



# QlikView Security

Customized authentication

March 2012

Version 1.0

Author: Fredrik Lautrup, Michael Bienstein

## Table of Contents

Introduction	4
Overview of authentication	4
Header solution	5
When to use	5
Architecture	5
Security	5
Trust	6
Examples	7
WebTicket	9
When to use	9
Architecture	9
Security	10
WebTicket protocol	10
Custom-developed authentication through Authenticate.aspx	12
When to use	12
Architecture	12
Security	12
Example	13
How to choose solution	14
Appendix A: PHP WebTicket code	15
Appendix B: .NET WebTicket code	17
Appendix C: Authenticate.aspx for LDAP	19
Appendix D: Detailed description of Authentication API	22
How to use the API	22
Representing users	22
Representing web requests and configuration	23
Creating or obtaining an IUser object	23
Creating a generic IUser object	23
Representing a Windows account	24
Creating an IUser based on a name and getting groups from the DSC	25
Logging in the IUser and performing re-authentication	26
Logging in once only	26

Logging in and performing re-authentication each time

27

## Introduction

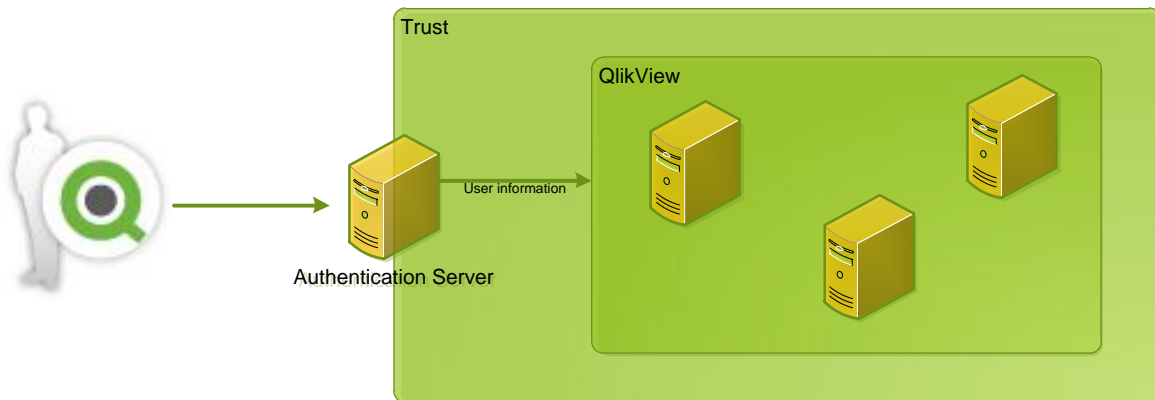
This document describes good practices regarding the implementation architecture for authentication solutions using the Ajax client for version 11. Three different approaches, including the pros and cons of each, to the authentication architecture are described.

What is included in this document are examples of implementations, each implementation is unique and need to be customized which makes it important that each solution is thoroughly tested before put 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 server.
2. The server verifies the credentials.
3. The user information is transferred to the QlikView platform.

The first two steps are handled by the external authentication server when authentication integration has been implemented. This document describes three methods for transferring user information to QlikView and when and how the different methods can be used.

## 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:

- IP address restrictions, if QVWS is used
- Additional authentication methods (for example, SSL using certificate authentication, basic authentication, or digest), if IIS is used

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

### When to use

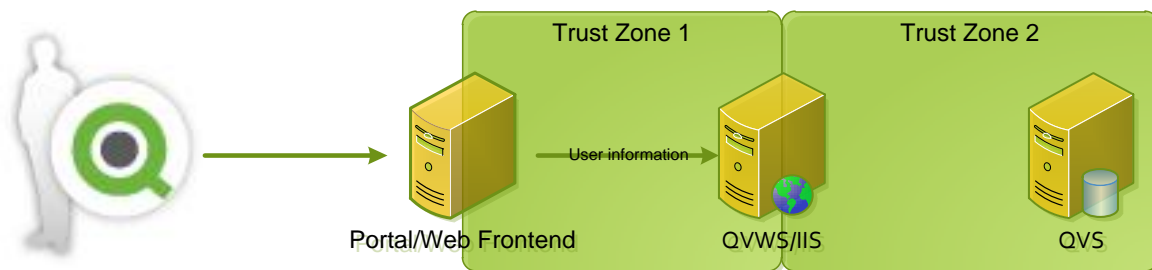
The header solution requires that reverse proxy functionality can be integrated into the portal or web frontend. Apart from basic functionality, the reverse proxy should be able to connect to the QVWS or IIS, adding the user information to the HTTP/HTTPS header.

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.

In most cases, this also adds the need to develop custom code in the portal or web frontend.

### 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.

## Trust

The trust between the portal or web frontend and the QVWS or IIS server 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. Examples of how the trust is built are shown below.

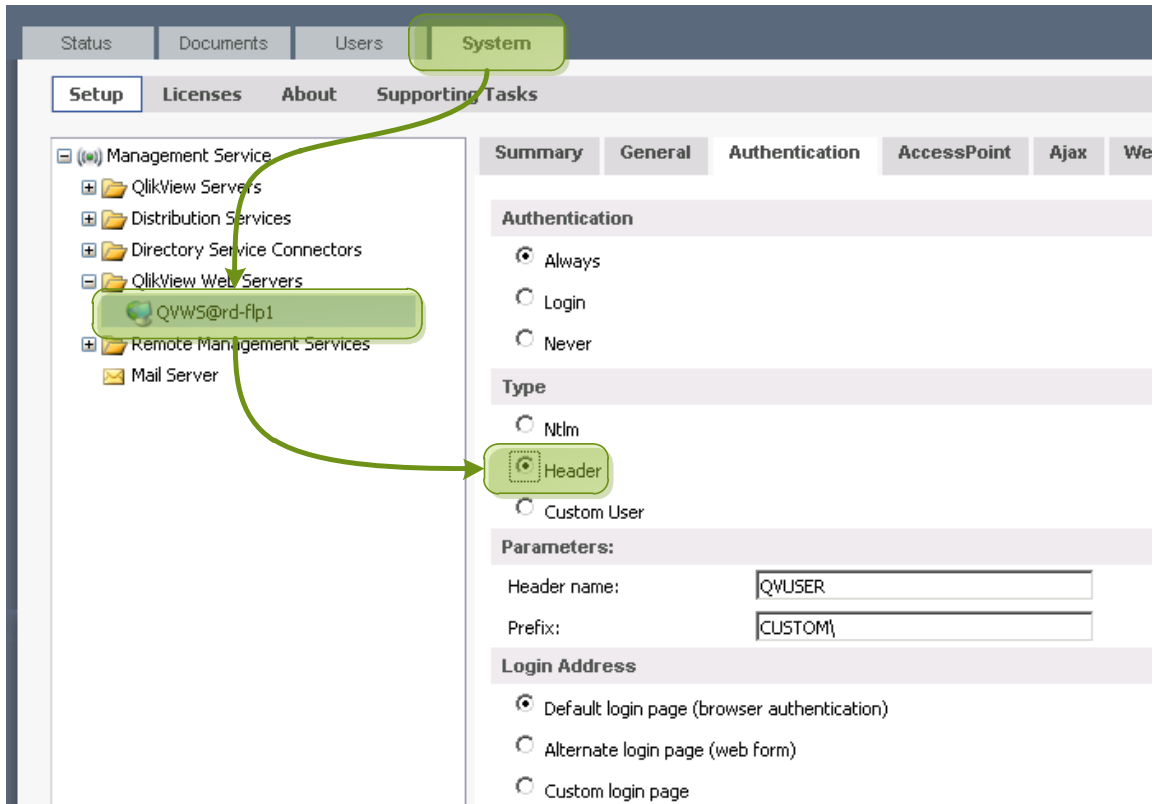
Trust between servers in Trust Zone 2 is established using QlikView session tickets.

## **IP restrictions**

For the header solution to be secure IP address access controls need to be put in place such that only the proxy/portal can access the QlikView system. Under no circumstances may the end user browser be able to communicate directly with the QlikView webserver.

## Examples

For the solution to work, the QVWS or IIS has to be configured to accept authentication using headers. This is configured in the QlikView Management Console (QMC), under **System>QlikView Web Servers>Authentication**.



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

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

This solution is based on that 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

Use the WebTicket solution when there is a need to integrate with a single sign-on system.

**WebTicket is also a good solution to solve authentication of IFrame integrations into portal systems.**

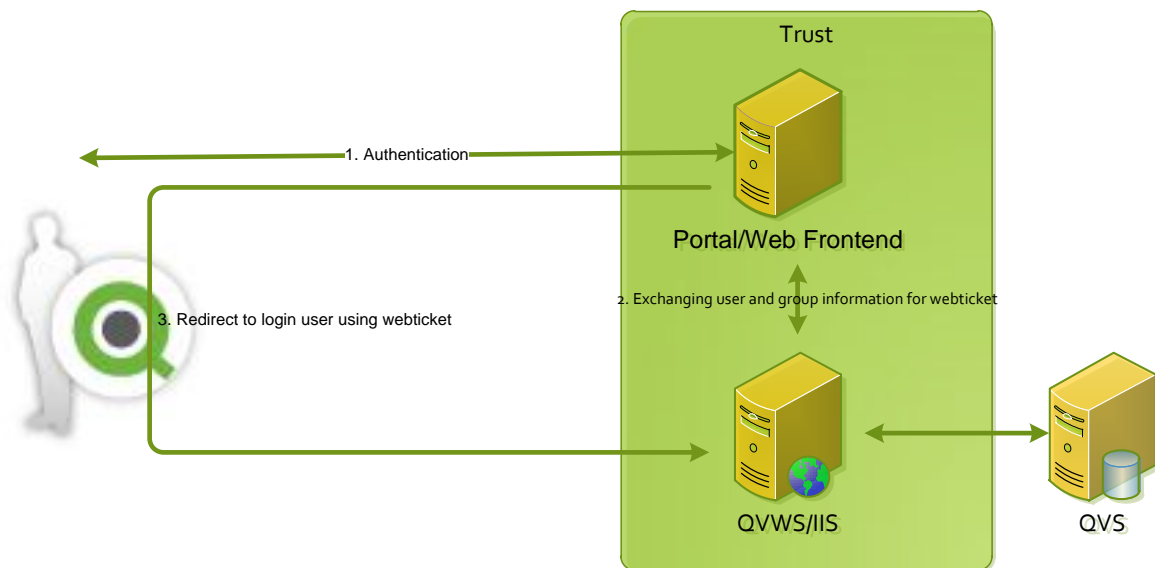
The WebTicket supports the transfer of group information to the QlikView system from the portal or web frontend.

The WebTicket solution also supports document reconnect and document chaining.

Note that the WebTicket is only valid on the webserver it was collected.

### Architecture

The figure below provides an overview of the WebTicket solution.



## Security

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

1. 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 back to the QVWS or IIS with the WebTicket.
4. QlikView validates the ticket and, if valid, the user information is transferred to the QVS.
5. The user is now logged in to the QlikView platform.

The trust between the servers is configured using `config.xml` (which, in default installations, is found under `C:\ProgramData\QlikTech\WebServer`) to create the IP address trust. When other authentication alternatives (for example, basic, digest, 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 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, and false, if they are supplied as Windows Security Identifiers (SID).

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 access point 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.

## Custom-developed authentication through Authenticate.aspx

### When to use

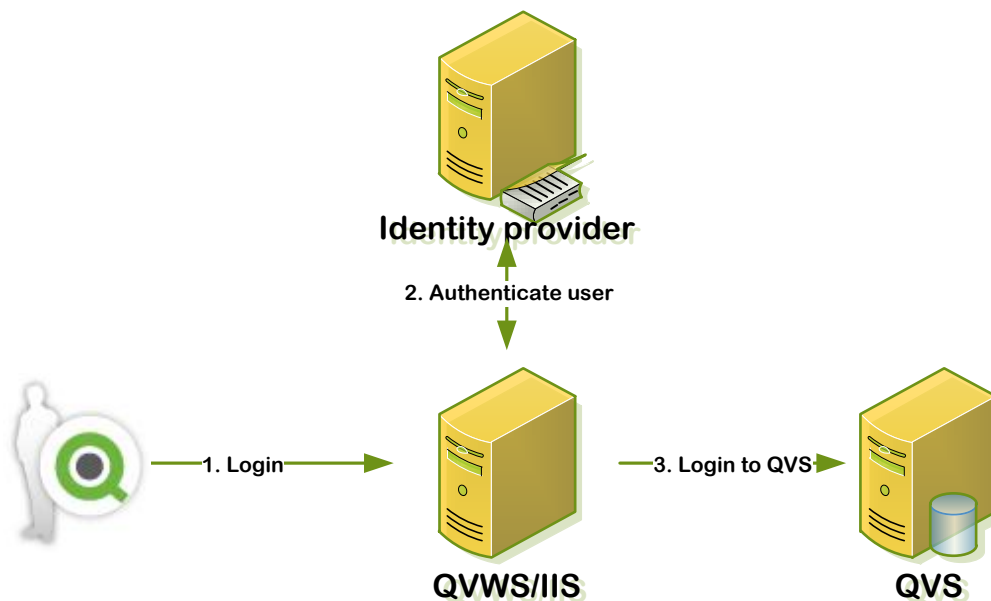
Use custom-developed authentication when there is a user repository, but no web-based authentication server that can be used to integrate with an identity provider (for example, a database or LDAP directory). In this case, QlikView has to perform the authentication towards the identity provider and then transfer the user information to the QVS.

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

Groups can be transferred to the QlikView system through customization.

### Architecture

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



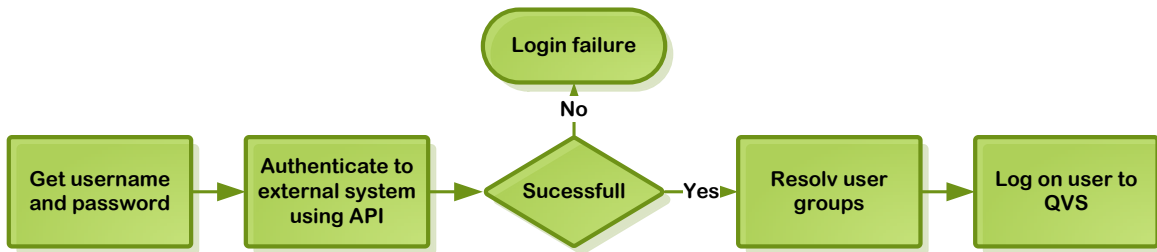
### Security

The procedure for logging in is as follows:

1. The user logs in to QVWS using forms or basic authentication to send in the credentials.
2. The customized `Authenticate.aspx` checks the details towards an external security system (for example, an LDAP server or a SAML identity provider).
3. If successful, `Authenticate.aspx` transfers the user information to the QVS.

## Example

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



In appendix C you have an example of an `authenticate.aspx` for LDAP.

## How to choose solution

All three solutions support the principle of authenticating users outside QlikView and add flexibility to the QlikView product. Which solution that is the best is based on the architecture where the solution is going to be used?

- As a basic principle, header solution makes for easier integration into portals, but needs development or adaption of reverse proxy functionality in the portal or web frontend.
- The WebTicket solution is best in environments where a third party authentication system can be used to integrate 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 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

The example below is based on that the authentication is performed by an external party and that 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 WebTicket code

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

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Xml.Linq;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (txtServerName.Text == string.Empty) txtServerName.Text = System.Environment.MachineName;
        if (txtUserName.Text == string.Empty) txtUserName.Text = System.Security.Principal.WindowsIdentity.GetCurrent().Name;

        if (txtTry.Text == string.Empty) txtTry.Text = "http://" + txtServerName.Text + "/qlikview/";
        if (txtBack.Text == string.Empty) txtBack.Text = Request.Url.ToString().Replace("Default", "About");
    }
    protected void btnGetWebTicket_Click(object sender, EventArgs e)
    {
        StringBuilder groups = new StringBuilder();
        if (!string.IsNullOrEmpty(txtGroups.Text))
        {
            groups.Append("<GroupList>");
            foreach (string group in txtGroups.Text.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>{1}</Global>",txtUserName.Text, groups);
    }
}
```

```

        string result = CommHelper.Execute(webTicketXml, new Uri("http://" + txtServerName.Text +
"/QvAJAXZfc/GetWebTicket.aspx"), chkAnonymous.Checked, 10000);
        if (string.IsNullOrEmpty(result))
        {
            txtTicket.Text = "No result : " + result;
            return;
        }
        XDocument doc = XDocument.Parse(result);
        txtTicket.Text = doc.Root.Element("_retval_").Value;
    }
    protected void lnkCallAccesspoint_Click(object sender, EventArgs e)
    {
        Response.Redirect("http://" + txtServerName.Text +
            string.Format("/QvAJAXZfc/Authenticate.aspx?type=html&webticket={0}&try={1}&back={2}", txtTicket.Text, txtTry.Text,
txtBack.Text));
    }
    protected void lnkAuthError_Click(object sender, EventArgs e)
    {
        Response.Redirect("http://" + txtServerName.Text +
            string.Format("/QvAJAXZfc/Authenticate.aspx?type=html&webticket={0}&try={1}&back={2}", "crapticket", txtTry.Text,
txtBack.Text));
    }
}

```

## 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;
            }
        }
    }
}

```

# QlikView

```
    }
    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 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 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 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 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 .NET interface that represents any sort of user and the user's groups is `QlikView.AccessPoint.IUser` in `Accesspoint.dll`. 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 user name 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 user name 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 with 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 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 will actually issue 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 on each request because they come from something inherent in the request (for example, basic, 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()`.

- The groups are Windows SIDs.
- If there is no such authentication configured, TODO

```
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

“NTDOMAIN\XXX”. 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.