



# QlikView Security

Customized authentication

January 2013

Version 2.0

Author: Fredrik Lautrup, Michael Bienstein

## Table of Contents

Introduction	4
Overview of authentication	4
Header solution	5
When to use	5
Architecture	6
Security	6
Trust	6
Examples	6
WebTicket	8
When to use	8
Architecture	9
Security	10
WebTicket protocol	10
Custom-developed authentication through Authenticate.aspx	11
When to use	11
Architecture	12
Security	12
Example	13
How to choose solution	13
Appendix A: PHP WebTicket code	14
Appendix B: .NET WebTicket code	16
Appendix C: Authenticate.aspx for LDAP	18
Appendix D: Detailed description of the Authentication API	21
How to use the API	21
Representing users	21
Representing web requests and configuration	22
Creating or obtaining an IUser object	22
Creating a generic IUser object	22
Representing a Windows account	23
Creating an IUser based on a name and getting groups from the DSC	24
Logging in the IUser and performing re-authentication	25
Logging in once only	25

Logging in and performing re-authentication each time

26

## Introduction

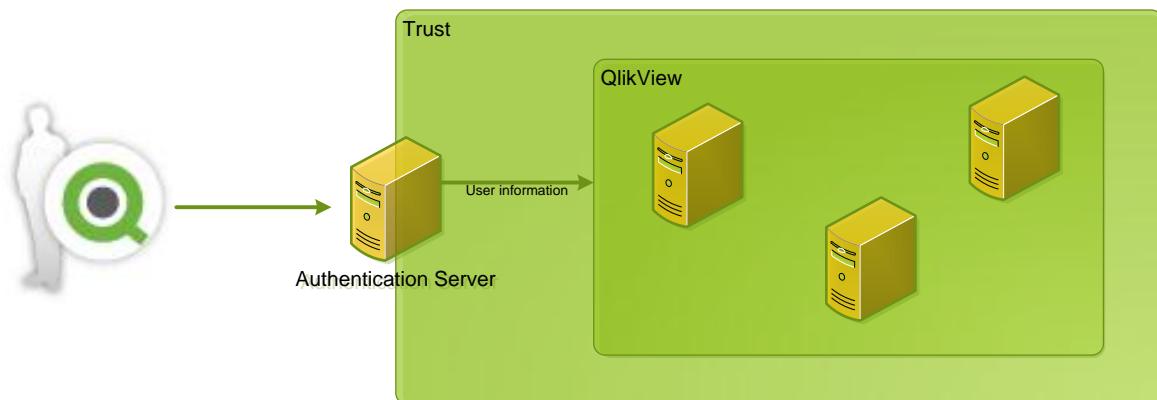
This document describes good practices regarding the implementation architecture for authentication solutions using web clients for QlikView version 11. Three different approaches, including the pros and cons of each, to the authentication architecture are described.

The document includes examples of implementations. However, each implementation is unique and has to be customized, which is why it is important that each solution is thoroughly tested before used in production.

## Overview of authentication

The basic principles for customization of authentication in QlikView are flexibility and trust. To achieve flexibility in the authentication of users to QlikView, authentication should be performed outside the QlikView platform. To make this as secure as possible in QlikView, a trust has to be established between the authenticating party and QlikView. Based on this trust, QlikView can trust the identity of the user supplied by the authenticating party.

This approach allows QlikView to support a wide number of authentication methods without having to implement them in QlikView.



The procedure for authenticating a user is as follows:

1. Send the credentials to the authenticating system.
2. The system verifies the credentials.
3. The user information is transferred to the QlikView platform.

The first two steps are handled by the external authentication system when authentication integration has been implemented. The third step is performed by QlikView. This document describes three methods for the third step of transferring user information to QlikView and when and how the different methods can be used. It also examines how the QlikView web code that runs

in the web browser (HTML and JavaScript) interacts with the customized authentication systems for steps 1 and 2.

## Header solution

In the header solution, a trust relationship (Trust Zone 1) is established between the portal or web frontend and the QlikView Web Server (QVWS) or Microsoft IIS using one of the following alternatives:

- Implicit trust: The authentication system runs in the same web server as QlikView.
- IP address restrictions that ensure that the web requests come from a specific machine that is known to run the authentication system
- Additional authentication that proves that the request comes from the authentication system (for example, if using IIS, it is possible to force the authentication of a Windows account representing a portal via SSL using certificate authentication, basic authentication, or even NTLM)

When the trust relationship has been established, the QlikView environment trusts the user information supplied by the portal or web frontend.

### When to use

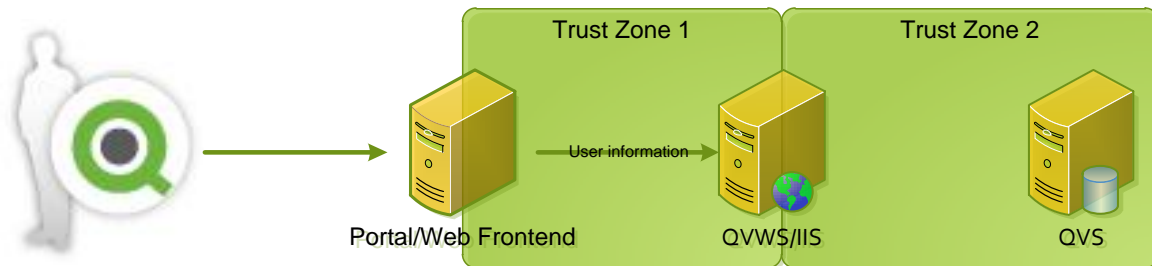
Many commercial and open source identity federation products allow for the separation of authentication from business applications like QlikView. The applications receive information about users by transferring these using HTTP headers. Such systems are typically technically integrated via filters running inside the web server (for example, ISAPI filters in IIS) or via Reverse Proxies running in front of the web server of the business application (for example, the QVWS or IIS that runs QlikView).

The limitation of this solution is that groups cannot be transferred to the QlikView system. They have to be resolved by the Directory Service Connector (DSC) or not at all.

If the ID is in the form "DOMAIN\UserID", the QVWS or IIS asks the DSC to find the user's groups. The DSC asks the relevant Directory Service Provider (DSP) identified using the domain to find the user's groups. If the user ID is not in the form specified above, or if there is no configured DSP for the given domain, no groups are associated with the user's web session.

## Architecture

The figure below provides an overview of the header solution.



## Security

The user information sent between the portal or web frontend and the QVWS or IIS is transferred using HTTP headers. If SSL with certificates is used, all traffic is encrypted and the user identity protected.

If IP restrictions are used to build the trust, the network between the portal or web frontend and QVWS or IIS has to be protected using physical security controls and network configuration to avoid eavesdropping.

For this solution to become secure, it is vital that the user cannot communicate directly with the QVWS or IIS, but has to access the portal or web frontend in order to access QlikView. This is typically achieved by configuring a firewall around the QVWS or IIS so that it only accepts web requests coming from the authentication system.

## Trust

The trust between the portal or web frontend and the QVWS or IIS can be built using either IP address restrictions for the QVWS or SSL/TLS using certificate authentication, basic authentication, and so on when using IIS.

Trust between servers in Trust Zone 2 is established using QlikView session tickets.

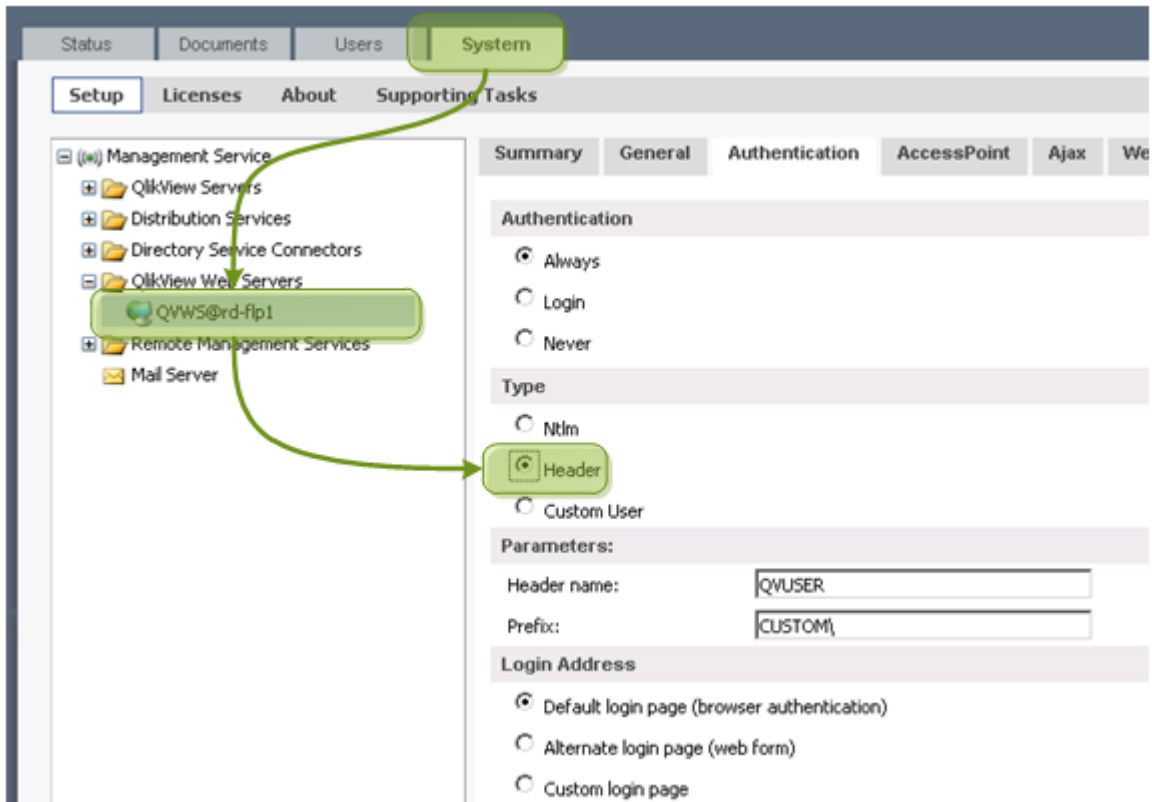
### IP restrictions

For the header solution to be secure, IP address access controls have to be put in place so that only the proxy/portal can access the QlikView system. Under no circumstances may the end user browser communicate directly with the QlikView web server.

## Examples

This system cannot be used with anonymous logins, since the authentication system does not allow it. Therefore set the **System>QlikView Web Servers>Authentication** radio button to **Always** in the QlikView Management Console (QMC).

In addition, the QVWS or IIS has to be configured to accept authentication using headers (see the figure below).



## Apache integration using mod\_proxy and mod\_rewrite

The following example shows how to use the Apache open source web server (<http://httpd.apache.org>) as an authentication system that is configured as a Reverse Proxy in front of the QVWS. The authentication procedure is as follows:

1. The user credentials are transferred to Apache via the basic authentication system. This means that the user is prompted for username and password via a pop-up dialog box.
2. The credentials are checked against a text file on disk that is read by the Apache web server.
3. If the credentials correctly identify a user in the text file, the web request is handled by adding the username to an HTTP header and sending it to the QVWS.

The Apache web server is configured via a text file called `httpd.conf`. The configuration below has to be accompanied by a configuration to create trust between the Apache server and the QVWS. This can, for example, be IP address filters.

```
ProxyRequests Off
SSLProxyEngine on

<Proxy *>
    Order allow,deny
    Allow from all
</Proxy>

RewriteEngine On

#RewriteLog /var/log/httpd/rewrite.log
#RewriteLogLevel 9
RewriteCond %{LA-U:REMOTE_USER} (.+)
RewriteRule . - [E=RU:%1]
RequestHeader set QVUSER %{RU}e

<Location />
    AuthType Basic
    AuthName "Qlikview"
    AuthBasicProvider file
    AuthUserFile /etc/httpd/conf/passwords
    Require valid-user
</Location>

ProxyPass /qlikview https://[server name]/qlikview
ProxyPassReverse /qlikview https://[server name]/qlikview
ProxyPass /QvAJAXZfc https://[server name]/QvAJAXZfc
ProxyPassReverse /QvAJAXZfc https://[server name]/QvAJAXZfc
ProxyPass /QvAjaxZfc https://[server name]/QvAJAXZfc
ProxyPassReverse /QvAjaxZfc https://[server name]/QvAJAXZfc
```

## WebTicket

In this solution, the user is authenticated towards a trusted third-party that integrates with QlikView using a predefined API called WebTicket. The use of a WebTicket enables the third-party system to transfer the user to the QlikView platform in a secure way. The authenticity of the authenticating server is ensured by the use of IP address filters or authentication.

### When to use

WebTicket is best used when the authentication system is seen by the browser as being a totally different web site to the one that runs QlikView and there is no possibility to modify the QlikView web site with a filter solution.



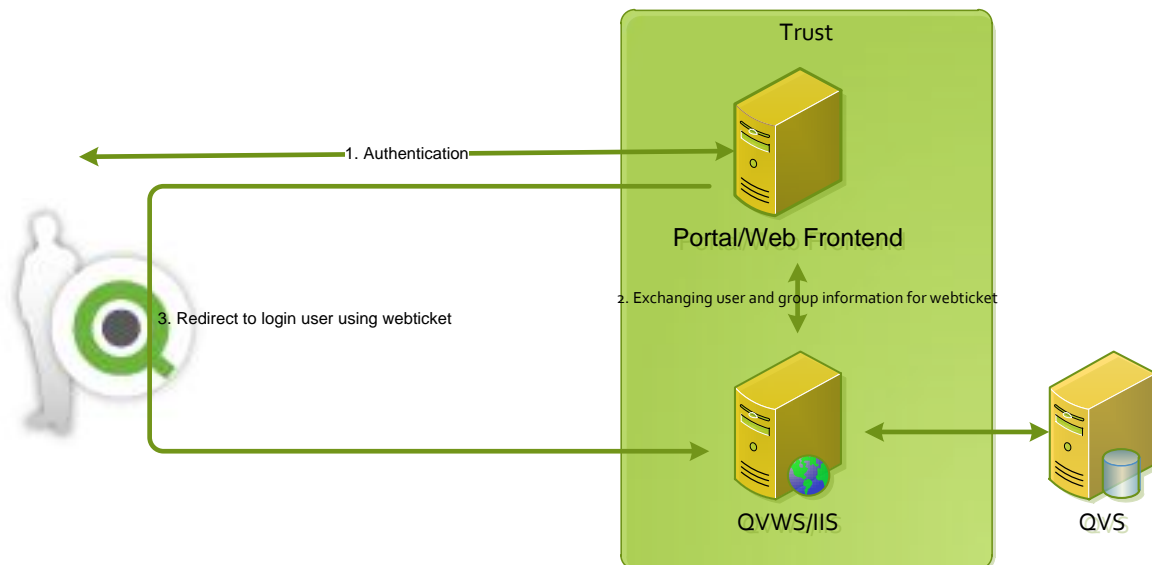
However, WebTicket is easy to configure and can be used even when there is an option to re-write the `Authenticate.aspx` page (see below). This means WebTicket is a good solution when:

- There is a need to integrate with a single sign-on system that does not use HTTP headers.
- Using IFrame integrations into portal systems.
- The authentication system knows the user's groups and this information should be transferred to QlikView for authorization decisions.

The WebTicket solution supports document reconnect and document chaining.

## Architecture

The figure below provides an overview of the WebTicket solution.



The procedure for logging in to the application is as follows:

1. [Optional:] The user accesses the QlikView server. If the user is not authenticated, the user is redirected to the authentication server.
2. The user authenticates to the authentication server using the configured method. The authentication server sends information, containing the username and the groups that the user belongs to, to the QVWS or IIS. In return, the portal gets a WebTicket that is valid for a limited period of time.
3. Upon successful authentication and WebTicket exchange, the authentication server redirects the user to the QVWS or IIS with the WebTicket.

4. QlikView validates the WebTicket and, if valid, the user information is transferred to the QVS.
5. The user is now logged in to the QlikView platform.

A current limitation is that the QVWS or IIS used in steps 2 and 3 (i.e. the server you collect the ticket from and the server you redirect the user to) must be the same machine.

## Security

The trust between the servers is configured using `config.xml` (which, in default installations, is found in `C:\ProgramData\QlikTech\WebServer`) to create the IP address trust. When other authentication alternatives (for example, basic or certificates) are needed, the IIS web server functionality is used.

Example of `config.xml` using trusted IP to trust 192.168.0.2 and 192.168.0.3:

```
<Authentication>
  <AuthenticationLevel>Always</AuthenticationLevel>
  <LoginAddress>/qlikview/login.htm</LoginAddress>
  <LogoutAddress>logout.htm</LogoutAddress>
  <GetTicket url="/QvAjaxZfc/GetTicket.aspx" />
  <GetWebTicket url="/QvAjaxZfc/GetWebTicket.aspx">
    <TrustedIP>192.168.0.2</TrustedIP>
    <TrustedIP>192.168.0.3</TrustedIP>
  </GetWebTicket>
  <HttpAuthentication url="https://_/scripts/GetTicket.asp" scheme="Basic" />
  <HttpAuthentication url="/QvAJAXZfc/Authenticate.aspx" scheme="Ntlm" />
</Authentication>
```

## WebTicket protocol

The request for a WebTicket in step 2 is done to the `/QvAJAXZfc/GetWebTicket.aspx` page with a POST containing XML in the following format:

```
<Global method="GetWebTicket">
  <UserId>[User Name]</UserId>
  [Groups]
</Global>
```

where `[User Name]` is the name of the user, and `[Groups]` are added in the following format:

```
<GroupList>
  <string>
    [Group Name 1]
  </string>
  <string>
    [Group Name 1]
  </string>
</GroupList>
<GroupsIsNames>
  true
</GroupsIsNames>
```

`<GroupIsNames>` should be true, if groups are supplied as names for use with Document Metadata Service (DMS) authorization. Set it to false, if they are supplied as Windows Security Identifiers (SID) that allow NTFS authorization.

The ticket is returned in the following format:

```
<Global><_retval_>[WebTicket]</_retval_></Global>
```

where `[WebTicket]` is the actual ticket that is represented by a string of random bytes.

The ticket should then be included in a redirect to:

```
http://[Server]/QvAJAXZfc/Authenticate.aspx?type=html&webticket=[WebTicket]&try=[Try URL]&back=[Back URL]
```

where `[Try URL]` refers to the URL that the user is redirected to after a successful login, `[Back URL]` is the URL redirected to after a failed login attempt, and `[Server]` is the server where the QlikView AccessPoint resides.

For an example of PHP code to use in order to add login using WebTickets, see Appendix A.

For an example of .NET code to use in order to add login using WebTicket, see Appendix B.

## Configuration

Web tickets work with both the QVWS and IIS, while not mandatory it can be easier to use IIS for the QlikView Web Server when working with ticketing. It may be required to host your login page anyway.

The accounts that is authenticated using web tickets are not Windows account and therefore only work with QlikView running in DMS mode.

If using web tickets with IIS, IIS needs to be set up to use Anonymous permissions – it will be expecting windows permissions by default – specifically it is the QVAJAXZFC directory that needs its permission changing.

## Custom-developed authentication through Authenticate.aspx

### When to use

The QlikView authentication API is designed to be used by the `Authenticate.aspx` web page. The web page can be customized using IIS to handle all three steps of sending, validating, and transferring a user's credentials to the QlikView web session.

This is typically necessary when there is an existing user repository (for example, LDAP or a database), but no existing web-based authentication server that can be used to integrate with it. In this case, the web-based authentication system has to be created and it may be simpler to create it as part of the QV web tier, as opposed to making a separate authentication system.

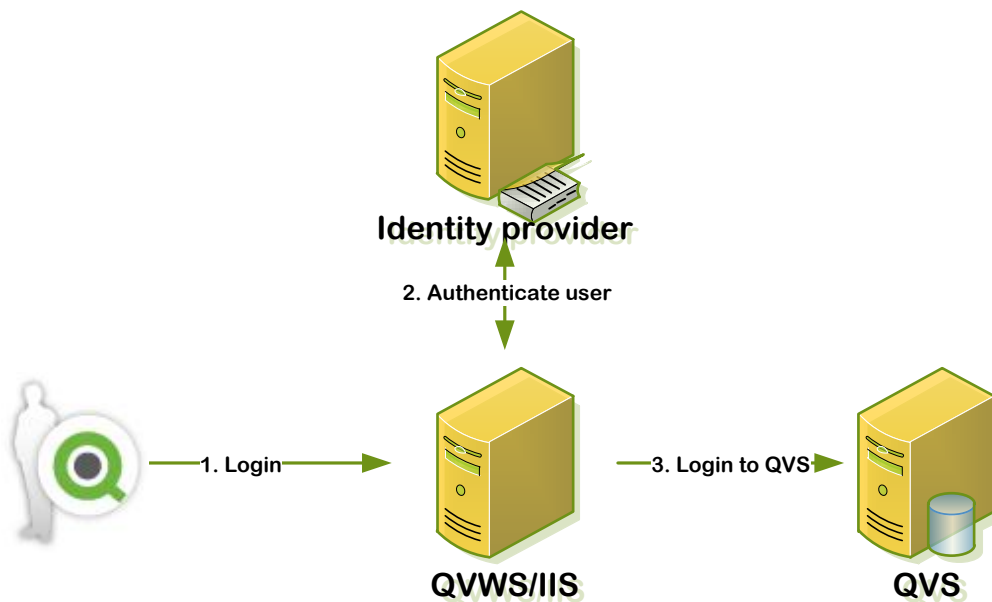
This solution is also suitable in integrations with cloud-based authentication systems, where it is not possible to configure the authentication system to call the QlikView platform in a custom way (that is, using headers or WebTicket). In this scenario, the QlikView web tier has to be adapted to the requirements of the cloud system. A good example of this is salesforce.com.

The development environment for `Authenticate.aspx` is supplied in .NET languages.

`Authenticate.aspx` supports the transferring of group information to the QlikView system.

## Architecture

The figure below provides an overview of the custom-developed authentication solution.



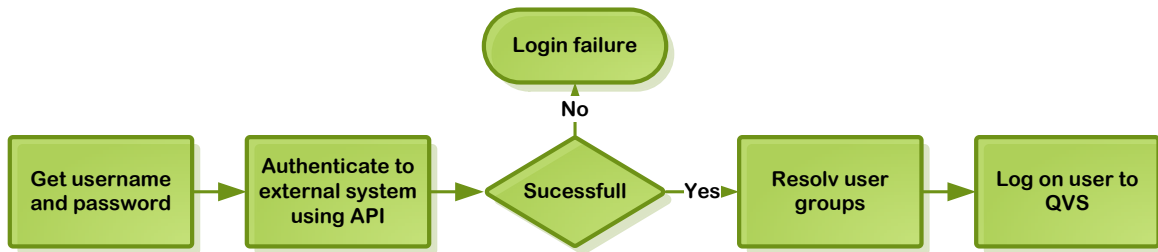
## Security

The procedure for logging in is as follows:

1. The user logs in to IIS using any authentication system. This typically means that the user ID and password are sent to IIS, but it may also mean that the user's fingerprint or a one-time token from a cloud authentication provider is sent.
2. The customized `Authenticate.aspx` checks the details towards an external security system (for example, an LDAP server, an SAML identity provider, or salesforce.com).
3. If successful, `Authenticate.aspx` transfers the user information (potentially including groups) to the QVS.

## Example

A typical `Authenticate.aspx` procedure includes the steps in the figure below.



For an example of `Authenticate.aspx` for LDAP, see Appendix C.

## How to choose solution

All three solutions support the principle of authenticating users outside QlikView and add flexibility to the QlikView product. Which solution to use depends on the architecture where the solution is going to be used:

- As a basic principle, the header solution makes for easier integration into portals, but needs development or adaption of reverse proxy functionality in the portal or web frontend.
- The header solution is preferred, if an SSO system with header support exists.
- The WebTicket solution is best in environments where a third-party authentication system can be integrated with and if group information has to be transferred.
- If integration of content is done using IFrames, WebTicket is a good solution.
- `Authenticate.aspx` is best suited when there is an external directory or identity provider to authenticate towards, but no web frontend to integrate with.

All solutions above are seen as good practice and should, if possible, be used instead of `GetTicket` (as used in versions prior to QlikView 11).

## Appendix A: PHP WebTicket code

In the example below, the authentication is handled by an external party and the username is supplied in the variable.

Text within [] contains explanations of the information that has to be supplied to the variables.

```
<?php
$user      = "[get user from authentication solution]";
$server    = "[QlikView server to redirect to]";
$groups    = "[groups supplied by authentication solution]";
$GroupIsSID = "[Yes if groups are SID's]";

$tempGroup = explode(";", $groups);
$groupList = "";

$xmlRequest = new DOMDocument();

//Create XML request
$root = $xmlRequest->appendChild($xmlRequest->createElement("Global"));
$root->appendChild($xmlRequest->createAttribute("method")->appendChild($xmlRequest->createTextNode("GetWebTicket")));
$root->appendChild($xmlRequest->createElement("UserId", $user));

//Append group information to XML request
if ($groups != "") {
    $groupTag = $root->appendChild($xmlRequest->createElement("GroupList"));

    for ($i = 0; $i < sizeof($tempGroup); $i++) {
        $groupTag->appendChild($xmlRequest->createElement("string", "$tempGroup[$i]"));
    }

    //Defines if groups are SID:s or names
    if ($GroupIsSID == 'Yes') {
        $root->appendChild($xmlRequest->createElement("GroupIsNames", "false"));
    } else {
        $root->appendChild($xmlRequest->createElement("GroupIsNames", "true"));
    }
}
```

```
$txtRequest = $xmlRequest->saveXML();

define('xmldata', $xmlRequest->saveXML());
define('URL', 'http://' . $server . ':80/QvAJAXZfc/GetWebTicket.aspx');
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, URL);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_TIMEOUT, 4);
curl_setopt($ch, CURLOPT_POSTFIELDS, xmldata);

//Request ticket
$result = curl_exec($ch);

$info = curl_getinfo($ch);

$xmlResponse = new DOMDocument();
$xmlResponse->loadXML($result);

//Get ticket
$ticket = $xmlResponse->getElementsByTagName("_retval_")->item(0);

//Create redirect URL with ticket
$redirectURL = "http://$server/QvAJAXZfc/Authenticate.aspx?type=html&webticket=$ticket->nodeValue&try=[Try URL]&back=[Back URL]";

header('Location: ' . $redirectURL);
?>
```

## Appendix B: .NET Web Ticket code

**Note:** The full solution is supplied in the accompanying zip file.

```
<%@ Page Language="C#" AutoEventWireup="true" %>
<%@ Import Namespace="System.Net" %>
<%@ Import Namespace="System.IO" %>

<html>
<head>
</head>
<body>

<form runat="Server" id="MainForm">
  <H2>QlikView WebTicket Example</H2>
  Username: <asp:TextBox runat="server" id="txtUser">MyTestUser</asp:TextBox> <br />
  Password: <asp:TextBox runat="server" id="txtPassword">MyPassword</asp:TextBox> <br />
  Groups: <asp:TextBox runat="server" id="txtGroups">Sales,Finance</asp:TextBox> <br />
  <asp:Button ID="GO" runat="server" Text="Login..." onclick="GO_Button_Click" />
</form>

</body>
</html>
<script language="c#" runat="server">

  //Firsly Set up a few properties
  static string QlikViewServerURL = @"http://localhost/QVAJAXZFC/getwebticket.aspx"; //address of the QlikView server
  static string ticketinguser = "qservice"; //Service account used to ask for a ticket (QV Administrator), this is not the end user
  static string ticketingpassword = "Q1ktech";
  static string document = ""; // specify a single QVW or leave blank for access point

  protected void GO_Button_Click(object sender, EventArgs e)
  {
    //Grab the details the user provides
    string username = txtUser.Text;
    string password = txtPassword.Text;
    string groups = txtGroups.Text;

    // Test if the user is valid
    bool loginOK = ValidateUser(username, password);

    //If the user is valid get a ticket and log them in
    string ticket = "";
    if (loginOK)
    {
      //Get the Ticket
      ticket = getTicket(username, groups, ticketinguser, ticketingpassword); // add groups into the empty string if required
    }
  }
}
</script>
</script>
</script>
```



# QlikView

```
//Build a redirect link to either access point or to a single document
string RedirectLink = "";

if (document.Length > 0)
{ //Send to a single document
    RedirectLink = "/qvajaxzfc/authenticate.aspx?type=html&try=/qvajaxzfc/opendoc.htm?document=" + document +
"&back=/LoginPage.htm&webticket=" + ticket;
}
else
{ //Send to a Access Point
    RedirectLink = "/qvajaxzfc/authenticate.aspx?type=html&try=/qlikview&back=/LoginPage.htm&webticket=" + ticket;
}
//Redirect the user
Response.Redirect(RedirectLink);
}

}

// This function is a place holder for code to test the user id and password against a database or LDAP
// For the purpose of this example we are not going to test it at all, just return "true" for logged in
// Of course this means anyone would be let in, so this needs to be added
private bool ValidateUser(string User, string pass)
{
    //here you would have some logic to test the user id and password against a data base.
    return true;
}

// This function is going to take the username and groups and return a web ticket from QV
// User and groups relate to the user you want to request a ticket for
// ticketinguser and password are the credentials used to ask for the ticket and needs to be a QV admin
private string getTicket(string user, string usergroups, string ticketinguser, string ticketingpassword)
{
    StringBuilder groups = new StringBuilder();
    if (!string.IsNullOrEmpty(usergroups))
    {
        groups.Append("<GroupList>");
        foreach (string group in usergroups.Split(new char[] { ',' }))
        {
            groups.Append("<string>");
            groups.Append(group);
            groups.Append("</string>");
        }
        groups.Append("</GroupList>");
        groups.Append("<GroupsIsNames>");
        groups.Append("true");
    }
}
```

```

        groups.Append("</GroupsIsNames>");
    }
    string webTicketXml = string.Format("<Global method=\"GetWebTicket\"><UserId>{0}</UserId></Global>", user);

    HttpWebRequest client = (HttpWebRequest)WebRequest.Create(new Uri(QlikViewServerURL));
    client.PreAuthenticate = true;
    client.Method = "POST";
    client.AutomaticDecompression = DecompressionMethods.GZip | DecompressionMethods.Deflate;

    client.Credentials = new NetworkCredential(ticketinguser,ticketingpassword);

    using (StreamWriter sw = new StreamWriter(client.GetRequestStream()))
        sw.WriteLine(webTicketXml);
    StreamReader sr = new StreamReader(client.GetResponse().GetResponseStream());
    string result = sr.ReadToEnd();

    XDocument doc = XDocument.Parse(result);
    return doc.Root.Element("_retval_").Value;
}
</script>

```

## Appendix C: Authenticate.aspx for LDAP

**Note:** The full solution is supplied in the accompanying zip file.

```

<%@ Import Namespace="System" %>
<%@ Import Namespace="System.Linq" %>
<%@ Import Namespace="QlikView.AccessPoint" %>
<%@ Import Namespace="QlikView.AccessPoint.HttpInterfaces" %>
<%@ Import Namespace="QvIISWebServer" %>
<%@ Import Namespace="System.Security.Principal" %>
<%@ Import Namespace="System.DirectoryServices" %>
<%@ Page Language="C#" Debug="true" %>
<%@ Import Namespace="System.Collections.Generic" %>

<script runat="server">
List<string> groups=new List<string>();
protected void Page_Load(object sender, EventArgs e)
{
    Response.Clear();
    Context context = this.GetContext();
    context.Response.AddVersionHeader();
    //Saving the Basic authentication header to a string ("Basic UserName:Password")
    string auth = Request.Headers["Authorization"];

```

```

if (auth != null && auth.IndexOf("Basic ") == 0)
{
    //Removing "Basic " from the string and decoding the username and password
    string UnameColonPwd =
        System.Text.Encoding.UTF8.GetString(
            System.Convert.FromBase64String(auth.Substring(6)));
    int colonPos = UnameColonPwd.IndexOf(':');
    if (colonPos >= 0)
    {
        //Splitting username:password into two separate strings
        string userid = UnameColonPwd.Substring(0, colonPos);
        string pwd = UnameColonPwd.Substring(colonPos+1);
        if (authUserAndGetGroups(userid, pwd)==true)
        {
            //Creating an user object with groups
            IUser user;
            user = new NamedUser(userid, groups, true);
            //Associate the IUser with the session cookie and either redirect or sent back a status code
            QlikView.AccessPoint.User.GenericAuthentication(context, user);
            //Stop now so we don't issue a new BASIC challenge
            return;
        }
        else
        {
            //If the username/password is incorrect the user is asked to try again
            Context.Response.StatusCode=401;
            Context.Response.AppendHeader("WWW-Authenticate","Basic realm=\"QlikView\"");
            return;
        }
    }
}
else
{
    Context.Response.StatusCode=401;
    Context.Response.AppendHeader("WWW-Authenticate","Basic realm=\"QlikView\"");
}
}

private bool authUserAndGetGroups(string userid, string password)
{
    try
    {
        //Connectionstring to the LDAP, with username and password from the login (OpenLDAP syntax)
        DirectoryEntry nRoot = new DirectoryEntry("LDAP://127.0.0.1:389/dc=jbitest,dc=com");
        nRoot.Username = "uid="+userid+",ou=People,dc=jbitest,dc=com";
        nRoot.Password = password;
        nRoot.AuthenticationType = AuthenticationTypes.None;
    }
}

```

```
//Verifying the user
Object temp = nRoot.NativeObject;

//Creating a LDAP search object
DirectorySearcher nDS = new DirectorySearcher(nRoot);
//Filter the search to only find groups connected to the username(full dn) from the login
(OpenLDAP syntax)
nDS.Filter = "(&(objectClass=groupOfNames)(member="+nRoot.Username+")";
nDS.PageSize = 1000;

//Performing the search and catching all of the results matching the filter set in nDS.Filter
SearchResultCollection results = nDS.FindAll();
foreach (SearchResult searchResult in results)
{
    foreach (string propertyKey in searchResult.Properties.PropertyNames)
    {
        //The property key containing the group name on the LDAP server will most likely be named "cn" (OpenLDAP)
        if(propertyKey=="cn")
        {
            ResultPropertyValueCollection valueCollection = searchResult.Properties[propertyKey];
            foreach (Object propertyValue in valueCollection)
            {
                //Adding all groups connected to the userid to a list we are going to send with the username to the "AccessPoint"
                groups.Add(propertyValue.ToString());
            }
        }
    }
}
//If no exceptions occurred the authentication has been successful and possibly even resolved the groups
return true;
}
catch (Exception e)
{
    //If exceptions occurred the authentication has most likely been unsuccessful (or the syntax used might be wrong)
    return false;
}
}
</script>
```

## Appendix D: Detailed description of the Authentication API

The API for the `Authenticate.aspx` page works as follows:

- If no parameters are present, a login is attempted. 200 mean success, 401 means failure.
- If the `keep=` parameter is present, re-authentication (that is, ensuring that there already is a user associated with the session cookie) is attempted. In some scenarios, the fact that the web call is made is enough to force the browser to prove who the user is again. For example, when basic authentication or NTLM is used to protect the page, the browser is asked for the login again, but the browser does not ask the user. However, in other scenarios, it may be difficult to work out who the user is for a second time around and therefore a previous authentication has to be considered a success. 200 mean success, 401 means failure.
- If the `type=html` parameter is present, a redirect to another web page after success or failure of the above functionality is attempted. In this case, it is optional to provide the `Try` and `Back` parameters containing URLs to redirect to in case of success or failure. If no `Back` parameter is provided, `/qlikview/index.htm` is used (that is, the default QlikView AccessPoint web page). If no `Try` parameter is provided, the `Back` parameter is used for both success and failure. This is useful with AccessPoint when using the form login. The form login calls the `Authenticate.aspx` page and uses the parameters to indicate where to redirect to if the login is successful or not. This can also be useful for an initial login where subsequent re-authentication cannot be performed.

### How to use the API

The API is a set of .NET classes and methods designed to be used in code deployed in IIS. The correct versions of ASP.NET and .NET have to be installed in IIS. For QlikView 11, the correct version is .NET Framework 4. The relevant executable is `QVWebServer.exe`, which references the assemblies in `c:\Program Files\QlikView\Server\WebServer`. Make sure to have these in the web application.

Then identify the security requirements for the project and how the workflow for HTTP calls works for login of users, re-authentication of users based on session cookies, and finally logout of users.

### Representing users

The `QlikView.AccessPoint.IUser` in `Accesspoint.dll` .NET interface represents any sort of user and the user's groups. There are two classes that implement this:

- `QlikView.AccessPoint.AnonymousUser` represents an anonymous user.
- `QlikView.AccessPoint.NamedUser` represents a named user and the user's groups.

In practice, the code never creates an `AnonymousUser` object. It only creates `NamedUser` objects, since the API handles all `AnonymousUser` scenarios.

## Representing web requests and configuration

The API requires access to the web request, so that it can read parameters and create or modify cookies. Some methods also require access to the QlikView web tier configuration (for example, in order to know the HTTP header value to use).

The basic object representing the web request and configuration is called `Context`. It is represented by the .NET class `QlikView.AccessPoint.Context`. To get a `Context` object on an ASPX web page, use the following code:

<pre>&lt;%@ Import Namespace="QlikView.AccessPoint" %&gt;  &lt;%@ Page Language="C#" AutoEventWireup="true" %&gt;  &lt;script runat="server"&gt;      protected void Page_Load(object sender, EventArgs e) {          Context context = this.GetContext();          context.Response.AddVersionHeader();      } &lt;/script&gt;</pre>	<p>Declare that the <code>QlikView.AccessPoint</code> namespace is used, so that there is access to the classes defined in the API.</p> <p>The page is written in C#.</p> <p>The C# code is to run on the server.</p> <p>Define a method/function that runs when the page is loaded.</p> <p>Get the <code>Context</code> object.</p> <p>Add the current QlikView version as a special response header.</p>
---	--

## Creating or obtaining an `IUser` object

The object representing a user and the user's groups can be created in three ways as described below.

### Creating a generic `IUser` object

This is the most generic and flexible approach. Use your own code that tells you who the user is. This means that the username and the user's groups are known. All that is needed is to create an object to represent the user:

```
IUser user = new NamedUser("userX", null, true);
```

#### Additional example:

```
List<string> groups = new List<string>();
groups.Add("EXPORTXLS"); //Allow him to export to Excel for this session
groups.Add("FRANCE"); //He can see France data
IUser user = new NamedUser("NTDOMAIN\XXX", groups, true);
```

The generic form of this function/constructor is:

```
NamedUser(string name, IEnumerable<string> groups, bool groupsAreNames)
```

- The username can be anything, but historically the Windows format of DOMAIN\User has been used. Keep in mind that the username is capitalized when sent to the QVS (for example, “xxx” and “XXX” are considered to be the same).
- To get “NTDOMAIN\XXX” in .NET, write “NTDOMAIN\\XXX” because of how .NET handles a “\” in a string. “\” is used for special characters (for example, “\n”), which means “start a new line”. So “\\” really means just “\”.
- The groups are either Windows groups represented by SIDs or textually named groups. Use SIDs, if the QVS is in NTFS mode, and names, if it is in DMS mode. By their nature, SIDs are more difficult to list, since they are just numbers. Never write them yourself (see the second example above).

## Representing a Windows account

If using the IIS authentication system, the Windows account is already available to the code. The simplest way of integrating with this is to use a helper method in the API and not create an `IUser` object explicitly at all. For more information, see further down.

However, if the authentication system is more complicated and makes the Windows account available to the .NET code, see the code snippet below (note that this should only be an issue for OEMs that use the ASP.NET platform for their web applications):

```
if (context.Principal != null && context.Principal.Identity.IsAuthenticated)
{
    //Your web infrastructure has an authenticated Pricipal (user)
    WindowsIdentity wi = context.Principal.Identity as WindowsIdentity;
    if (wi != null)
    {
        //It is using a Windows account. We convert the SIDs to text
        List<string> groups = wi.Groups.Select(i =>
            i.Translate(typeof(SecurityIdentifier)).Value).ToList();
        //And make the object
        return new NamedUser(wi.Name, groups, false);
    }
    return null;
}
```

However, if there is a web form or some other system that provides the username and password of a Windows account, the following API call can be useful:

```
IUser QlikView.AccessPoint.User.TryLogon(string Name, string Password);
```

**Example:**

```
IUser user = QlikView.AccessPoint.User.TryLogon("NTDOMAIN\\XXX", "m1b2e3");
```

- The `Name` should be in the form `DOMAIN\User`.
- In case of failure, null is returned.
- The groups are represented by Windows SIDs.
- Note that the Windows API uses an interactive logon rather than a batch logon.

## Creating an `IUser` based on a name and getting groups from the DSC

If the username is retrieved from some system, use the following API call to create an object representing this:

```
IUser QlikView.AccessPoint.User.ResolveUser(string name, Settings settings);
```

The `Settings` object allows the API to find a DSC to ask for the groups. This can be retrieved from the `Context` object found earlier:

```
IUser user = QlikView.AccessPoint.User.ResolveUser("NTDOMAIN\\XXX", context.Settings);
```

The way that the DSC works is that it loads a set of DSPs and then asks each DSP to load a particular directory resource based on a configuration that an administrator has set up in the QMC. Each directory resource can load multiple “domains” (for example, `NTDOMAIN` or `QTIT`). Each domain has a simple name that uniquely identifies it in the DSC. When trying to get the list of groups for a user from the DSC, the DSC finds the domain to ask by looking at the username. For example, if asking for the groups of `NTDOMAIN\XXX`, the DSC asks the `NTDOMAIN` domain for the groups in that domain. If asking for the groups of `XXX`, the DSC returns zero (0) groups, since there is no domain. So, the username must be in the Windows format of `DOMAIN\User` for the DSC group resolution to work.

To test group resolution through the DSC, use the following URL:

```
http://localhost:4730/qtlds.asmx?ResolveGroups&_user=Domain \User
```

where `Domain` is the prefix of the DSC (for example, “CUSTOM”).

The important limitations of this system are:

- The DSP for the directory has to be configured. If using a directory that does not have a DSP already developed, a DSP has to be written. That is far more complicated than using the API to pass groups.
- The groups for a user can only be found in the same domain as the user. However, in practice, the account `NTDOMAIN\XXX` is member of groups in other domains. This has a big impact on projects in large organizations, since it requires getting the IT teams that handle the Active Directory or LDAP servers involved and that may take a long time.
- It requires the web server to make a web call to the DSC as part of the authentication. There are a number of issues around which service gets placed in which network zone.



The DSC is used by QlikView Publisher in the batch processing as well, so this introduces confusion when making the choice.

## Logging in the IUser and performing re-authentication

There is an API call used to associate the `IUser` object with the session cookie. The old `IUser` associated with the cookie is dis-associated. The function returns a string that is the session ID stored in the session cookie:

```
string QlikView.AccessPoint.User.Login(Context context, IUser user)
```

The helper method that sends the relevant response (redirect or error code) is as follows:

```
void QlikView.AccessPoint.User.SendLoginResult(Context context, bool success)
```

The helper method that handles almost everything (that is, associate the `IUser` object with the session cookie or just ensure that the user is already authenticated as well as sending the right response) is as follows:

```
void QlikView.AccessPoint.User.GenericAuthentication(Context context, IUser user)
```

- If the `keep=` parameter is present in the URL, it only checks if some user is logged in.
- If the `IUser` object is null, `SendLoginResult()` is used to issue a failure web page.

The main problem with using `GenericAuthentication()` is that, in some authentication scenarios, the user credentials are only available once at the beginning. For re-authentication, a check that the user already exists is the only thing needed, but `GenericAuthentication()` requires us to work out who the user is for a second time around and actually issues a failure if it is not correct.

Because of this, there is another API method that allows finding out who is actually associated with the current session cookie:

```
void QlikView.AccessPoint.User.GetUserAndId(Context context, out IUser user, out string id)
```

- Returns both the `IUser` object and the session cookie ID associated with the current request.
- If no such cookie exists, it logs in the anonymous user and returns it.
- The `Context` object represents the current HTTP request and the configuration context.

## Logging in once only

So, there are essentially two main scenarios to consider. When coming in from an external authentication system (for example, salesforce.com), we would log in just once. The functionality for logging in has to be separated from the re-authentication functionality.

```
<%@ Import Namespace="QlikView.AccessPoint" %>
<%@ Page Language="C#" AutoEventWireup="true" %>
```

```
<script runat="server">
    protected void Page_Load(object sender, EventArgs e) {
        Context context = this.GetContext();
        context.Response.AddVersionHeader();
        IUser u;
        if (context.Request.QueryString["keep"] != null) { //Re-authenticate
            string id;
            QlikView.AccessPoint.User.GetUserAndId(context, u, id);
            QlikView.AccessPoint.User.SendLoginResult(u is
QlikView.AccessPoint.AnonymousUser);
        } else { //Do authentication
            u = new NamedUser("NTDOMAIN\\XXX", null, true); //E.g.
            QlikView.AccessPoint.User.GenericAuthentication(u, context);
        }
    }
}
</script>
```

## Logging in and performing re-authentication each time

In this scenario, the user credentials can be determined for each request, since they come from something inherent in the request (for example, basic authentication, NTLM, or SSL certificates). In this case, simply construct the object representing the user and call

`GenericAuthentication()`:

```
<%@ Import Namespace="QlikView.AccessPoint" %>
<%@ Page Language="C#" AutoEventWireup="true" %>
<script runat="server">
    protected void Page_Load(object sender, EventArgs e) {
        Context context = this.GetContext();
        context.Response.AddVersionHeader();
        IUser u = new NamedUser("NTDOMAIN\\XXX", null, true); //E.g.
        QlikView.AccessPoint.User.GenericAuthentication(u, context);
    }
}
</script>
```

For this scenario type, there are a number of helper API calls:

```
void HttpAuthentication(Context context)
```

If the IIS authentication mechanisms (Windows Integrated or basic) are used, the Windows identity is taken and then a call is made to `GenericAuthentication()`

```
void HeaderAuthentication(Context context, string headerName, string prefix)
```

If using a filter in front of the web tier that enforces users being logged in, the filter typically passes the username in an HTTP header (for example, this is the case with SiteMinder and CAS). This function finds that username, gets the groups from the DSC, and uses

`GenericAuthentication()` to do the rest:

- The `headerName` is the name of the header to use.

The prefix is a string that gets pre-pended to the username. For example, if the filter passes a username of "XXX" and the prefix is "NTDOMAIN\", the name that is used to log the user is "NTDOMAINXXX". This is useful when looking for groups via the DSC, which requires the "NTDOMAIN\" prefix in order to know which directory resource to use for group resolution.

