

## Zielstellung und Rahmenbedingungen

Diese Dokumentation beschäftigt sich mit dem Thema „Angriffe auf AMQP-Messagebroker“. Begleitend zur Vorlesung „Betriebliche Informationssysteme“ galt es von uns in einem Praktikum zu identifizieren, welche potenziellen Angriffsvektoren existieren. Eingeschlossen ist hier auch das System „RabbitMQ“ welches das zu untersuchende Protokoll einschließt.

Ziel ist es bestimmt Angriffsvektoren aufzuspüren und zu bewerten. Durch Implementierung von einzelnen Clients auf Basis der RabbitMQ Java-Bibliothek sollen die Angriffe veranschaulicht werden. Mit veränderten Parametern richtet sich die Suche gezielt nach Faktoren die das System negativ beeinflussen und so die Verfügbarkeit stören, fokussiert auf „Denial of Service“. DoS kann dabei auf verschiedene Ressourcen bezogen sein (wie CPU, Arbeitsspeicher, Netzwerkbandbreite, ...).

Um die Begrenzungen ausfindig zu machen, gehört es zu Beginn zu unsere Aufgabe angemessene Werkzeuge zur Beobachtung der Ressourcen zu finden. Nur dann ist es möglich die Angriffe zu bewerten und ihre tiefere Auswirkung zu untersuchen.

Ferner sollen Vorschläge zur Schadensbegrenzung gegeben werden. Dazu gehört die Angabe auf welcher Ebene (Netzwerkebene, Protokollebene, ...) sich die Gefahren beseitigen lassen.

## Testumgebung

Als Betriebssystem für die Bereitstellung des RabbitMQ-Servers wurde ein Ubuntu Server 14.04.2 LTS mit Kernel 3.13.0-49 verwendet. Das Testsystem wurde dabei als Virtuelle Maschine (VM) in der Virtualisierungslösung „Virtualbox“ der Firma Oracle betrieben, um eine einfache Skalierung der Hardware, eine einfache Portabilität des Testsystem und einen reproduzierbaren Systemzustand, über die integrierte Snapshot-Funktion, bereitzustellen. Das Testsystem wurde über die Verwendung der in Virtualbox integrierten NAT-Funktion und der Portweiterleitung für alle von RabbitMQ benötigten Ports nach außen hin verfügbar gemacht.

Für die Beobachtung der CPU-, RAM-, Festplatten, Ressourcenverbrauches auf dem RabbitMQ-Server wurde das Kommandozeilenprogramm *glances* verwendet. Der Programmaufruf für jeden Test lautete wie folgt:

```
sudo glances -t 5 --disable-process
```

Es veranlasst die Ansicht alle 5 Sekunden zu erneuern und die Anzeige der aktuell laufenden Prozesse abzuschalten, um den Ressourcenverbrauch durch *glances* selbst zu reduzieren.

## Verwendete bzw. Erstellte Programme

RabbitMQ bietet einige Anwendungen, mit denen sich die Leistungsfähigkeit der Server messen lässt. Alle Hilfsprogramme sind innerhalb der *rabbitmq-client-tests.jar* vorzufinden. Diese JAR-Datei enthält weiterhin zahlreiche kleine Beispielpprogramme für das Testen der Funktionalität des eigenen Servers.

**Perftest** ist ein Performance-Test-Tool, welches beliebig viele Producer und Consumer erstellt und die Sende-/Empfangsrate misst. Zusammen mit der Latenzzeit werden alle Angaben auf der Konsole ausgegeben.

**HTML Performance Tools** bieten ebenfalls ein breites Spektrum an Funktionalität. Mit einer Reihe von Tools und Unterstützung von perftest lassen sich automatisierte Benchmarks erstellen. Die gelieferten Daten dienen dem Vergleich der Systems vor und während des Angriffs. Alle Ergebnisse werden in einer JSON-Datei gesichert und ansprechend in einer HTML-Seite dargestellt

**VirtualBox** Der von uns verwendete Server wurde durch die Virtualisierungslösung VirtualBox realisiert. Neben der einfachen Installation neuer Gast-Systeme besteht die Möglichkeit Sicherheitspunkte zu erstellen. Bei einem Ausfall des System kann so der ursprüngliche Sicherungspunkt wiederhergestellt werden. Als Grundlage dient die Servervariante von Ubuntu 14.04 LTS, welche durch die fehlende grafische Oberfläche zum einsparen von Ressourcen dient.

## Beschreibung der Anwendungsszenarien

S1	Direkte Nachrichten 1:1
----	-------------------------

---

<i>Beschreibung</i>	Ein Producer erzeugt kontinuierlich Nachrichten, die von einem Consumer kontinuierlich korrekt entnommen werden.
---------------------	--

<i>Aktion</i>	<code>perfTest -h</code>
---------------	--------------------------

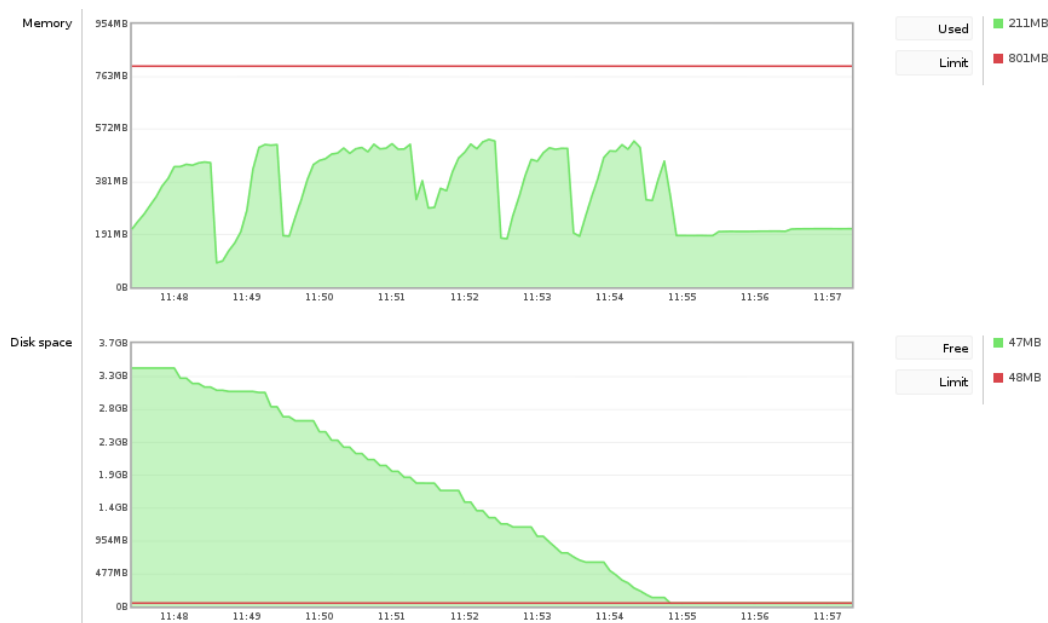
<i>Anmerkungen</i>	
--------------------	--

## Beschreibung der Angriffe

Beschreibung Standard Anwendung 1 Producer 5 Consumer während Anwendungsszenario läuft. Beschreibung Ressourcenverbrauch.

## A1 Ignorieren von Nachrichten

<i>Beschreibung</i>	Ein Producer erzeugt kontinuierlich Nachrichten gewisser Größe mit zufälligem Inhalt, die von einem oder mehreren Consumern empfangen, aber nicht quittiert werden. Der RabbitMQ-Server ist somit gezwungen, die Nachrichten in der Queue zwischenspeichern.
<i>Testparameter</i>	-dm NO (Aufrufparameter für Angriff) -c 5 (5 Consumer) -i 10 (10ms Pause zwischen 2 Consumer/Producer-Anfragen) -ms 102400 (Messagegröße: 10KB) -u <uri> (URI für Verbindung mit Server) 1 Producer (Standard) Nicht-Persistente Nachrichten (Standard)
<i>Befehlszeile</i>	<pre>Amqpstress -dm NO -c 5 -i 10 -ms 102400 -u amqp://testc:testp@localhost:5672/%2f</pre>
<i>Beobachtungen</i>	Nachrichten werden bis zur Hälfte des eingestellten Limits (hier 400MB) im RAM gehalten, bei überschreiten lagert der RabbitMQ-Server im Hintergrund die Nachrichten auf die Festplatte aus. Durch die Freigabe des RAM-Speichers entsteht ein Sägezahnmuster. Dieser Vorgang wiederholt sich bis die Festplatte voll ist und RabbitMQ alle Verbindungen schließt und in diesem Zustand verweilt. CPU-Last liegt konstant bei ca. 10%.
<i>Anmerkungen</i>	Verwendung von persistenten Nachrichten (-mp) steigert CPU-Last um x%.



---

## A2      Sofortiges Abweisen von Nachrichten

---

<i>Beschreibung</i>	Ein Producer erzeugt kontinuierlich Nachrichten gewisser Größe mit zufälligem Inhalt, die von einem oder mehreren Consumern empfangen, aber sofort abgewiesen (basic.Reject) werden. Der RabbitMQ-Server ist somit gezwungen, die Nachrichten in der Queue zwischenspeichern und erneut an den Consumer zu senden.
<i>Testparameter</i>	-dm REJECT (Aufrufparameter für Angriff) -c 5 (5 Consumer) -i 10 (10ms Pause zwischen 2 Consumer/Producer-Anfragen) -ms 102400 (Messagegröße: 10KB) -u <uri> (URI für Verbindung mit Server) 1 Producer (Standard) Nicht-Persistente Nachrichten (Standard)
<i>Befehlszeile</i>	<code>Amqpstress -dm REJECT -c 5 -i 10 -ms 102400 -mp -u amqp://testc:testp@localhost:5672/%2f</code>
<i>Beobachtungen</i>	
<i>Anmerkungen</i>	Verwendung von persistenten Nachrichten (-mp) steigert CPU-Last um x%.

---

**A3****Gebündeltes Abweisen von Nachrichten**

---

<i>Beschreibung</i>	Ein Producer erzeugt kontinuierlich Nachrichten gewisser Größe mit zufälligem Inhalt, die von einem oder mehreren Consumer empfangen, zunächst ignoriert werden, um sie bei Erreichen eines Schwellwertes gebündelt abzuweisen (basic.NACK). Dadurch ist der RabbitMQ-Server gezwungen alle Nachrichten zwischenspeichern und stoßweise alle Nachrichten bis zu der aktuellen Sequenznummer erneut zuzustellen.
<i>Testparameter</i>	-dm NACK (Aufrufparameter für Angriff) Schwellwert: 1000 Nachrichten (Standard) -c 5 (5 Consumer) -i 10 (10ms Pause zwischen 2 Consumer/Producer-Anfragen) -ms 102400 (Messagegröße: 10KB) -u <uri> (URI für Verbindung mit Server) 1 Producer (Standard) Nicht-Persistente Nachrichten (Standard)
<i>Befehlszeile</i>	<code>Amqpstress -dm NACK -c 5 -i 10 -ms 102400 -u amqp://testc:testp@localhost:5672/%2f</code>
<i>Beobachtungen</i>	
<i>Anmerkungen</i>	



---

**A4****Anlegen und Löschen von Queues**

---

<i>Beschreibung</i>	Ein Client erzeugt bis zu einem Schwellwert Queues und befüllt sie mit einer zufälligen Nachricht gegebener Größe. Wenn der Schwellwert erreicht ist, werden alle Queues ohne zu Warten gelöscht und der Zyklus beginnt erneut. Der RabbitMQ-Server muss die Überreste der Queues im RAM bzw. der Festplatte bereinigen, während neue Queues angelegt und befüllt werden.
<i>Testparameter</i>	-dq (Aufrufparameter für Angriff) -pc 10 (Schwellwert: 10 Queues) -i 10 (10ms Pause zwischen 2 Consumer/Producer-Anfragen) -ms 0 (Keine Nachrichten senden) -u <uri> (URI für Verbindung mit Server) 1 Client - Consumer und Producer zugleich (Standard) Nicht-Persistente Nachrichten (Standard)
<i>Befehlszeile</i>	<code>Amqpstress -dq -pc 10 -i 10 -ms 0 -u amqp://testc:testp@localhost:5672/%2f</code>
<i>Beobachtungen</i>	
<i>Anmerkungen</i>	

**A5****Versenden von Nachrichten mit großem Header**

<i>Beschreibung</i>	RabbitMQ bietet die Möglichkeit im Header der Nachricht bestimmt Parameter für die Weiterleitung zu deklarieren. Dieser Test beschäftigt sich mit der Auswirkung, wenn der Header unnötig ausgelastet wird. Das System ist gezwungen alle Weiterleitungsoptionen zu Prüfen, auch wenn diese keinem Ziel entsprechen.
<i>Testparameter</i>	-lh (Aufrufparameter für Angriff) -ms 10000 (Messagegröße: 10000 Byte) -hs 2500 (Headergröße - Anzahl Einträge in Map) -u <uri> (URI für Verbindung mit Server)
<i>Befehlszeile</i>	<code>Amqpstress -lh -ms 10000 -hs 2500 -u amqp://testc:testp@localhost:5672</code>
<i>Beobachtungen</i>	Die Anwendung generiert zu Beginn 2500 Weiterleitungsoptionen (Bestehend aus 8 Byte Key und 36 Byte Value = 44 Byte pro Eintrag) und schreibt sie in den Header jeder Nachricht. Hierdurch ist das System stark ausgelastet und der Durchsatz der Nachrichten schrumpft stark auf 10 - 20 Nachrichten pro Sekunde. Es macht keinen Unterschied ob der Key oder Value vergrößert wird, die geringe Übertragungsrate bleibt bestehen. Der Value wird daher nicht ausgewertet. Die CPU Last liegt bei 10 -15%
<i>Anmerkungen</i>	Die Headergröße ist bei etwa 2500 Weiterleitungsoptionen begrenzt. Wird diese überschritten lässt das System die Verbindung fallen, aufgrund einer zu großen Framegröße. Die Framegröße lässt sich beim Aufbau der Verbindung angeben, kann aber die bereits voreingestellten eingestellten 128 Byte nicht überschreiten.

Übertragungsrate nach Nachrichtengröße			Headergröße
10.000 Byte	1.000 Byte	100 Byte	Einträge (Byte)
140m/s	260m/s	350m/s	200 (8.800)
80m/s	120m/s	180m/s	500 (22.000)
30m/s	50m/s	70m/s	1.000 (44.000)
20m/s	30m/s	40m/s	2.000 (88.000)
10m/s	10m/s	20m/s	2.500 (110.000)
270m/s	3.000m/s	10.000m/s	kein Eintrag

**A6****Aufbauen mehrerer Channel über eine einzelne Verbindung**

<i>Beschreibung</i>	Neben einzelnen Verbindungen können in RabbitMQ auch mehrerer Kanäle aufgebaut werden. Hier stellt sich die Frage, wie das System mit einer Vielzahl von Kanäle zurecht kommt. Auf Basis einer einzelnen Verbindung wird das System so ausgelastet und beobachtet.
<i>Testparameter</i>	-mc (Aufrufparameter für Angriff) -ms 10000 (Messagegröße: 10000 Byte) -u <uri> (URI für Verbindung mit Server) -p 100 (100 Producer) -c 10 (10 Consumer)
<i>Befehlszeile</i>	<code>Amqpstress -mc -p 100 -c 10 -ms 10000 -u amqp://testc:testp@localhost:5672</code>
<i>Beobachtungen</i>	Das System ist stark ausgelastet. Ähneln aber der Auslastung unter der Erstellung mehrerer Verbindungen. Allerdings beansprucht der Aufbau der Channel extrem viel Zeit. Nach Aufbau aller Kanäle bricht die Übertragungsrate stark ein. Dabei liegt die CPU-Last bei etwa 45 - 55%
<i>Anmerkungen</i>	Zeit für Aufbau der Channel hängt stark von der Anzahl von Producer und Consumer ab. Nachfolgend zeigt sich ein Vergleich von mehreren Kanälen sowie mehreren Verbindungen.

**Tabelle 0.1:** Mehrere Channel

Übertragungsrate (schreiben)	Producer	Consumer	Nachrichtengröße (Byte)
7.000m/s	100	10	100
800m/s	100	10	1.000
100m/s	100	10	10.000

<b>CPU</b>	51.8%	<b>nice:</b>	0.0%	<b>LOAD</b>	2-core	<b>MEM</b>	39.8%
<b>user:</b>	40.2%	<b>irq:</b>	0.0%	<b>1 min:</b>	0.75	<b>total:</b>	1.95G
<b>system:</b>	8.2%	<b>iowait:</b>	0.6%	<b>5 min:</b>	0.61	<b>belegt:</b>	796M
<b>idle:</b>	48.2%	<b>steal:</b>	0.0%	<b>15 min:</b>	0.41	<b>frei:</b>	1.18G

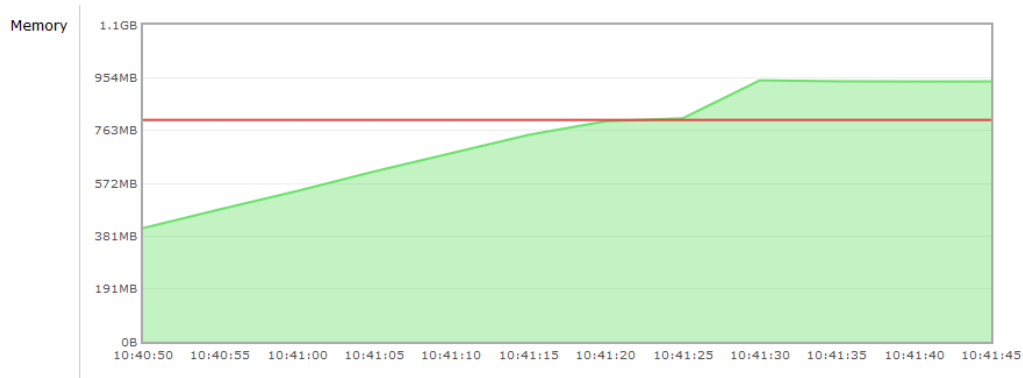
Tabelle 0.2: Mehrere Verbindungen

Übertragungsrate (schreiben)	Producer	Consumer	Nachrichtengröße (Byte)
24.000m/s	100	10	100
2.000m/s	100	10	1.000
100m/s	100	10	10.000

```
CPU      80.5%  nice:    0.0%  LOAD    2-core  MEM      54.1%
user:    66.1%  irq:     0.0%  1 min:   1.12  total:  1.95G
system:  8.2%   iowait:  0.4%   5 min:   0.50  belegt:  1.06G
idle:    19.5%  steal:   0.0%  15 min:  0.43  frei:    918M
```

**A7****Commit mehrerer Nachrichten im Transaktionsmodus**

<i>Beschreibung</i>	Im Transaktionsmodus ist es möglich mehrere Nachrichten als Folge zu übertragen, die aber vergleichbar mit Datenbanken als Einheit betrachtet werden. Nur durch die Commit()-Funktion wird der neue Zustand angenommen und die Nachrichten im System zur Verfügung gestellt. Ferner kann mit der Rollback()-Funktion der ursprüngliche Zustand wiederhergestellt werden. Zu untersuchen wäre die Situation, wenn permanent Nachrichten ohne commit gesendet werden - der Server muss die Nachrichten dann im Speicher belassen.
<i>Testparameter</i>	-tx (Aufrufparameter für Angriff) -p 100 (100 Producer) -ms 10000 (Messagegröße: 10000 Byte) -mct 10000 (Nachrichtenanzahl pro Producer: 10000) -co false (Commit der Transaktion - hier: Auslassen) -u <uri> (URI für Verbindung mit Server)
<i>Befehlszeile</i>	<pre>Amqpstress -tx -p 100 -ms 10000 -mct 10000 -co false -u amqp://testc:testp@localhost:5672</pre>
<i>Beobachtungen</i>	Durch Auslassen des Commits steigt der Speicherverbrauch stark an. Nach wenigen Sekunden ist die Speichergrenze erreicht und keine weiteren Nachrichten werden übertragen. Daraufhin verharren alle Producer bis der Speicher wieder freigegeben wird (Falls Consumer vorhanden)
<i>Anmerkungen</i>	Sporadisch friert der RabbitMQ-Server bei diesem Versuch ein. Ein Aufbau einer neuen Verbindung schlägt fehl und die Weboberfläche ist nicht mehr erreichbar. Nach längerer Zeit (etwa 10 Minuten) baut der Server aber wieder die Verbindungen ab und ein Zugriff ist wieder möglich. Dennoch bleibt der Speicherverbrauch bestehen.



---

**A8**      **Verzögerung des Verbindungsaufbaues**

---

*Beschreibung*      .

*Testparameter*      -u <uri> (URI für Verbindung mit Server)

*Befehlszeile*      `Amqpstress -u amqp://testc:testp@localhost:5672`

*Beobachtungen*

*Anmerkungen*

---

## A9      Ausnutzung des Hearthbeats

---

*Beschreibung*      .

*Testparameter*      -u <uri> (URI für Verbindung mit Server)

*Befehlszeile*      `Amqpstress -u amqp://testc:testp@localhost:5672`

*Beobachtungen*

*Anmerkungen*

## Auswirkungen der Angriffe



## Zusammenfassung und Fazit