Articles

Authentication Handler in AEM: custom approach

Bobby Mavrov / September 23, 2020 / AEM Implementations / 5 Comments

Contents [show]

AEM offers developers the opportunity to implement their custom Authentication Handler with a full range of customization using the Sling Authentication APIs. To create a custom handler, we need to implement the *AuthenticationHandler* interface.

Why Create Custom Authentication?

There are many possible cases where users' authentication could be necessary besides the default Form authentication on the default login page. Of course, we could create our login page, but we'll be looking into different approaches to tackle additional requirements in this example.

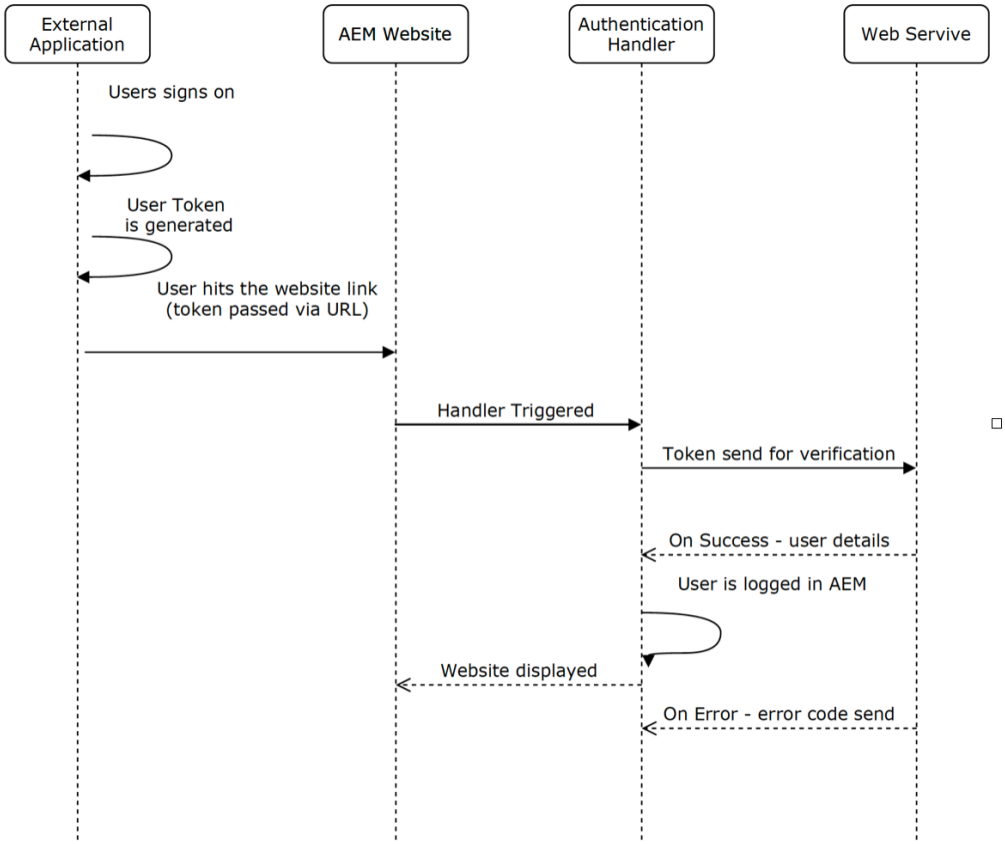
The following scenario presents an interesting example. In an organization where no Sigle Sing-On (SSO) has been implemented, the requirement is to have our website running on AEM be accessible only for users already logged in into another internal web application via a direct link generated by this second application.

The idea here is that these users will never access the actual default login page or any other login page. One way for them to log in would be to use the already authenticated application's link.

Solving The Problem With Custom Authentication Handler

A possible solution that we'll be delving into here is to have the second application external to AEM and generate a user token, passed as a URL parameter in the link to our AEM website. We want users to be authenticated to access our website as well. After receiving and verifying the request, our custom authenticator would then forward the token to a web service endpoint where it will be confirmed, and then user details will be returned upon success. Subsequently, our custom authenticator will then sign the user if it has already been created in AEM. If not, it will create it on the fly and then sign it in.

Solution Diagram



Complete code of the custom authentication

handler

```
package com.myproject.auth;


import com.day.crx.security.token.TokenUtil;
import org.apache.commons.lang.StringUtils;
import org.apache.http.HttpEntity;
import org.apache.http.NameValuePair;
import org.apache.http.client.entity.UrlEncodedFormEntit
y;
import org.apache.http.client.methods.CloseableHttpRespon
se;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.util.EntityUtils;
import org.apache.jackrabbit.api.JackrabbitSession;
import org.apache.jackrabbit.api.security.user.Authorizab
le;
import org.apache.jackrabbit.api.security.user.Group;
import org.apache.jackrabbit.api.security.user.User;
import org.apache.jackrabbit.api.security.user.UserManage
r;
import org.apache.sling.api.resource.ResourceResolver;
import org.apache.sling.api.resource.ResourceResolverFact
ory;
import org.apache.sling.auth.core.spi.AuthenticationHandl
```

```
er;
import org.apache.sling.auth.core.spi.AuthenticationInfo;
import org.apache.sling.jcr.api.SlingRepository;
import org.osgi.framework.Constants;
import org.osgi.service.component.annotations.Component;
import org.osgi.service.component.annotations.Reference;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import javax.jcr.RepositoryException;
import javax.jcr.Session;
import javax.jcr.SimpleCredentials;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@Component(service = CustomAuthenticator.class, immediate
= true, property = { "path=/content/mywebsite",
        Constants.SERVICE_RANKING +":Integer=60000", Cons
tants.SERVICE_DESCRIPTION +"=Custom Authenticator" })
public class CustomAuthenticator implements Authenticatio
nHandler {

    private static final String REQUEST_METHOD = "GET";
```

```
    static final String TOKEN_PARAMETER = "token";;
    private final Logger log = LoggerFactory.getLogger(Cu
stomAuthenticator.class);

    @Reference
    private ResourceResolverFactory resourceResolverFacto
ry;
    @Reference
    private SlingRepository repository;

    public AuthenticationInfo extractCredentials(HttpServ
letRequest request, HttpServletResponse response) {
        if (REQUEST_METHOD.equals(request.getMethod()) &&
(request.getParameter(TOKEN_PARAMETER) != null)) {
            Map<String, Object> param = new HashMap<>();
            param.put(ResourceResolverFactory.USER, "my-s
ystem-user");
            ResourceResolver resolver = null;
            try {
                String userId = obtainUserId(request.getP
arameter(TOKEN_PARAMETER));
                if(userId != null) {
                    resolver = resourceResolverFactory.ge
tServiceResourceResolver(param);
                    UserManager userManager = ((Jackrabbi
tSession) resolver.adaptTo(Session.class)).getUserManager
();
                    Authorizable user = userManager.getAu
thorizable(userId);
```

```
                    if(user == null) {
                        createNewUser(userManager, userI
d);
                        resolver.commit();
                    }

                    Session session = this.repository.log
in(new SimpleCredentials(userId, userId.toCharArray()));

                    if (session != null) {
                        return createAuthenticationInfo(r
equest, response, session.getUserID());
                    }
                }
            } catch (Exception e) {
                log.error("Exception in extractCredential
s while processing the request {}", e);
            }finally {
                if(resolver != null && resolver.isLive())
                    resolver.close();
            }
        }
        return null;
    }

    private AuthenticationInfo createAuthenticationInfo(H
ttpServletRequest request, HttpServletResponse response,
                                                        S
tring userId) throws RepositoryException {
```

```
        return TokenUtil.createCredentials(request, respo
nse, this.repository, userId, true);
    }

    public void dropCredentials(HttpServletRequest arg0,
HttpServletResponse arg1) {}

    public boolean requestCredentials(HttpServletRequest
request, HttpServletResponse arg1) {return true;}

    public String obtainUserId(String token) {
        HttpPost httpPost = new HttpPost("http://localhos
t:8080/api/user/getUserId");
        CloseableHttpClient httpClient = HttpClients.crea
teDefault();
        try (CloseableHttpResponse response = httpClient.
execute(httpPost)){
            List<NameValuePair> urlParameters = new Array
List<>();
            urlParameters.add(new BasicNameValuePair(TOKE
N_PARAMETER, token));
            httpPost.setEntity(new UrlEncodedFormEntity(u
rlParameters));
            HttpEntity entity = response.getEntity();
            if (response.getStatusLine().getStatusCode()
!= 200) {
                log.error("Unable to obtain user id from
web service! Status: " + response.getStatusLine().getStat
usCode());
```

```
                return null;
            }

            String userId = StringUtils.EMPTY;
            String output;
            BufferedReader bufferedReader = new BufferedR
eader(new InputStreamReader(response.getEntity().getConte
nt()));
            while ((output = bufferedReader.readLine()) !
= null) {
                userId = userId + output;
            }
            EntityUtils.consume(entity);
            return userId;
        } catch (Exception e) {
            return null;
        }
    }

    private boolean createNewUser(UserManager userManage
r, String userId) {

        try {
            Group group = (Group) userManager.getAuthoriz
able("dam-users");
            Authorizable user = userManager.getAuthorizab
le(userId);
            if (user == null) {
                user = userManager.createUser(userId,user
```

```
Id);
                group.addMember(user);
            } else if (!group.isMember(user)) {
                group.addMember(user);
                if(((User) user).isDisabled())
                    ((User) user).disable(null);
            }
        }catch (Exception e){
            log.error("Error while creating new user! ",
e);
            return false;
        }
        return true;
    }

    protected void bindRepository(SlingRepository paramSl
ingRepository) {
        this.repository = paramSlingRepository;
    }

    protected void unbindRepository(SlingRepository param
SlingRepository) {
        if (this.repository == paramSlingRepository) {
            this.repository = null;
        }
    }
}
```

Check if the Handler is Active After Deployment

We can verify our successful deployment here by going to *http://localhost:4502/system/console/slingauth*

Registered Authentication Handler

Path	Handler
/linksharepreview.html	Adhoc Asset Share Authentication Handler
/linkshare.html	Adhoc Asset Share Authentication Handler
/linkexpired.html	Adhoc Asset Share Authentication Handler
/libs/dam/gui/content/assets/assetlinkshare	Adhoc Asset Share Authentication Handler
/libs/dam/gui/content/adhocassetsharepage/linksharepreview	Adhoc Asset Share Authentication Handler
/libs/dam/gui/content/adhocassetsharepage/linkexpiredpage	Adhoc Asset Share Authentication Handler
/libs/dam/gui/content/adhocassetsharepage	Adhoc Asset Share Authentication Handler
/libs/cq/i18n	Adhoc Asset Share Authentication Handler
/etc/cloudservices/twitterconnect	AEM Communities OAuth - Social Auth Security Handler
/etc/cloudservices/facebookconnect	AEM Communities OAuth - Social Auth Security Handler
/content/mywebsite	Custom Authenticator
/	ImageServer Authentication Handler
/	Day CQ Login Selector Authentication Handler
/	Adobe AEM Screens Authentication Handler
/	Granite Client Certificate Authentication Handler
/	Token Authentication Handler
/	HTTP Basic Authentication Handler (enabled)

When the New Handler Will Come Into Place

Let’s first review under what conditions the authenticator will be activated. When we declare the OSGi component we set the following properties:

```
@Component(service = CustomAuthenticator.class, immediate
= true, property = { "path=/content/mywebsite",
        Constants.SERVICE_RANKING +":Integer=60000", Cons
tants.SERVICE_DESCRIPTION +"=Custom Authenticator" })
```

path=/content/mywebsite - the authenticator will be used for requests to the website only

service.ranking:Integer=60000 - we need higher value to make sure the service is involked to handle the authentication requiest

After the component is hit another check is performed:

```
if (REQUEST_METHOD.equals(request.getMethod()) && (reques
t.getParameter(TOKEN_PARAMETER) != null))
```

the handler will proceed with its execution only if the request is of type GET and if it contains parameter *token*. This will prevent users who have been logged out accessing the website without the necessary token if they have the link stored somewhere for example.

In case any of those conditions are not met the authenticator would be ignored but users could still login directly via the login page. If we want to avoid that we can disable the anonymous access to it via *Apache Sling Authentication Service*. We can also filter the access to it via our Publish Dispatcher. We could also modify the default login page for the default handler via *Adobe Granite Login Selector Authentication Handler*.

Once The Handler Has Been Triggered…

After obtaining the token parameter, our authenticator calls *obtainUserId()* where another request is created to the REST Web Service with the token again added as URL parameter. The function of this service is not part of the scope of this blog. The service does its job, and if the token is verified it returns whatever user details are be necessary. In our example only user name. If the token is not verified, then null is returned.

Time to sign in the user

After we have the user name we need to check if the user already exists in our AEM user repository using UserManager API.

```
Authorizable user = userManager.getAuthorizable(userId);
                    if(user == null) {
                        createNewUser(userManager, userI
d);
                        resolver.commit();
                    }
```

User here could have been manually created earlier or via some service obtaining them from LDAP or database.

If the user does not exist, though, *createNewUser()* is called where they are created with the username as parameter. For simplicity we'll use the username as password as well utilizing UserManager API again.

```
user = userManager.createUser(userId,userId);
```

This example illustrates how useful custom authentication handlers in AEM can be and how easy it is to implement one for the needs of your organization.

Contact Us

Tags:Adobe Experience Manager, AEM, AEM Authentication

5 Comments on **“Authentication Handler in AEM: custom approach”**

Karo

10.08.2020 AT 4:28 AM

REPLY

Just a thought, but it seems that your requests might hang. I’d suggest to use a try-with-resource for the closable object and invoke the EntityUtils to consume the entity otherwise your connection might hang,

ref:
https://hc.apache.org/httpcomponents-client-4.5.x/quickstart.html

★ Loading...

REPLY

Bobby Mavrov

10.08.2020 AT 4:30 PM

Hi Karo, thank you for pointing it out. I've updated the example code to include your suggestions.

★ Loading...

REPLY

mahaboobalishah

02.08.2021 AT 6:12 AM

We are storing AEM users same way as you have mentioned above. In our application 1000 of users will login everyday. So we are creating 1000 of users.
We are using userManager.getAuthorizable to check if the user is exits in AEM and this is taking more time when we have 1000's of users.
We are facing performance issue. What do you think can help here. I have created index for users but it did not help.

★ Loading...

REPLY

Bobby Mavrov

02.09.2021 AT 10:08 AM

Hi mahaboobalishah,

If I understand correctly you are actually creating 1000’s of unique users every day. Is that indeed the requirement for your application? One thing that could improve the performance in your case might be to store the user’s data somewhere else as well inside the AEM repository like in a ValueMap (like .properties file) or JSON. Then check against that if your user exist or it is active. In your custom authenticator you could also add functionality to update that file after user is successfully created but only if you don’t notice performance issues. If you do then the update could be done by additional service. You might need 2 of them. One to monitor for user updates from other and recreate them in your file, one to expose isUserActive(String userId) method to check against that file, as you can use that method in your authenticator. You also have to be cautious not to expose the user list to the public.

REPLY

Guru

04.17.2023 AT 11:31 AM

Tokenutil.create is deprecated

Loading...

Leave a Reply