JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

GK9.1: Security Concepts

Systemtechnik Labor
5BHIT 2017/18

Martin Wölfer

Version 0.1
Begonnen am 10.1.2018
Beendet am 18. Januar 2018

Note:
Betreuer: MICHT

Inhaltsverzeichnis

1 Einführung

Diese Uebung soll helfen die Funktionsweise eines Central Authentication Service (CAS). Dabei sollen erste Erfahrung mit den Technologien Single Sign On (SSO) und Ticket Granting Ticket Service (TGT) gemacht werden.

1.1 Ziele

Das Ziel dieser Uebung ist die Installation eines CAS-Servers und das Testen einer Anbindung mittels einem CAS-Clients. Die CAS Implementierung basiert auf einem CAS Protokoll, das mit Hilfe von 3 Teilen umgesetzt wird: der CAS-Client (meist ein Webbrowser) stellt eine Anfrage an ein Webapplikation stellt. Die Nutzung der Webapplikation erfordert einen Authentifikation, aus diesem Grund wird der Client an den CAS-Server weitergleitet. Wenn dieser Schritt erfolgreich war, dann wird vom CAS-Server ein Service-Ticket an den Client ausgestellt. Der Client wiederrum leitet das Ticket an die Webapplikation weiter, wo eine sichere Verbindung zwischen Webapplikation und CAS-Server aufgebaut wird, um das Ticket zu prüfen. Wenn diese Validierung erfolgreich war, kann das Resultat der Webapplikation an den Client zurückgegeben werden.

Die CAS-Architektur sieht folgendermassen aus:

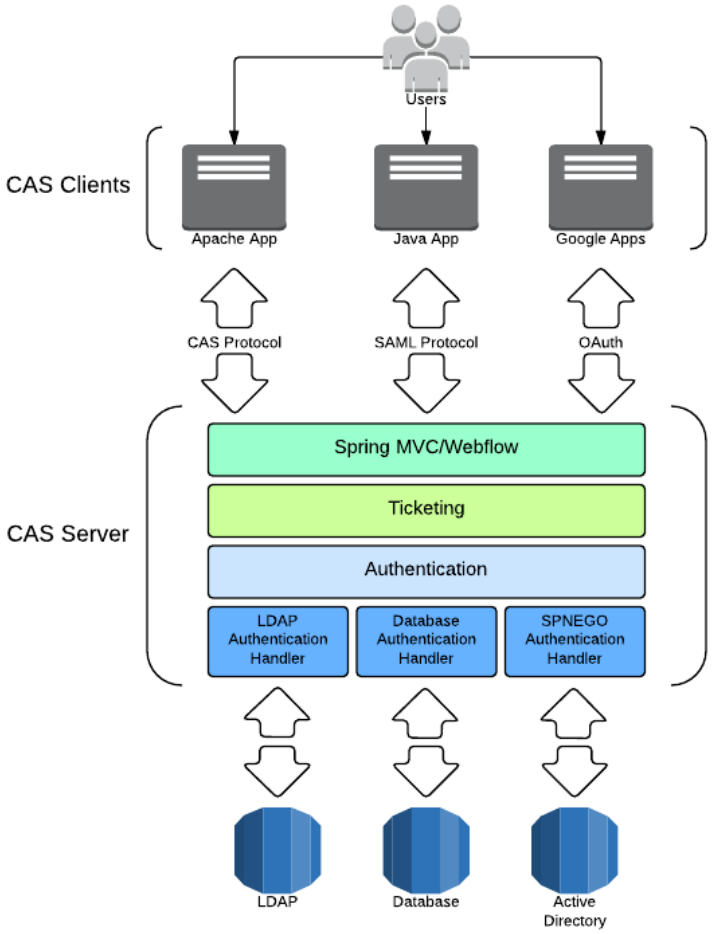

Abbildung 1: CAS Architektur

1.2 Voraussetzungen

- Grundlagen zu Central Authentication Service, Single Sign On- und Ticket Granting Ticket Service
- Java Programmierkenntnisse
- Installation Java JDK 1.8, Apache Tomcat
- Installation und Verwendung von Repository Management git
- Verwendung von Build Tools maven oder gradle

1.3 Aufgabenstellung

Installieren Sie eine CAS-Server und probieren Sie damit eine Webapplikation Ihrer Wahl einzubinden.

- Download: Apereo CAS 5.0.9
- Installation einer Demo Anwendung
- Entwicklung einer eigenen Webapplikation, die in der CAS Infrastruktur eingebunden wird
- Entwerfen Sie ein Konzept zum Testen der SSO- und TGT-Funktionalität

2 Versuche

Der erste Schritt für mich war es, es zu schaffen eine Demo laufen zu lassen. Am leichtesten lässt sich eine Demo ausführen, indem man eine Docker Instanz erstellt auf welcher ein vorkonfiguriertes image vorhanden ist.

2.1 cas-webapp-docker

[1]

Dies ist das offizielle GitHub repository von Apereo, um eine demo webapp in einer Docker Instanz laufen zu lassen.

2.1.1 Konfiguration

Zuerst muss natürlich das repository geklont werden:

```
1    git clone https://github.com/apereo/cas-webapp-docker
```

Der nächste Schritt ist es, das docker Konfigurationsfile, `Dockerfile`, anzupassen.

Hierbei muss vor allem darauf geachtet werden, dass das richtige **CAS Overlay Project** verwendet wird. D.h. es muss auf das CAS Overlay repository[2] gegangen werden, und die letzte Version in das `Dockerfile` eintragen.

2.1.2 Build

Nun muss das projekt gebuilded und ausgeführt werden. Dies funktioniert über die beiden Shell-Scripts `build.sh` und `run.sh`. Beide werden mit einem Parameter gestartet, **$CasVersion**, welcher bei beiden unbedingt gleich sein muss! Die tatsächliche Ausführung sieht dann beispielsweise folgendermaßen aus:

```
1  ./build.sh 1.0
   ./run.sh 1.0
```

2.1.3 Probleme

Das erste Problem war es, das README vom Repository zu verstehen. Es sind sehr viele wage Begriffe vorhanden und ich weiß noch immer nicht was genau es mit $CasVersion auf sich hat.

Das Hauptproblem war nun, wenn die Instanz gestartet wird, wird folgender Fehler ausgegeben:

```
***************************
APPLICATION FAILED TO START
***************************

Description:

The Tomcat connector configured to listen on port 8443 failed to start. The port may already be in use or the connector may be misconfigured.

Action:

Verify the connector's configuration, identify and stop any process that's listening on port 8443, or configure this application to listen on another port.
>
```

Abbildung 2: Fehler bei der Ausführung des Projekt cas-webapp-docker

2.1.4 Lösungsversuche

Der erste Versuch war, da die Fehlernachricht sagt der Port sei besetzt, den Port 8443 frei zu machen. Doch der Befehl `ss -lntu | grep 8443` gab nichts aus, also ist der Port definitv frei.

Ein weiteres Problem könnte im Dockerfile sein, zuerst habe ich Versucht die Java Versionen anzupassen da diese nicht mit meinem lokalen System übereingestimmt haben, aber nach der Erkenntnis das in einer abgekapselten Docker-Instanz gearbeitet wird, habe ich realisiert das es nicht an diesen Einstellungen liegen kann.

Tatsächlich wurde aber 10 Stunden vor der Labor-Einheit ein Issue[3] in dem Repository gepostet, welches die exakt gleiche Fehlermeldung hat!

2.2 cas

[4]

Das offizielle Cas Repository von Apereo, um einen CAS Server lokal aufsetzen zu können

2.2.1 Konfiguration

[5]

Das Problem hier bei ist, dass diese Konfiguration von einer lokalen Installation auf der Maschine ausgeht. Da das Ziel ist, eine dockerized-Installation aufzusetzen, war diese Dokumentation nicht sehr hilfreich.

2.2.2 Probleme

Es gibt zwar ein docker Ordner in dem Repository, aber in diesem ist kein `README.md` vorhanden, um zu verstehen wie die docker-Instanz gestartet werden muss.

2.3 cas-overlay-demo

[6]

Dies ist ein Projekt, mit dem Ziel eine Demo-webapp mit dem **Maven Overlay** laufen zu lassen.

2.3.1 Konfiguration

Da dies ein komplett vorgefertiges, und leicht deploybares maven Projekt ist, sollte keine Konfiguration benötigt werden.

2.3.2 Build

Der einzige command der ausgeführt werden muss, um das Projekt zu starten ist:

```
mvn clean install
```

2.3.3 Probleme

Das Projekt wurde erfolgreich installiert, nur ist das Problem für mich nun - was jetzt? In der Dokumentation steht folgender Ausschnitt:

" [..] and launch the CAS webapps via your favorite web application server with -Dcas.standalone.config=cas-overlay-server-demo/config. "

2.4 cas-overlay-docker

[7]

Dieses Projekt ist eine weitere Implementation des Overlay Mechanismus in CAS.

2.4.1 Konfiguration

Die docker-Instanz ist bereits vorkonfiguriert und sollte nur auszuführen sein mit:

```
1    docker-compose up --force-recreate
```

2.4.2 Probleme

Gleich bei der Ausführung des Befehls, wird eine Fehlermeldung ausgegeben:

```
Step 4/14 : COPY src/main/webapp/ /tmp/cas-overlay/src/main/webapp/
ERROR: Service 'cas' failed to build: COPY failed: stat /var/lib/docker/tmp/docker-builder700030628/src/main/webapp: no such file or directory
```

Abbildung 3: Fehler bei der Ausführung des Projekt cas-overlay-docker

2.5 docker-cas

[8]

Eine weitere docker Implementation von CAS, mit der zusätzlichen Funktion eine Anmeldungsdelegation installiert bzw. konfiguriert zu haben, mit welcher sich über Facebook, Twitter und OpenID Connect zur webapp verbinden könnte.

2.5.1 Build & Konfiguration

Nachdem `build.sh` ausgeführt wurde, gibt es Files welche zum konfigurieren sind: `cas.properties`, `pac4jContext.xml` und `server.xml`.

In `server.xml` kann der Hostname und der Port des Server geändert werden.

In `pac4jContext.xml` kann die Authentifikation via den 3rd-party Diensten konfiguriert werden. Da ich nicht sicher war, wie diese Anmeldungen über das Schul-WLAN funktionieren würden, habe ich sie deaktiviert indem ich die Einträge bei `cas.pac4j` auskommentiert habe.

2.5.2 Run

Nun muss einfach nur `run.sh` ausgeführt werden:

```
1 ./run.sh
```

Es müsste nun einfach auf die vorkonfigurierte Adresse (default **localhost:8080**) gegangen, und dann sollte der login via dem CAS Service funktionieren

2.5.3 Probleme

Das problem hierbei war komplex und versteckt. Der build hat gut funktioniert, doch nachdem versucht wird in die docker-Instanz zu wechseln, wird ein Fehler ausgegeben, dass das Image (**tb4mmaggots/tomcat**) weder **local** noch **remote** vorhanden ist. Es wird aufgefordert sich anzumelden, um das Image vom remote-Server zu beziehen, aber tatsächlich bin ich schon angemeldet.

Bei der Suche nach diesem Image, hat sich herausgestellt, dass das Projekt vor 2 Jahren erstellt wurde, und anscheinend in diesem Zeitbereich das verwendete Image von docker-hub gelöscht wurde, da es vom Benutzer **tb4mmaggots** nur folgende Images gibt:

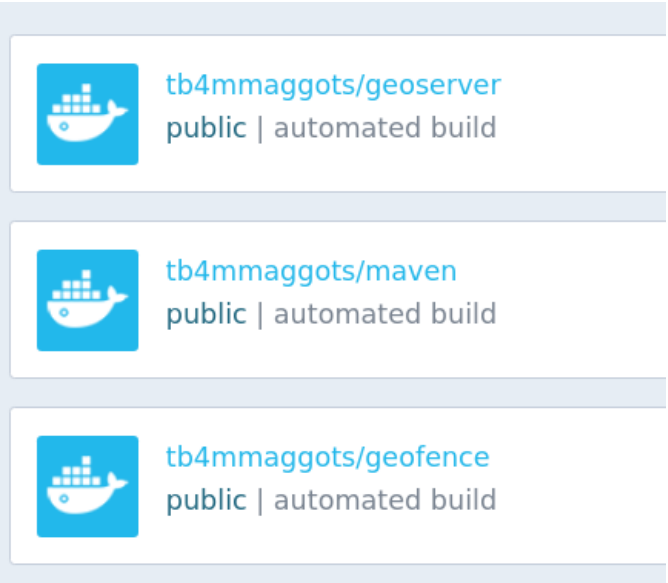

Abbildung 4: Verfügbare images von tb4mmaggots

2.5.4 Lösungsversuche

Der erste versuch war es, statt dem vorkonfigurierten image von tb4mmaggots, einfach ein normales tomcat image zu verwenden. Es hat funktioniert, die docker-instanz zu starten und auch in diese zu wechseln, aber auf dem nun verwendeten image ist nun kein CAS-Server installiert oder konfiguriert.

Weiter wurde versucht der **tb4mmaggots/geoserver** und **tb4mmaggots/maven** verwendet, aber beide images haben sich für diesen Zweck als nutzlos herausgestellt.

3 Ergebnisse

3.1 dockerized-cas

[9]

Diese Projekt bietet, so wie viele andere bereits erwähnte Projekte, ein bereits vorkonfiguriertes image an welches mit docker ausgeführt werden kann.

3.1.1 Run

Es ist nichts zu konfigurieren, es muss lediglich folgendes ausgeführt werden:

```
1 docker-compose up
```

Es sollte nun folgendes in der Konsole zu sehen sein:

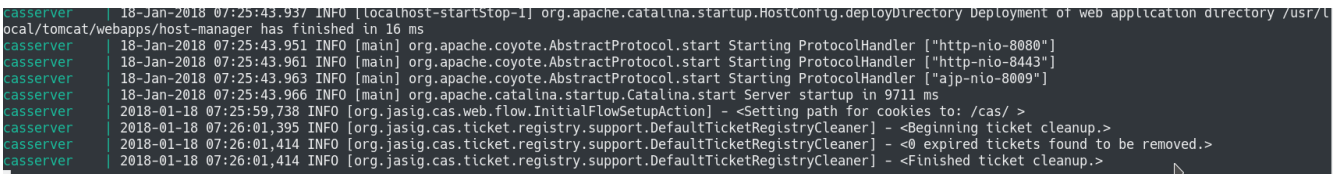

Abbildung 5: Tomcat Server wurde gestartet

3.1.2 Testing

Es wird nun auf **localhost:8080/cas** und man kann folgendes sehen:

Abbildung 6: Login ist möglich auf Tomcat Server via CAS

In der Dokumentation steht, das sich mit folgendem User anzumelden ist:

- User: casuser
- Password: Mellon

Bei der richtigen Angabe der Werte wird folgendes Ausgegeben:

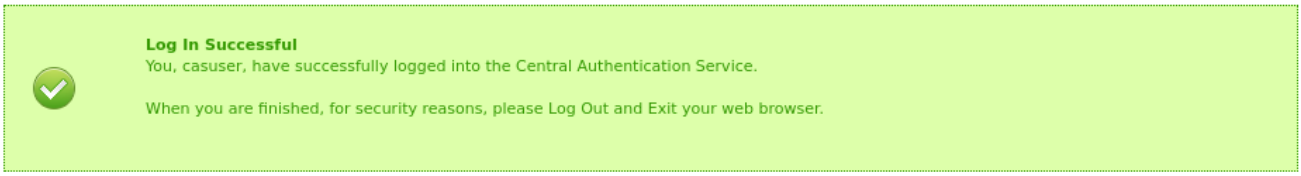

Abbildung 7: Login via CAS ist erfolgreich

Bei falschen Anmeldedaten gibt es folgende Ausgabe:

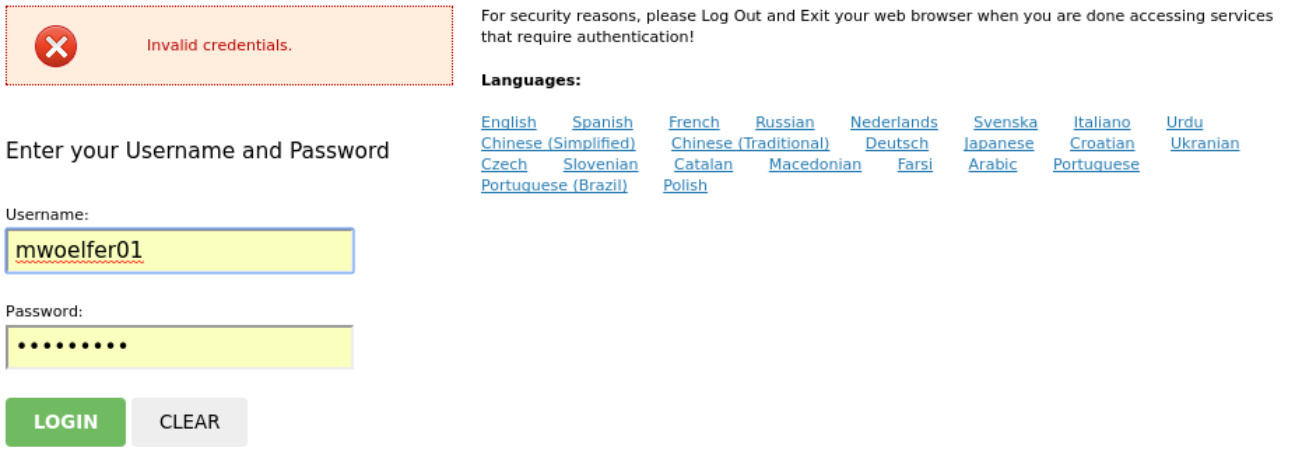

Abbildung 8: Login via CAS ist fehlgeschlagen

Literatur

[1] Apereo. https://github.com/apereo/cas-webapp-docker.

[2] Apereo. https://github.com/apereo/cas-overlay-template.

[3] Deric-dominic. https://github.com/apereo/cas-webapp-docker/issues/20.

[4] Apereo. https://github.com/apereo/cas.

[5] Apereo. https://apereo.github.io/cas/developer/Build-Process.html.

[6] Casinthecloud. https://github.com/casinthecloud/cas-overlay-demo.

[7] Crpeck. https://github.com/crpeck/cas-overlay-docker.

[8] Terranex. https://github.com/Terranex/docker-cas.

[9] J-fuentes. https://github.com/j-fuentes/dockerized-cas.

Abbildungsverzeichnis