Qlik Sense Support Tips

Qlik Sense Ticket based authentication - step by step configuration example

Posted by Vincenzo Esposito ★ in Qlik Sense Support Tips on May 11, 2016 8:40:31 AM

I often talk with people who requires more informations about how Qlik Sense based Authentication works. There is a nice, well made, article from Fredrick Here, which explain the concept behind this authentication method; moreover the documentation on-line can give you further details.
Despite these resources, I find myself to re-elaborate them in a simpler way for non techincal people giving them the ability to involve the right staffs, or give further details on how to implement it. This article aims to help those who experience this pains.

In general, a ticket authentication is just one of different ways for a system to identify whether or not a user has the access right. It is a bit like a bangle which allows you to have as many drinks as you want, in an all-inclusive club. In that case you pay once, before or when you arrive at the club's reception (authentication), the receptionist give you the bangle (ticket in the digital word) that allow you to ask for (request) all drinks you want.
In the above example there are two parts involved in the process: the club, who need to deliver the right service to the right costumer, and you (the resource's consumer).
Ticket authentication in Qlik is just a bit more complex due to a new part in the process.
That new part is the "Authentication Module" which needs to check your identity on something you know, you have or you are and ask a ticket to Qlik on your behalf (the resource's consumer). Authentication Module subsequently forwards your request to Qlik, including the ticket in the header request, which eventually, allows you to access Qlik resources.
If the above definition is clear, this post is definitely not for you, please refer to the previous article and to the help on-line. Otherwise, in this article, you can find out a simpler way to explain the above definition and a step-by-step configuration example.

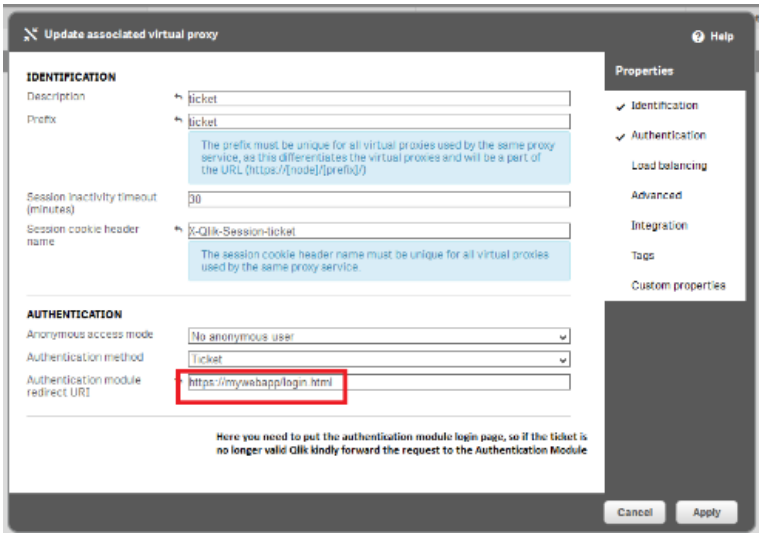
Let's figure out how it works in a (similar) real life example. Picture a coffee shop or a pub where you need to pay your orders first. If you order for a drink, the barkeeper asks to see your receipt and check if the receipt is valid and whether your order match what you have actually payed. He serves you the drink, then mark the receipt as "served", so it's no longer valid for another order. If you have no receipt, the barkeeper kindly "forwards" you to the cashier, who ask for a payment (authentication) before give you the right receipt (ticket).
In the digital world, the barkeeper role, is played by Qlik, which serves analytics instead of drinks; the cashier is executed by your web site's Authentication Module which ask for the ticket (receipt) to Qlik instead of cash desk, finally the customer's roles is played by your customer which need to log-in to your application in order to consume the analytics served by Qlik.
Let's sophisticate just a little the above example. In the real world no one except the cashier, can deliver the receipts. This statement is not completely true in the digital world, where any server on the network can pretend to be the cashier and ask to Qlik a ticket on behalf of fake users. How to fix this issue? Qlik verify the identity of the cashier (Authentication module) asking for a ID card (Certificate) if it's valid, Qlik release a valid ticket.
Let's see what I've just said, how becomes a configuration example:

Ingredients

1. Your web application, with an authentication process already set in place
2. A valid Qlik Sense license (Qlik Sense Desktop does not support any authentication)
3. A basic knowledge of HTML and JavaScript is required, as well as a server side language knowledge. A web developer is really appreciate!
4. Willingness to adapt the current example to your real case, since each situation is different from each others

Step 1

- Let's create a new virtual proxy and name it as your convenience. You can find further details here
- Select as Authentication method Ticket.



Step 2

Do you remember the ID Card? Well, now it's time for Qlik to release that identification document called "Certificate". You can find out more on how to export Certificate here .

Step 3

Let's educate our Authentication Module to ask for a ticket on the user login action. The authentication module run on the server side, receive some information about the user credential (most of the time in terms of user and password) and if correct allow the user to access in the web application.
A "pseudo-code" should look like this:

```
if(checkUserCredential(user,passwd)){
   // Here the user is authenticated, so we can give the access to our Web App
   AskTicketToQlik(userName, userDir);
}else {
   // In this case the checkUserCredential fails so we're going to display a log-in error message
}
```

In the above example "checkUserCredential" is a your function to check whether the user is allowed to access into the system. In the simplest case, get some parameter as User/Password to do that task, but you can also imlement a very complex function which take into account the user thumbprints, their eye's retina recognize the tone of voice or all the coolest staff you have in mind. It's really up to you.
AskTicketToQlik is the function you need to implement to ask a valid ticket in case the user is authenticated.

Step 4

Let's write the AskTicketToQlik. This function need to know on behalf of which user Qlik need to generate a valid ticket. Normaly Qlik identify any user with the user name and the user directory in the for (<userDirectory>\<userName>).
The first thing to do is to show the ID Card (the Certificate exported at Step 2) to Qlik Sense to prove this is the real Authentication Module and has the right to ask for a ticket.

```
/************** BEGIN Certificate Acquisition **************/
String certFolder = "c:\\javaTicket\\"; //This is a folder reference to the location of the jks files used for securing ReST communication
String proxyCert = certFolder + "client.jks"; //Reference to the client jks file which includes the client certificate with private key
String proxyCertPass="secret"; //This is the password to access the Java Key Store information
```

```
String rootCert = certFolder + "root.jks"; //Reference to the root certificate for the client cert. Required in this example because Qlik Sense certs are used.
String rootCertPass = "secret"; //This is the password to access the Java Key Store information
/************** END Certificate Acquisition **************/
```

Than we need to configure the certificate for the connection

```
/************** BEGIN Certificate configuration for use in connection **************/
KeyStore ks = KeyStore.getInstance("JKS");
ks.load(new FileInputStream(new File(proxyCert)), proxyCertPass.toCharArray());
KeyManagerFactory kmf = KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
kmf.init(ks, proxyCertPass.toCharArray());
SSLContext context = SSLContext.getInstance("SSL");
KeyStore ksTrust = KeyStore.getInstance("JKS");
ksTrust.load(new FileInputStream(rootCert), rootCertPass.toCharArray());
TrustManagerFactory tmf = TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
tmf.init(ksTrust);
context.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
SSLSocketFactory sslSocketFactory = context.getSocketFactory();
/************** END Certificate configuration for use in connection **************/
```

Subsequentially we're going to present the certificate to Qlik Sense

```
/************** BEGIN HTTPS Connection **************/
System.out.println("Browsing to: " + "https://" + host + ":4243/qps/" + vproxy + "/ticket?xrfkey=" + xrfkey);
URL url = new URL("https://" + host + ":4243/qps/" + vproxy + "/ticket?xrfkey=" + xrfkey);
HttpsURLConnection connection = (HttpsURLConnection ) url.openConnection();
connection.setSSLSocketFactory(sslSocketFactory);
connection.setRequestProperty("x-qlik-xrfkey", xrfkey); connection.setDoOutput(true);
connection.setDoInput(true);
connection.setRequestProperty("Content-Type","application/json");
connection.setRequestProperty("Accept", "application/json");
connection.setRequestMethod("POST");
/************** BEGIN JSON Message to Qlik Sense Proxy API **************/
String body = "{ 'UserId':'" + args[0] + "','UserDirectory':'" + args[1] +"',";
body+= "'Attributes': [],"; body+= "}"; System.out.println("Payload: " + body);
/************** END JSON Message to Qlik Sense Proxy API **************/
```

Finally we're going to get the response from Qlik Sense.

```
OutputStreamWriter wr= new OutputStreamWriter(connection.getOutputStream());
wr.write(body);
wr.flush(); //Get the response from the QPS BufferedReader
in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
StringBuilder builder = new StringBuilder();
String inputLine;
while ((inputLine = in.readLine()) != null)
{
                                                                builder.append(inputLine);
}
in.close();
String data = builder.toString();
System.out.println("The response from the server is: " + data);
```

/********** END HTTPS Connection **********/

The ticket got from Qlik will be use for the actual request.

Ticket.java
4.1 K

64 Views Categories: Documents Tags: authentication, ticket, web ticket, qlik sense authentication

0 Comments

There are no comments on this post