Oracle ® Enterprise Data Quality

Security Group Filtering

Version 9.0

January 2012

ORACLE

Table of Contents

1 Introduction

This document describes a sample plug-in script which performs custom, run-time filtering of user authorization groups, based on the user's location as determined by IP address.

1.1 Background

This plug-in is designed to address legislative requirements which state that some data must not be taken out of a particular country. In particular, a user logging on from a different country must not be able to view or access the restricted data, even if they would normally have sufficient privileges to do so.

Project access in OEDQ is normally controlled by assigning users and projects to *groups*. Both users and projects may have multiple groups. A user has access to a project if they are a member of at least one of the project's groups as assigned to the project in Director. The plug-in described here operates at run-time to filter the groups assigned to a user, based on the user's IP address at the time they log on.

For this script to work as intended, projects should be assigned to groups based upon their data access restrictions. That is, all projects that may only be accessed from within country A should be assigned to one group, projects that may only be accessed from country B should be assigned to a second group. Any projects with no country-based access restrictions can be made available to all groups. Users can then be granted access to each group as usual, and the plug-in script will provide per-session, IP address-based filtering of the groups actually available to a user.

This script can be configured to filter user authorizations based on either IPv4 or IPv6 addresses. It is not possible to filter on both IPv4 and IPv6 addresses at the same time.

NOTE: Users with the Add Project permission (usually administrators and power users) always have access to all projects. This bypasses the group system and is unaffected by this plug-in. By default, the Administrators and Project Owners user groups have this permission. It is possible to circumvent this issue by removing the Add Project permission from all users once all the system has been fully configured. If it is necessary to create further projects in the future, a user can be granted the Add Project permission as a temporary measure.

2 The authorizations filter plug-in

This plug-in consists of a JavaScript plug-in script plus configuration data which can be provided either as an XML file or as a comma-separated list . The user can activate the plug-in and select which type of configuration file is to be used by editing the **security.properties** file in the OEDQ server configuration directory.

2.1 Installation

The plug-in is installed by adding files to, and editing files in, the OEDQ configuration directory. This directory is found at the top level of the location in which OEDQ was installed and is named **config**. For example, if you have installed OEDQ using the provided Windows installer using the default installation path, your configuration directory will be at **C:\Program Files\Datanomic\dnDirector\config**. On a Linux or Unix platform, the OEDQ files will typically be stored under **/opt/dndirector**, in which case the configuration directory will be at **/opt/dndirector/config**.

To install the filter:

- Create a new script file, **userfilter.js**, in your configuration directory.
- Place the JavaScript code from Appendix A into **userfilter.js**.
- Create a new file to hold the IP address filtering rules in your configuration directory. Filter rules can be expressed either in CSV format, in which case your rules file should be named **ipranges.csv**, or in XML format, in which case your rules file should be named **ipranges.xml**. You will add data to these files in the configuration step, described in section 2.2 below.
- Edit the **security.properties** file in the configuration directory, or create it if it does not already exist, and add the line:

```
user.filter.scriptfile = userfilter.js
```

- Next, add one of the two following lines to **security.properties**. If you wish to use the XML version of the configuration file, add:

```
user.filter.xfile = ipranges.xml
```

If you wish to use the CSV version of the configuration file, add:

```
user.filter.cfile = ipranges.csv
```

- Finally, restart the application server so that it can pick up the new settings from the properties files.

2.2 Configuration

This plug-in can accept configuration data either in XML or CSV format. In either case, the configuration data maps a group to one or more IP ranges which correspond to permitted access locations. If a user logs on to the system from an IP address outside the permitted ranges, the associated group will be blocked from the user for the duration of the session.

The data in these files can be edited to modify the location-based filtering of project access. When editing the data, whichever file format is used, the following points should be remembered:

- If a group is not mentioned in the configuration file, no location-based filtering will be applied to it.
- The IP address ranges in the files are those which are allowed to access projects in the associated groups. To allow access to a group from more locations, add an IP range or widen the scope of an existing IP range. To disallow access to a group from some locations, remove the corresponding IP address range, or narrow the scope of the relevant range.

2.2.1 XML file format

The XML configuration file has the following structure:

```
<ipranges>
<group name="name">
<iprange start="x" end="y"/>
...
</group>
...
     </ipranges>
```

The configuration data consists of one or more `<group>` elements, where each group is identified by name. Each group specifies one or more IP address ranges that are permitted access to the projects in that group. An address range is specified as an `<iprange>` element, with a start and an end attribute defining the limits of the address range. In this way, multiple valid IP address ranges can be configured for each group.

All the group/IP range mapping data in the file must be contained within the `<ipranges>` tag.

For example, suppose an XML configuration file contains the following data:

```
<ipranges>
<group name="group1">
<iprange start="1.1.1.0" end="1.1.1.20" />
<iprange start="10.1.0.0" end="10.1.0.255" />
</group>
<group name="group2">
<iprange start="10.8.1.0" end="10.8.1.125" />
</group>
     </ipranges>
```

This configuration data means that:

- Projects that belong to group1 can only be accessed by logging on from an IP address that is either in the range 1.1.1.0 to 1.1.1.20 or in the range 10.1.0.0 to 10.1.0.225
- Projects that belong to group2 can only be accessed by logging on from an IP address in the range 10.8.1.0 to 10.8.1.125.

Note that if a group is not specified in the configuration file, access to projects in that group will not be controlled by IP address. For example, user access to projects belonging to a third group named group3 will be unaffected by this script and configuration data.

2.2.2 CSV file format

The CSV configuration file format specifies one group/address range mapping per line. Each line is consists of three comma-separated fields, as follows:

```
group name, start of address range, end of address range
```

For example, suppose a CSV configuration file contains the following data:

```
group1, 1.1.1.0, 1.1.1.20
group2, 10.8.1.0, 10.8.1.125
     group1, 10.1.0.0, 10.1.0.255
```

This configuration file is functionally identical the the sample XML given above. That is:

- Projects that belong to group1 can only be accessed by logging on from an IP address that is *either* in the range 1.1.1.0 to 1.1.1.20 *or* in the range 10.1.0.0 to 10.1.0.225
- Projects that belong to group2 can only be accessed by logging on from an IP address in the range 10.8.1.0 to 10.8.1.125.

Note that it is not necessary for all the valid IP address ranges for a group to be specified on adjacent lines. Again, if a group is not present within the file, no IP address filtering will be performed for that group.

Appendix 1: The Filter Script

The filter script

```
// User filter test which reads CSV or XML
// =======================================
addLibrary("logging");
addLibrary("io");

// Main filtering function for the script
// ======================================
function filter(user, ip)
{
  logger.log(Level.INFO,
        "Filtering user {0} from IP {1}",
            user.getUserName(), ip);
  var xprop = props["user.filter.xfile"];
  var cprop = props["user.filter.cfile"];
  var cfile = cprop == null ? null : findFile(cprop);
  var xfile = xprop == null ? null : findFile(xprop);

  if (cfile == null && xfile == null)
  {
    logger.log(Level.INFO, "No IP range files available");
    return true;
  }

  // Process CSV file
  // ================
  if (cfile != null)
  {
    // CSV file has group name and IP start/end on each line; a group
    // matches if the IP is in any of the ranges
    var hash= new Object();
    var rdr= IO.createCSVReader(cfile);
    var arr;

    while ((arr = rdr.read()) != null)
    {
      // Ignore lines with too few fields
      if (arr.length >= 3)
      {
        // Store group in hash once only
        var grp= trim(arr[0]);
        var start = trim(arr[1]);
        var end= trim(arr[2]);
        if (hash[grp] == null)
```

```
      {
        hash[grp] = false;
      }

      if (ipInRange(ip, start, end))
      {
        hash[grp] = true;
      }
    }
  }

  // Now reject any groups which did not match
  for (var g in hash)
  {
    if (!hash[g])
    {
      user.removeGroupByName(g);
    }
  }
}

// Process XML file
// XML schema is:
//
//<ipranges>
//<group name="name">
//<iprange start="x" end="y"/>
//...
//</group>
//...
// </ipranges>
//
// The purifyXML call here removes the <?xml ..?> header
// which E4X does not like
// ================
if (xfile != null) {
  var xml= new
       XML(XMLTransformer.purifyXML(IO.load(xfile)));
  var grps= xml.group;
  var ng= grps.length();
  for (var i = 0; i < ng; i++)
  {
    var grp = grps[i];
    var ranges = grp.iprange;
    var ok= false;

    for (var j = 0; j < ranges.length(); j++)
```

```
      {
        var range = ranges[j];
        if (ipInRange(ip, range.@start, range.@end))
        {
          ok = true;
          break;
        }
      }

      if (!ok)
      {
        user.removeGroupByName(grp.@name);
      }
    }
  }
  return true;
      }

// Trim function
// =============
function trim(a)
{
  return a.replace(/ /g, '');
}
```