|  |  |
| --- | --- |
| C:\WINNT\Profiles\Administrator\Desktop\logo_ec_17_colors_300dpi.gif | EUROPEAN COMMISSIONDIRECTORATE-GENERAL FOR HEALTH AND FOOD SAFETYGeneral Affairs**Information systems** |

Contact:
SUBIGER Jérôme, jerome.subiger@ext.ec.europa.eu

OpenNCP

# Proxy configuration

## Issue description:

Deployment of OpenNCP in a corporate architecture could cause issues about access to external resources at least when the module needs to use an authenticated proxy throughout these protocols: HTTP, HTTPS, SOCKS, FTP.

Following this, some manual configuration should solve the problem (modification of the axis2.xml file, adding some parameters to the JVM):

http.proxyHost=<proxyAddress>

http.proxyPort=<proxyPort>

https.proxyHost=<proxyAddress>

https.proxyPort=<proxyPort>

Otherwise application returns a ConnectionRefused Exception or NotAuthenticated Exception.

## Solution description

Java doesn't support/ignores parameters -Dhttp.proxyUsern = someUserName and -Dhttp.proxyPassword = somePassword.

The only way to solve this issue is to execute the authentication process into the Java source code.

## Java implementation:

### Proxy Authenticator Implementation:

In order to provide username/password of the proy, the Java class AuthenticatedCTS2Webservice should be modified with a call to: Authenticator.setDefault(new ProxyAuthenticator("USER", "PWD"));

***public class ProxyAuthenticator extends java.net.Authenticator{***

 ***private String user, password;***

 ***public ProxyAuthenticator(String user, String password) {***

 ***this.user = user;***

 ***this.password = password;***

 ***}***

 ***protected PasswordAuthentication getPasswordAuthentication() {***

 ***return new PasswordAuthentication(user, password.toCharArray());***

 ***}***

***}***

### ProxySelector Implementation:

The next step is to extend the ProxySelector class in order to implement or own ProxySelector by providing references to the HTTP or HTTPS proxy.

***public class MyProxySelector extends ProxySelector {***

***// Reference to default Proxy.***

***ProxySelector defsel = null;***

***/\****

 ***\* Inner class representing a Proxy.***

 ***\*/***

***class InnerProxy {***

 ***Proxy proxy;***

 ***SocketAddress addr;***

 ***int failedCount = 0;***

 ***InnerProxy(InetSocketAddress a) {***

 ***addr = a;***

 ***proxy = new Proxy(Proxy.Type.HTTP, a);***

 ***}***

 ***SocketAddress address() {***

 ***return addr;***

 ***}***

 ***Proxy toProxy() {***

 ***return proxy;***

 ***}***

 ***int failed() {***

 ***return ++failedCount;***

 ***}***

 ***}***

***/\****

 ***\* A list of proxies, indexed by their address.***

 ***\*/***

***HashMap<SocketAddress, InnerProxy> proxies = new HashMap<SocketAddress, InnerProxy>();***

***public MyProxySelector(ProxySelector def) {***

 ***// Save the previous default***

 ***defsel = def;***

 ***// Populate the HashMap (List of proxies)***

 ***InnerProxy i = new InnerProxy(new InetSocketAddress("psbru.ec.europa.eu", 8012));***

 ***proxies.put(i.address(), i);***

***}***

***/\****

 ***\* This is the method that the handlers will call. Returns a List of proxy.***

 ***\*/***

***public java.util.List<Proxy> select(URI uri) {***

 ***// Let's stick to the specs.***

 ***if (uri == null) {***

 ***throw new IllegalArgumentException("URI can't be null.");***

 ***}***

***/\****

 ***\* If it's a http (or https) URL, then we use our own list.***

 ***\*/***

***String protocol = uri.getScheme();***

***if ("http".equalsIgnoreCase(protocol)***

 ***|| "https".equalsIgnoreCase(protocol)) {***

 ***ArrayList<Proxy> l = new ArrayList<Proxy>();***

 ***for (InnerProxy p : proxies.values()) {***

 ***l.add(p.toProxy());***

 ***}***

***return l;***

***}***

***/\****

 ***\* Not HTTP or HTTPS (could be SOCKS or FTP) defer to the default selector.***

 ***\*/***

***if (defsel != null) {***

 ***return defsel.select(uri);***

***} else {***

 ***ArrayList<Proxy> l = new ArrayList<Proxy>();***

 ***l.add(Proxy.NO\_PROXY);***

 ***return l;***

***}***

 ***}***

***/\****

 ***\* Method called by the handlers when it failed to connect to one of the***

 ***\* proxies returned by select().***

 ***\*/***

***public void connectFailed(URI uri, SocketAddress sa, IOException ioe) {***

 ***if (uri == null || sa == null || ioe == null) {***

 ***throw new IllegalArgumentException("Arguments can't be null.");***

***}***

***/\****

 ***\* Let's lookup for the proxy***

 ***\*/***

***InnerProxy p = proxies.get(sa);***

***if (p != null) {***

 ***if (p.failed() >= 3)***

 ***proxies.remove(sa);***

 ***} else {***

 ***if (defsel != null)***

 ***defsel.connectFailed(uri, sa, ioe);***

 ***}***

***}***

## Proxy Credentials:

If we prefer to prevent static calls into the Java source code, proxy configuration properties should be stored into the EPSOS\_PROPERTIES database with different flags:

***APP\_BEHIND\_PROXY: true or false.***

***APP\_PROXY\_ HOST:*** [***http://proxyAddress***](http://proxyAddress)

***APP\_PROXY\_PORT: 9999.***

***APP\_PROXY\_USER: Username.***

***APP\_PROXY\_PASSWORD: Password.***

In this way, these properties will be also available for all EHealth components

# Migration to WebLogic

## Issue description

The European Commission corporate application server for Java applications is WebLogic. This can be the case for other institutions as well. DG Sante has tried to migrate the OpenNCP into WebLogic without succeeding. The main problems relate to incompatible libraries and java version. Migration work only when Weblogic 10.3.5 with the jkd1.6. is used. All other combinations don't work.

## Tested environments

* Weblogic 12.1.3 with jkd1.7\_xxx: Failure
* Weblogic 12.1.3 with jkd1.6\_xxx: Failure
* Weblogic 11 R1 (10.3.6) with jdk 1.7\_xxx: Failure
* Weblogic 11 R1 (10.3.6) with jdk 1.6\_xxx: Failure
* Weblogic 10.3.5 with the jkd1.6\_xxx: Success

## Incompatible library

Compatibility issues with com.sun.xml.ws.\* libraries which were in webservices-rt-2.1-b16.jar. The exception encountered is

"com.sun.xml.ws.transport.http.servlet.WSServletException: WSSERVLET11: failed to parse runtime descriptor: java.lang.ExceptionInInitializerError"

## Attempted fixes

We tried to solve the issue by creating an jar file, inserting a weblogic-application.xml file in which, we do a configuration so that Weblogic loads preferentially the libraries in webservices-rt-2.1-b16.jar. This solution has not given satisfactory results.

We also removed certains jar files as xercesImpl-2.10.0.jar and xml-apis-1.4.01.jar from Epsos modules to ensure compatibility with Weblogic librairie: Also without success.

We also added a weblogic.xml file to the epsos modules with the <prefer-web-inf-classes>true</prefer-web-inf-classes>, but it didn't help.

## Other problems related to WebLogic migration

In certain cases (depends on the configuration), we encountered many ClassCastException between weblogic.xml.jaxp.RegistrySAXParserFactory and javax.xml.parsers.SAXParserFactory