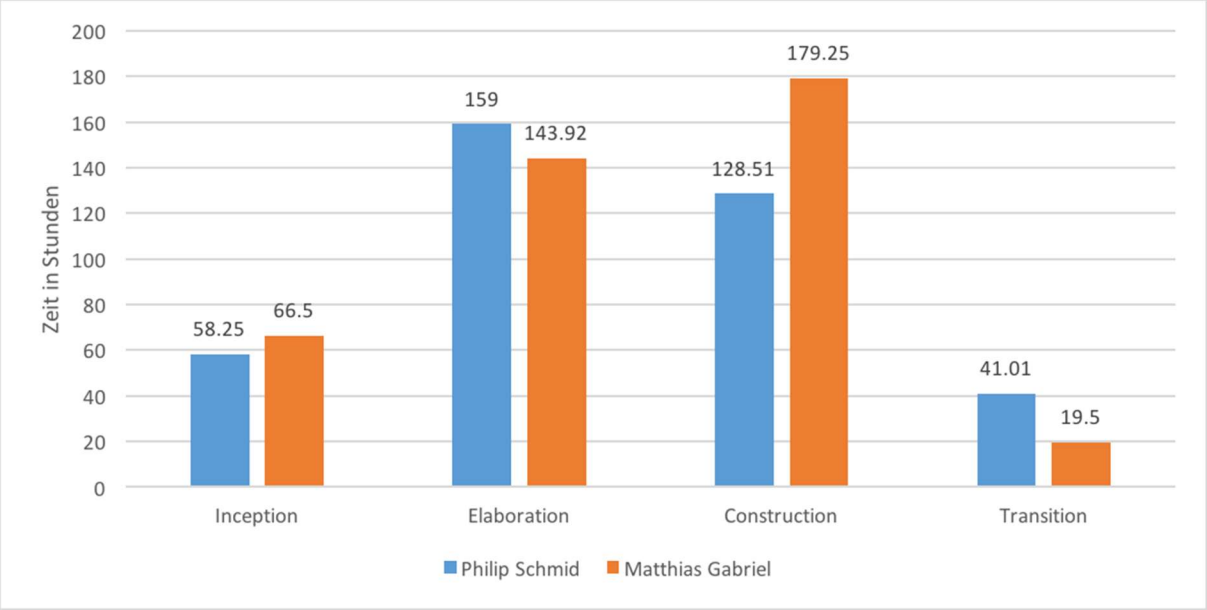


Diagramm 20 Ist-Zeit pro Teammitglied und Projektphase

Die Soll-Arbeitszeit-Vorgabe für diese Arbeit lag bei 360 Stunden, verteilt über 17 Wochen. Nachstehend die Aufschlüsselung der effektiv aufgewandten Zeit pro Teammitglied und Epic. Für diese Arbeit wurden insgesamt 18 Epics definiert und abgearbeitet.

Epic	Matthias	Philip	Total
Project initialization		3.83	**3.83**
Risk analysis		0.5	**0.5**
Setup toolchain		1.5	**1.5**
Evaluate architecture	37	43	**80**
Architecture prototype	156.75	125	**281.75**
Alpha	22.01	55.17	**77.17**
Beta	54.25	60.75	**115**
System tests		14	**14**
Create presentation		3	**3**
Create poster	1.5	0.5	**2**
Product release	1	6	**7**
Finalize documentation	84.26	48.58	**132.85**
Project Meeting	30	47.33	**77.33**
Total:	**386.77**	**409.16**	**795.93**

Tabelle 27 Aufgewandte Zeit in Stunden pro Teammitglied und Epic

III.14.1.2. Projektphase

Nachfolgend wird im Diagramm 21 Issues pro Projektphase aufgezeigt, wie viele Issues pro Projektphase gelöst wurden.

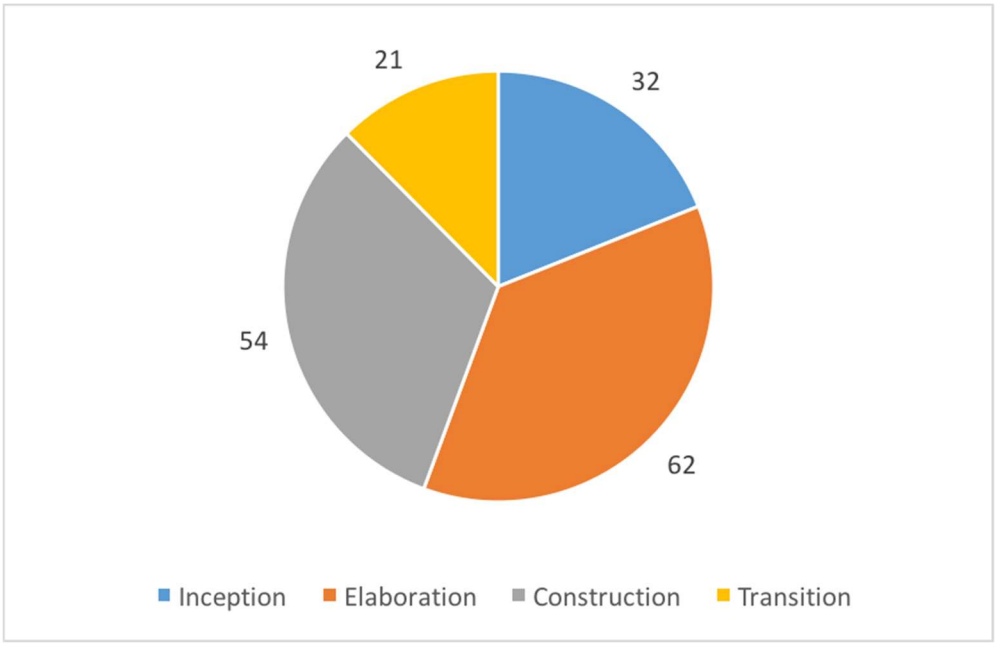


Diagramm 21 Issues pro Projektphase

Im Diagramm 22 Investierte Zeit in Stunden pro Projektphase wird aufgezeigt, wie viele Stunden Aufwand pro Projektphase investiert wurden.

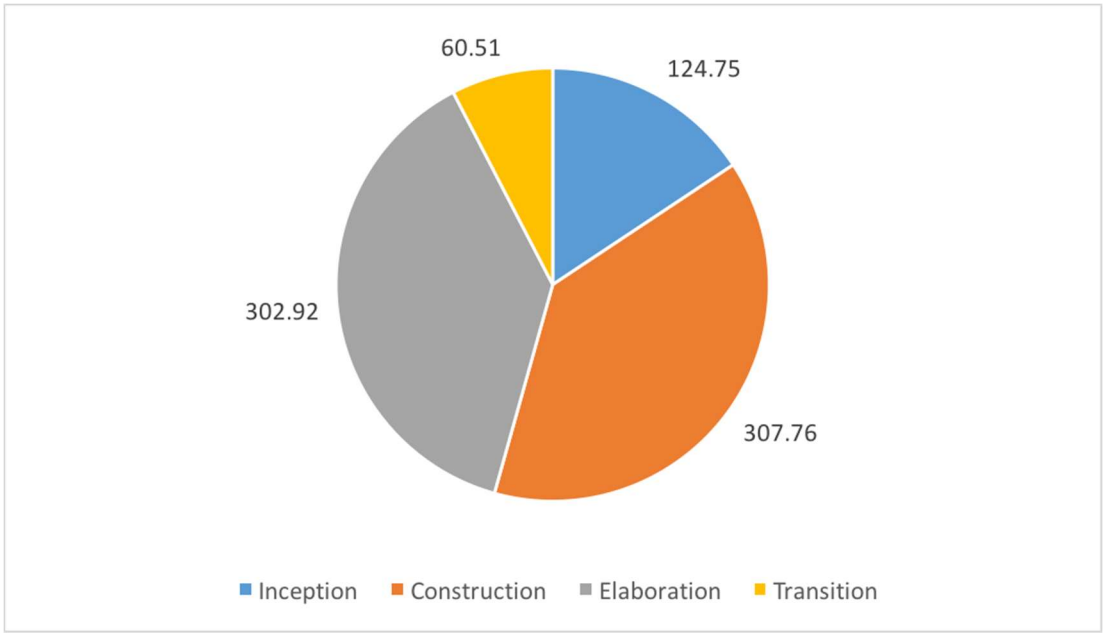


Diagramm 22 Investierte Zeit in Stunden pro Projektphase

III.14.1.3. Issuetyp

Nach nachfolgenden Diagramm wird aufgeschlüsselt, wie viele Issues von welchem Issuetyp gelöst wurden. Über die gesamte Arbeit hinweg wurden 169 Issues gelöst.

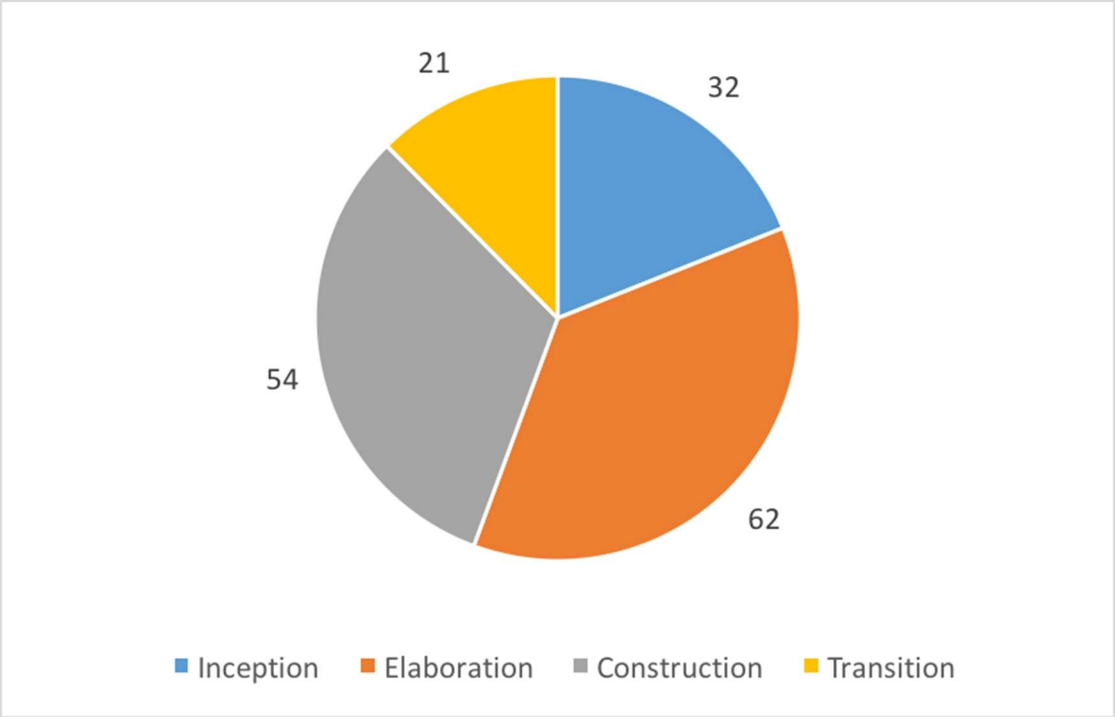


Diagramm 23 Anzahl Issues gruppiert nach Issuetyp

III.14.2. Codestatistik

III.14.2.1. Lines of Code (LOC)

In der anschliessenden Tabelle ist aufgelistet, wie viele Lines Of Code pro verwendete Programmiersprache oder verwendeter Definitions-Syntax programmiert oder geschrieben wurden.

Software / Komponente	Sprache / Syntax	LOC
Kommunikations-Middleware (Kafka Streams)	Java	2320
Engine (Core)	Java	2020
Engine (Flink)	Java	570
Deployment (inkl. Compose)	Dockerfile	460
mod_mshield	C	600
mod_mshield	Markdown	630
mod_mshield	Make	70
	Total:	**6670**

Tabelle 28 LOC gruppiert nach Software

Die aufgezeigten Zahlen sind jeweils auf 10-er Werte gerundet. Allgemein muss beachtet werden, dass diese Zahlen bloss grobe Richtwerte darstellen und nicht als exakte LOC-Zahlen betrachtet werden dürfen.

IV. VERZEICHNISSE

IV.1. Abbildungsverzeichnis

IV.2. Diagrammverzeichnis

IV.3. Tabellenverzeichnis

IV.4. Abkürzungen und wichtige Begriffe

Begriff	**Beschreibung**
Anomalie	Als Anomalie wird eine Abweichung vom Normalfall oder eine Unregelmässigkeit bezeichnet.
Apache Flink	Apache Flink ist ein Open Source Stream Processing Framework, welches unter dem Dach der Apache Foundation steht. Genauere Informationen sind im Kapitel III.4.1.3 Apache Flink zu finden.
Apache HTTPD	Das Apache HTTPD Projekt stellt einen mittels Modulen erweiterbaren HTTP-Server zur Verfügung.
Apache Kafka	Apache Kafka bezeichnet sich selbst als distributed streaming platform und wird in MarkovShield sowohl für die Kommunikation zwischen den Komponenten, als auch für die Persistierung verwendet. Wie Apache Flink ist auch Apache Kafka der Apache Foundation angeschlossen.
Click	Als Click wird ein einzelner effektiver Klick eines Benutzers auf der Webseite bezeichnet. Ein solcher Klick löst im Normalfall ein Request aus, sollte im Browser nichts gecached sein. Jeder Click wird über eine eindeutige ClickUUID identifiziert.
Clickstream	Eine Folge von Klicks, welche der Benutzer auf einer Webseite gemacht hat. Die Reihenfolge der Klicks spielt eine wichtige Rolle.
Docker	Docker ist eine Container Technologie welches erlaubt, Applikationen komplett voneinander getrennt laufen zu lassen.
Docker Compose	Docker Compose ist ein Docker Helper Tool das es erlaubt mehrere Docker Container als komplettes Setup zusammen in einem einzelnen File zu definieren und anschliessend auch so laufen zu lassen.
Erwartungswert	Der Erwartungswert beschreibt das arithmetische Mittel der Zahlenmenge. Er wird auch als Mittelwert bezeichnet, wobei diese Bezeichnung nicht eindeutig ist.
FRAUD	Betrug/Missbrauch von schützenswerten Daten

Frequency Model	Im Zusammenhang mit MarkovShield ein Model welches erfasst, wie oft auf welchen Link auf einer Webseite geklickt wurde.
Gatling	Ein Performance Testing Tool für Webseiten. Siehe auch Kapitel III.11.1.3 Tools.
Inter-Quartile-Range	Der Inter-Quartile-Range oder auch IQR ist ein Mass, welches der Differenz zwischen dem 25 und 75 Quantil entspricht.
Login Server	Server der die Pre-Authentifizierung der Benutzer übernimmt indem. Der Request wird an den Login Server weitergeleitet, sollte von mod_mshield/MOD_BUT her erkannt werden, dass die Session für die aufgerufene URL noch nicht genügend authentifiziert ist.
Login Server	Server der die Pre-Authentifizierung der Benutzer übernimmt indem. Der Request wird an den Login Server weitergeleitet, sollte von mod_mshield/MOD_BUT her erkannt werden, dass die Session für die aufgerufene URL noch nicht genügend authentifiziert ist.
Man-in-the-Browser	Ein Angriff auf den Browser des Benutzers. Eine detaillierte Erklärung kann im Kapitel II.2.1 Man-in-the-Browser nachgelesen werden
Man-in-the-Middle	Ein Angriff, bei welchem die Kommunikation zwischen zwei Systemen abgefangen und manipuliert wird.
MOD_BUT	Apache Modul, welches erlaubt, Sessions auf dem Reverse Proxy zu verwalten.
Mutual Exclusion (Mutex)	Objekt das von Threads gelockt und anschliessend wieder freigegeben werden kann. Mittels Mutex können kritische Bereiche abgesichert werden, dass sich nicht mehrere Threads gleichzeitig in diesem Bereich befinden können.
Outlier	Als Outlier werden oft Datenpunkte bezeichnet, welche sich sehr klar von allen anderen Messungen abhebt. Wie ein Outlier definiert ist und wie mit Ihnen umgegangen werden soll hängt stark vom spezifischen Umfeld ab.
OWA	Outlook Web Access: Eine Webseite, welche von Microsoft Exchange zur Verfügung gestellt wird, um auf die Mails zuzugreifen.

Propability Density Function	Mithilfe der Probability Density Function oder PDF können Aussagen über die Wahrscheinlichkeit des Auftritts einer Variable getroffen werden.
R	R ist eine Programmiersprache, welche oft für statistische Aufgaben genutzt wird. Es gibt eine umfassende Auswahl an Libraries für allerlei statistische Aufgaben.
Redis	Redis ist ein In-Memory Data Store, welcher auch als Datenbank, Cache und als Message Broker genutzt werden kann. Im Kapitel III.8.1.1 Kommunikation des Ergebnisses ist die Nutzung in MarkovShield genauer erklärt.
Reverse Proxy (RP)	Ein Proxy, welcher vor andere Server (z.B. Webserver) geschaltet wird und Ressourcen für Clients von den Backend Servern holt.
Risk Level	Wert welcher die Kritikalität einer URL wiederspiegelt.
Session	Eine Session wird mit dem ersten Aufruf der Webseite angelegt und so lange behalten, bis sich der Benutzer wieder ausloggt oder der Session Timeout abläuft. Jede Session wird über eine eindeutige SessionUUID identifiziert.
Session Hiding	Webapplikation interne Cookies werden nicht bis zum Benutzer selbst geschickt, sondern stattdessen auf Reverse Proxy Level zurückbehalten und beim erneuten Aufrufen dem Request wieder angehängt. Die Korrelation geschieht dabei über die SessionUUID. Dies ermöglicht es interne Cookies vor dem Benutzer zu verstecken.
SMTP	Simple Mail Transfer Protokoll: Ein Kommunikationsprotokoll, welches extrem verbreitet ist, um Mails über versenden. Standardmässig ist der Payload von SMTP unterschlüsselt - eine sichere, über TLS verschlüsselte, Variante existiert und wir auch häufig eingesetzt.
SSO	Single-Sign-On: Einmalige Authentifizierung des Benutzers für mehrere Systeme
Standardabweichung	Die Standardabweichung trifft eine Aussage über die Streuung einer Zahlenmenge.

Step-Up	Als Step-Up wird die Authentifizierungslevel Hochstufung innerhalb des Apache 2 Modules MOD_BOT bezeichnet. Beispiel Levels sind: 0 = keine Benutzerauthentifizierung, 1 = Benutzer authentifiziert sich über Benutzername und Passwort und 3 = Authentifizierungslevel 1 aber zusätzlich braucht es noch einen Token (z.B. via SMS/Email).
SUSPICIOUS	Ein mögliches MarkovShield Engine Resultat das zurückgegeben wird, wenn eine Session vom historischen Benutzerverhalten abweicht und somit als potentieller Betrug erkannt wird.
TAN	Transaction Authentication Number: Ein Token, welcher in der Regel über einen zweiten Kommunikationskanal (z.B.: per SMS oder Mail) dem Benutzer geschickt wird, um die Identität des Benutzers zusätzlich zu überprüfen.
Transition Model	Im Kontext von MarkovShield ist das Transition Model ein Model, welches aufzeigt, mit welcher Wahrscheinlichkeit von einem aktuellen Webseiten Link auf einen anderen Link zugegriffen wird.
True Stream Processing	Als True Stream Processing wird ein theoretisch endloslaufendes Programm bezeichnet, welches den neuen Input unmittelbar nach dem Eintreffen.
User Model	Ein aus historischen Daten trainiertes Model, welches pro Benutzer angelegt, regemässig aktualisiert und abgespeichert wird. Die aktuellen Sessions werden jeweils gegen dieses Model verglichen um mögliche Anomalien detektieren zu können.
Dynamic Shared Object (DSO)	DSOs erlauben es, den Apache Server mittels Modulen dynamisch aufzubauen und somit um zusätzliche Funktionalitäten zu erweitern.
GNU Projekt Debugger (GDB)	Mittels GDB können Programme, welche mit C, C++ oder anderen Programmiersprachen geschrieben sind, debugged werden.
GNU Compiler Collection (GCC)	GCC steht für GNU Compiler Collection. Mittels GCC können mehrere Programmiersprachen wie C oder C++ kompiliert werden.
Apache eXtenSion tool (APXS)	APXS ist ein Tool welches erlaubt, Apache HTTPD Webserver Module kompilieren zu können. Im Hintergrund greift APXS auf GCC zu.

RegEx	RegEx steht für Regular Expression. Ein RegEx oder auch RegEx Pattern genannt, beschreibt eine Menge von Zeichenketten mit Hilfe von spezifischen Regeln.
UUID	Eine zufällige Nummer, welche ein Objekt eindeutig identifiziert.

IV.5. Bibliographie

alanthl. (16. September 2011). *EMW Cap OTP*. Von Technology Bloggers: http://www.technologybloggers.org/tag/emv-cap-otp/ abgerufen

angular.io. (29. Mai 2017). Abgerufen am 29. Mai 2017 von angular.io Homepage: https://angular.io/

Apache Flink. (kein Datum). *Introduction to Apache Flink®.* Von Apache Flink: https://flink.apache.org/introduction.html abgerufen

Apache Gearpump. (kein Datum). *Apache Gearpump (incubating): Overview.* Von http://gearpump.apache.org/overview.html abgerufen

Apache Kafka. (5. Mai 2017). *examples/kafka-streams/src/main/java/io/confluent/examples/streams/interactivequeries/.* Von https://github.com/confluentinc/examples/tree/master/kafka-streams/src/main/java/io/confluent/examples/streams/interactivequeries abgerufen

Apache Kafka. (kein Datum). *Apache Kafka Configuring The Log Cleaner.* Von kafka.apache.org: https://kafka.apache.org/documentation/#design_compactionconfig abgerufen

Apache Kafka. (kein Datum). *Apache Kafka Log Compaction Basics.* Von kafka.apache.org: https://kafka.apache.org/documentation/#design_compactionbasics abgerufen

Apache Kafka. (kein Datum). *Apache Kafka Potential breaking changes in 0.10.1.0.* Von kafka.apache.org: https://kafka.apache.org/documentation/#upgrade_10_1_breaking abgerufen

Apache Kafka. (kein Datum). *Apache Kafka Topics and Logs.* Von kafka.apache.org: https://kafka.apache.org/documentation/#intro_topics abgerufen

Apache Kafka. (kein Datum). *KTable (kafka 0.10.2.1 API)*. Von https://kafka.apache.org/0102/javadoc/org/apache/kafka/streams/kstream/KTable.html abgerufen

Apache Software Foundation. (kein Datum). *Apache kafka*. Von Apache kafka: https://kafka.apache.org/ abgerufen

Atlassian. (17. Mai 2017). *BitBucket Pipelines*. Von /product/features/pipelines: https://bitbucket.org/product/features/pipelines abgerufen

Cain, C. (22. Dezember 2014). *Analysing Man-in-the-Browser (MITB) Attacks*. Von SANS: https://www.sans.org/reading-room/whitepapers/forensics/analyzing-man-in-the-browser-mitb-attacks-35687 abgerufen

Commons CLI. (kein Datum). Von http://commons.apache.org/proper/commons-cli/ abgerufen

Confluent. (15. Dezember 2016). *Streaming in Practice - Putting Apache Kafka in Production.* Von slideshare.net: https://www.slideshare.net/ConfluentInc/streaming-in-practice-putting-apache-kafka-in-production abgerufen

Confluent Inc. (kein Datum). *Developer Guide.* Von docs.confluent.io: http://docs.confluent.io/3.0.0/streams/developer-guide.html abgerufen

Das, T., Zaharia, M., & Wendell, P. (30. Juli 2015). *Diving into Apache Spark Streaming's Execution Model*. Von The Databricks Blog: https://databricks.com/blog/2015/07/30/diving-into-apache-spark-streamings-execution-model.html abgerufen

DaveGamble. (10. Juni 2017). *Github.com*. Abgerufen am 10. Juni 2017 von DaveGamble/cJSON: https://github.com/DaveGamble/cJSON

Del Monte, V., Krettek, A., & al, e. (kein Datum). *Apache Flink Side Inputs Proposal.* Von https://docs.google.com/document/d/1hqUmrLY_wPTeS5bqG36Qq9P8LeDjZ_db61ky7OQy1hw/edit# abgerufen

Docker Inc. (26. Mai 2017). *Docker*. Von Install Docker: https://docs.docker.com/engine/installation/ abgerufen

Docker Inc. (26. Mai 2017). *Docker Compose*. Von Docker Compose: https://docs.docker.com/compose/ abgerufen

Edenhill, M. (26. Mai 2017). *Github*. Von Kafkacat Github Repository: https://github.com/edenhill/kafkacat abgerufen

Entrust Inc. (März 2014). *Defeating Man-in-the-Browser Malware*. Von entrust.com: https://www.entrust.com/wp-content/uploads/2014/03/WP_Entrust-MITB_March2014.pdf abgerufen

Farivar, R., & Knusbaum, K. (10. Juli 2016). *Performance Comparison of Streaming Big Data Platforms*. Von slideshare.net: https://www.slideshare.net/HadoopSummit/performance-comparison-of-streaming-big-data-platforms abgerufen

Flink. (kein Datum). *Apachje Flink 1.2.0 Documentation: SSL Setup*. Von ci.apache.org/projects/flink: https://ci.apache.org/projects/flink/flink-docs-release-1.2/setup/security-ssl.html abgerufen

Free Software Foundation, Inc. (05. Juni 2017). *gcc.gnu.org*. Abgerufen am 10. Juni 2017 von GCC, the GNU Compiler Collection: https://gcc.gnu.org/

Free Software Foundation, Inc. (10. Juni 2017). *gcc.gnu.org*. Abgerufen am 10. Juni 2017 von Options for Debugging Your Program: https://gcc.gnu.org/onlinedocs/gcc/Debugging-Options.html#Debugging-Options

GDB Developers. (04. Juni 2017). *gnu.org*. Abgerufen am 10. Juni 2017 von GDB: The GNU Project Debugger: https://www.gnu.org/software/gdb/

Götze, W., & Deutschmann, C. (2002). *Statistik. Lehr- und Übungsbuch mit Beispielen aus der Tourismus- und Verkehrswirtschaft.* . München. Von https://de.wikipedia.org/wiki/Normalverteilung abgerufen

Gühring, P. (12. 09 2006). *SecureClient*. Abgerufen am 2017 von futureware: http://www2.futureware.at/svn/sourcerer/CAcert/SecureClient.pdf

Guy, D. (2017). Von Slack abgerufen

Guy, D., & et. al. (03. Januar 2017). *KIP-99: Add Global Tables to Kafka Streams*. Von cwiki.apache.org: https://cwiki.apache.org/confluence/display/KAFKA/KIP-99%3A+Add+Global+Tables+to+Kafka+Streams#KIP-99:AddGlobalTablestoKafkaStreams-GlobalKTable abgerufen

hacking-lab.com. (07. Juni 2017). *hacking-lab.com*. Abgerufen am 07. Juni 2017 von hacking-lab.com: https://www.hacking-lab.com/index.html

Jackson Modules Base Afterburner. (kein Datum). Von https://github.com/FasterXML/jackson-modules-base/tree/master/afterburner abgerufen

Jackson Project Home @github. (kein Datum). Von https://github.com/FasterXML/jackson abgerufen

Jhguch, & Chen-Pan, L. (16. Mai 2012). *File:Boxplot vs PDF.svg*. Von commons.wikimedia.org: https://commons.wikimedia.org/wiki/File:Boxplot_vs_PDF.svg abgerufen

Juma, I. (1. Februar 2016). *Apache Kafka Security 101 - Confluent*. Von confluent.io: https://www.confluent.io/blog/apache-kafka-security-authorization-authentication-encryption/ abgerufen

Kreps, J. (25. Februar 2015). *Buildingf a Streaming Platform (Part 1).* Von confluent.io: https://www.confluent.io/blog/stream-data-platform-1/ abgerufen

Kumar, A. (15. September 2016). *Configuring Messages Retention in Apache Kafka.* Von allprogrammingtutorials.com: http://www.allprogrammingtutorials.com/tutorials/configuring-messages-retention-time-in-kafka.php abgerufen

Leonhardi, B. (24. Mai 2016). *Spark Streaming 2.0 is it suitable for Low Latency & Even based Real time Analyics*. Von Hortonworks: https://community.hortonworks.com/questions/35133/spark-streaming-20-is-it-suitable-for-low-latency.html abgerufen

librdkafka. (2017. April 14). *CONFIGURATION.md*. Von https://github.com/edenhill/librdkafka/blob/7d1d271695a5b703c52cac5242e2123032353cae/CONFIGURATION.md abgerufen

Libsmile. (kein Datum). Von https://github.com/pierre/libsmile abgerufen

Node.js Foundation. (29. Mai 2017). *NodeJS*. Abgerufen am 29. Mai 2017 von NodeJS Overview: https://nodejs.org/en/

OWASP. (5. Juni 2016). *Man-in-the-browser attack - OWASP*. Von owasp: https://www.owasp.org/index.php/Man-in-the-browser_attack abgerufen

Park, J., & Chung, H. (02. May 2009). *Consumers' travel website transferring behaviour: analysis using clickstream data-time, frequency, and spending*. Von tandfonline.com: http://www.tandfonline.com/doi/abs/10.1080/02642060903026254 abgerufen

Poloczek, J. (14. Februar 2017). *Testing Topologies in Kafka Streams.* Von semaphoreci.com: https://semaphoreci.com/community/tutorials/testing-topologies-in-kafka-streams abgerufen

redis. (kein Datum). *Redis Secuirty - Redis*. Von redis.io: https://redis.io/topics/security abgerufen

redislabs. (26. Mai 2017). *Redis-CLI*. Von redis-cli, the Redis command line interface: https://redis.io/topics/rediscli abgerufen

Slack. (17. Mai 2017). *Slack.com*. Von Slack.com: https://slack.com/ abgerufen

The Apache Software Foundation. (22. Mai 2017). *Apache Module mod_headers documentation*. Von Apache HTTPD: https://httpd.apache.org/docs/current/mod/mod_headers.html abgerufen

The Apache Software Foundation. (22. Mai 2017). *Apache Module mod_proxy*. Von Apache HTTPD: https://httpd.apache.org/docs/2.4/mod/mod_proxy.html abgerufen

The Apache Software Foundation. (22. Mai 2017). *Apache Module mod_rewrite*. Von Apache HTTPD: http://httpd.apache.org/docs/current/mod/mod_rewrite.html abgerufen

The Apache Software Foundation. (29. Mai 2017). *Apache Portable Runtime Project Library*. Abgerufen am 29. Mai 2017 von Home: https://apr.apache.org/

The Apache Software Foundation. (10. Juni 2017). *httpd.apache.org*. Abgerufen am 10. Juni 2017 von apxs - APache eXtenSion tool: https://httpd.apache.org/docs/2.4/programs/apxs.html

The Apache Software Foundation. (10. Juni 2017). *httpd.apache.org*. Abgerufen am 10. Juni 2017 von httpd - Apache Hypertext Transfer Protocol Server: https://httpd.apache.org/docs/2.4/programs/httpd.html

The Apache Software Foundation. (11. Juni 2017). *httpd.apache.org*. Abgerufen am 11. Juni 2017 von Multi-Processing Modules (MPMs) : https://httpd.apache.org/docs/2.4/en/mpm.html

The Apache Software Foundation. (11. Juni 2017). *httpd.apache.org*. Abgerufen am 11. Juni 2017 von Apache-MPM prefork: https://httpd.apache.org/docs/2.4/de/mod/prefork.html

The Apache Software Foundation. (11. Juni 2017). *httpd.apache.org*. Abgerufen am 11. Juni 2017 von Apache-MPM worker: https://httpd.apache.org/docs/2.4/de/mod/worker.html

The Apache Software Foundation. (11. Juni 2017). *httpd.apache.org*. Abgerufen am 11. Juni 2017 von Apache MPM event: https://httpd.apache.org/docs/2.4/en/mod/event.html

Thereska, E., Guy, D., Noll, M., & Narkhede, N. (26. Oktober 2016). *Unifying Stream Processing and Interactive Queries in Apache Kafka.* Von confluent.io: https://www.confluent.io/blog/unifying-stream-processing-and-interactive-queries-in-apache-kafka/ abgerufen

Trustwave Holdings, Inc. (22. Mai 2017). *ModSecurity Home*. Von ModSecurity: https://modsecurity.org/ abgerufen

Valgrind Developers. (10. Juni 2017). *valgrind.org*. Abgerufen am 10. Juni 2017 von valgrind.org Overview: http://valgrind.org/

Venkataraman, S. (2015). *Drizzle: Low Latency Execution for Apache Spark*. Von spark-summit.org: https://github.com/amplab/drizzle-spark abgerufen

Venkataraman, S. (14. Februar 2017). *Drizzle—Low Latency Execution for Apache Spark: Spark Summit East talk by Shivaram Venkataraman*. Von slideshare.net: https://www.slideshare.net/SparkSummit/drizzlelow-latency-execution-for-apache-spark-spark-summit-east-talk-by-shivaram-venkataraman abgerufen

Venkataraman, S., Panda, A., Ousterhout, K., Ghodsi, A., Franklin, M. J., Recht, B., & Stoica, I. (kein Datum). *Drizzle: Fast and Adaptable Stream Processing at Scale.* Von shivaram.org: http://shivaram.org/drafts/drizzle.pdf abgerufen

Wahl, J. (20. März 2013). *Describing Data: Why median and IQR are often better than mean and standard deviation*. Von dataz.io: https://www.dataz.io/display/Public/2013/03/20/Describing+Data%3A+Why+median+and+IQR+are+often+better+than+mean+and+standard+deviation abgerufen

Wai-Ki Ching, X. H.-K. (2013). Markov Chains Models, ALgorithms and Applications Second Edition.

webtatic.com. (11. Juni 2017). *webtatic.com*. Abgerufen am 11. Juni 2017 von PHP 7.0 on CentOS/RHEL 6.8 and 7.3 via Yum: https://webtatic.com/packages/php70/

Wikipedia.org. (5. Mai 2017). *Mobile TAN (mTAN)*. Abgerufen am 11. Juni 2017 von Wikipedia.org: https://en.wikipedia.org/wiki/Transaction_authentication_number#Mobile_TAN_.28mTAN.29

Wikipedia.org. (06. Mai 2017). *Wikipedia.org*. Abgerufen am 29. Mai 2017 von Wikipedia.org: Andrey Markov: https://en.wikipedia.org/wiki/Andrey_Markov

Wikipedia.org. (28. Mai 2017). *Wikipedia.org*. Abgerufen am 29. Mai 2017 von Wikipedia.org: Markov chain: https://en.wikipedia.org/wiki/Markov_chain

Wikipedia.org. (28. April 2017). *Wikipedia.org*. Abgerufen am 29. Mai 2017 von Wikipedia.org: Single-page Application: https://en.wikipedia.org/wiki/Single-page_application

Wikipedia.org. (06. März 2017). *Wikipedia.org: Rational Unified Process*. Abgerufen am 31. Mai 2017 von Wikipedia.org: https://de.wikipedia.org/wiki/Rational_Unified_Process

Williamson, W. (07. Juli 2014). *Stopping the Man in the Browser*. Von SecurityWeek.Com: http://www.securityweek.com/stopping-man-browser abgerufen

Zaharia, M., Das, T., Li, H., Shenker, S., & Ion, S. (2012). *Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters.* Von https://www.usenix.org/system/files/conference/hotcloud12/hotcloud12-final28.pdf abgerufen