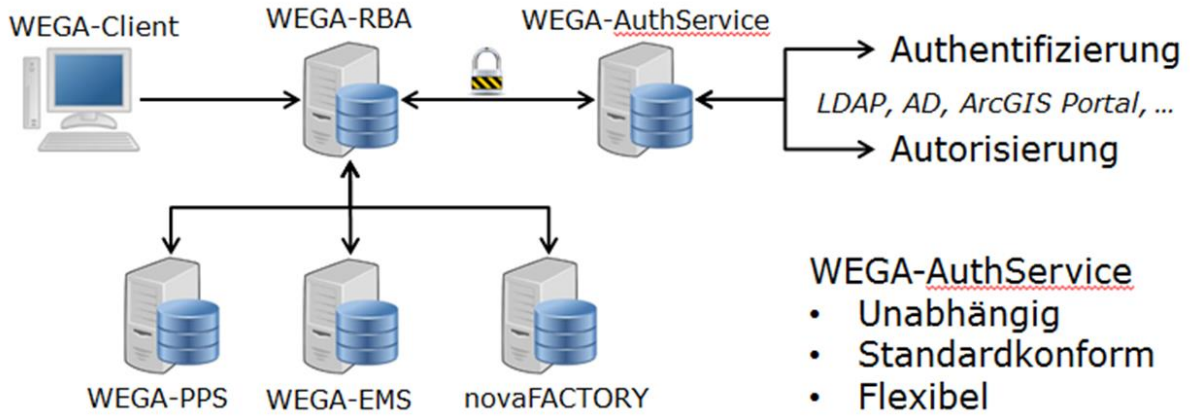
Sichere Authentifizierung per WEGA-AuthService

Seit novaFACTORY 7.3 und WEGA 8.3 kann die Authentifizierung über den M.O.S.S.-eigenen WEGA-AuthService auf Basis Apache Shiro vorgenommen werden.

WEGA-AuthService ist ein REST-Dienst, der die Authentifizierung/Autorisierung über eine Shiro-Realm-Konfiguration für Anfragen von WEGA-AuthClients durchführt. Die Clients übermitteln als Credentials entweder Username/Password oder ein Token und erhalten bei erfolgreicher Anmeldung die Liste der Rollen des angemeldeten Benutzers zurück. Um eine Anmeldung mit Token durchführen zu können, muss zuerst eine Anmeldung mit Username/Password erfolgt sein.



Der WEGA-AuthService ist insbesondere dann zu empfehlen, wenn die Authentifizierung (und die damit verbundene Autorisierung) redundanzfrei und einheitlich über unterschiedliche M.O.S.S.-Serverkomponenten (z.B. novaFACTORY und WEGA-EMS) genutzt werden soll. Außerdem erlaubt es die einfache Erweiterbarkeit des WEGA-AuthService auch spezielle Anforderungen an Authentifizierung und Autorisierung umzusetzen, die mit der bisherigen Umgebung der Tomcat-Valves und –Realms nicht realisierbar waren. Als Beispiel seien hier die passwortlose Anmeldung mit einem kryptographischen Token oder die Kombination mit ArcGIS Online genannt, die auch außerhalb unserer Produkte eingesetzt werden können.

Installation und Konfiguration

Für WEGA-AuthService steht ein separates Setup zur Verfügung, welches vor der Installation weiterer M.O.S.S.-Produkte installiert werden muss. Nach der Installation ist die zentrale Konfigurationsdatei shiro.ini anzupassen.

Die Konfiguration des WEGA-AuthService erfolgt in der Datei `<InstallDir>\WegaAuthService\webapps\WEB-INF\conf\shiro.ini`. Diese besteht mindestens aus den Sektionen [main] für die Konfiguration der benutzten Benutzerspeicher und der verwendeten internen Klassen und aus der Sektion [urls] für die Konfiguration der zu schützenden Pfade der Webapplikation. Im einfachsten Fall ist in [main] ein sogenannter IniRealm konfiguriert, der in etwa dem Tomcat MemoryRealm entspricht. Bei einem IniRealm stehen die Informationen über die Benutzer und deren Rollen – also die Informationen für Authentifizierung und Autorisierung – direkt in der Konfigurationsdatei shiro.ini. In dieser sind dazu die Sektion [users] für die Konfiguration der Benutzer, das jeweilige Passwort und die Rollen des Benutzers und die Sektion [roles] für die Rechte der Rollen hinzuzufügen. In der Sektion [users] ist das Format:

```
Benutzername = Passwort,Rolle1,Rolle2,Rolle3,…
```

In der Sektion [urls] gibt es die Möglichkeit zu unterscheiden, welche Funktion Pfade haben (z.B. Logout), ob sie geschützt sind (`authc`) oder ob sie für jeden ohne Anmeldung frei zugreifbar sind (`anon`). Hier ist es möglich weitere Filter einzusetzen, die die Funktionalität entsprechend erweitern. Einer dieser Filter ist z.B. die Einschränkung der Anmeldung auf bestimmte Rollen.

Eine solche Konfigurationsdatei würde dann beispielsweise so aussehen:

```
[main]
authc.loginUrl = /shirologin.jsp

[users]
RBAAdministrator = RBAAdministrator,RBAAdministrator,wegaems,novaFACTORY,emsDb
RBAUser = RBAUser,RBAAuskunft

[roles]
RBAAdministrator = *
RBAAuskunft = *.read
emsDb = *:*:none
novaFACTORY = *:*:none
wegaems = *:*:none

[urls]
/shirologin.jsp = authc
/logout.jsp = logout
/index.jsp = anon
[…]
```

Eine Konfiguration mit Authentifizierung über Active Directory anstatt des IniRealms ist im Folgenden angefügt. Die Änderungen gegenüber obigem Beispiel finden sich hauptsächlich in der Sektion [main], in der die Konfiguration für den Zugriff auf die Domain Controller konfiguriert ist. Hier sind hauptsächlich die Adresse des verwendeten Servers sowie ein Benutzername und Passwort für den Zugriff anzugeben. Alternativ können die LDAP-Suchausdrücke für den Zugriff auf die Benutzer und die Gruppen (=Rollen) angegeben werden, wenn diese nicht dem Standard entsprechen sollten. Die Sektionen [users] und [roles] sind hier nicht enthalten, da sich diese Informationen im Active Directory befinden.

```
[main]
cacheManager = org.apache.shiro.cache.MemoryConstrainedCacheManager
securityManager.cacheManager = $cacheManager

ldapContextFactory = de.moss.wega.auth.realm.JndiLdapContextFactory
ldapContextFactory.environment[java.naming.ldap.attributes.binary] = objectSid
ldapContextFactory.url = ldap://ad-server.domain
ldapContextFactory.systemUsername = aduser
ldapContextFactory.systemPassword = adpassword
ldapContextFactory.principalSuffix = @domain

adRealm = de.moss.wega.auth.realm.ActiveDirectoryRealm
adRealm.ldapContextFactory = $ldapContextFactory
adRealm.searchBase = "DC=domain"
adRealm.principalSuffix = @domain
adRealm.authorizationCachingEnabled = true

securityManager.realms = $adRealm

authc.loginUrl = /shirologin.jsp
```

```
[urls]
/shirologin.jsp = authc
/logout.jsp = logout
/index.jsp = anon
[…]
```

Im Deployment-Descriptor web.xml wird die Shiro-Konfiguration für die http-Zugriffe definiert:

```
<context-param>
  <param-name>shiroConfigLocations</param-name>
  <param-value>/WEB-INF/conf/shiro.ini</param-value>
</context-param>

<filter>
  <filter-name>ShiroFilter</filter-name>
  <filter-class>org.apache.shiro.web.servlet.ShiroFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>ShiroFilter</filter-name>
  <url-pattern>/*</url-pattern>
  <dispatcher>REQUEST</dispatcher>
  <dispatcher>FORWARD</dispatcher>
  <dispatcher>INCLUDE</dispatcher>
  <dispatcher>ERROR</dispatcher>
</filter-mapping>

<listener>
  <listener-class>
org.apache.shiro.web.env.EnvironmentLoaderListener
  </listener-class>
</listener>
```

Gleichzeitig muss die Container-based-security in web.xml deaktiviert oder gar nicht erst konfiguriert werden. Das betrifft folgende Einträge, die zu löschen oder zu deaktivieren sind:

- `<security-constraint>`
- `<login-config>`
- `<security-role>`