Paper 365-2011

Single Sign-On Configuration and Troubleshooting for SAS® 9.2 Enterprise BI Web Applications

Stuart J Rogers and Heesun Park, SAS Institute Inc., Cary, NC

ABSTRACT

Single Sign-On (SSO) means different things for different configurations and applications. The Single Sign-On logic and process for SAS® 9.2 Enterprise BI Web applications are examined based on various enterprise-level Web infrastructure configurations in which SAS Web applications are deployed. With such deployments, troubleshooting configuration and authentication errors is confusing and time-consuming. This paper explains Single Sign-On in terms of initial source of authentication, security arrangement, and trust relationships among the Web components and how SAS Web applications manage and consume externally authenticated user identity. Then this paper explores the troubleshooting strategies available for each of the enterprise-level Web infrastructure configurations. A methodology, along with recommended solutions, is presented for common error messages.

INTRODUCTION

WEB APPLICATION PROCTECTION MECHANISM

Security implementation of J2EE Web applications consists of two parts. One part is authentication, which basically is an access control to the Web application itself. The other part is authorization, which controls what operation is allowed on resources, such as servers and data, by the authenticated user. This paper mainly focuses on the authentication process for Web applications, which requires coordination with the existing Web infrastructure. The SAS 9.2 Enterprise Business Intelligence bundle comes with a SAS® Metadata Server which provides very sophisticated resource permission control. The goal of SSO is to use the authentication mechanism provided by the enterprise through its user registry and to seamlessly integrate this mechanism with the SAS 9.2 Metadata Server using the resource authorization service.

J2EE Web applications, including SAS 9.2 Web applications, can be protected in two ways. One way is that the Web application itself provides its own authentication mechanism. SAS metadata-based authentication (or the SAS "host" authentication) falls into this category. This method is efficient for a simple configuration that does not require Web perimeter security protection through an external user registry. The other way to delegate the authentication for the Web application to the application server is through the Web application's deployment descriptor (the `web.xml` file). In the deployment descriptor, the authentication method and security role mapping for the Web application can be defined (among other things). This mechanism allows the use of many types of authentication methods as well as integration with the Web space protection security package through an external user registry, such as the Lightweight Directory Access Protocol (LDAP) server. This method supports large and complex enterprise-level security configuration.

Note that application server can delegate the authentication responsibility to the Web security package or to the network domain. In other words, the application server "trusts" the other party for user authentication and accepts the authenticated user. Clearly tight integration between the application server and the authentication provider is required to safely transport the authenticated user information.

JAAS AND SSO

Java Authentication and Authorization Service (JAAS) is the backbone of the authentication process for the Web application server and Web applications. In essence, JAAS is a Java based implementation of a Pluggable Authentication Module (PAM).In its simplest form, it is like Java Database Connectivity (JDBC), an abstraction over authentication module providers. To a large extent, JAAS makes Web applications independent from the authentication method and thus makes the Web applications portable. Implementation of JAAS consists of a stack of JAAS login modules that handle different types of authentication methods. For example, an application server carries JAAS login modules for basic authentication, certificate-based authentication, token-based authentication, and so on. Often times, these JAAS login modules are considered "system" login modules. A Web application also can provide its own JAAS login modules. The SAS 9.2 Enterprise Business Intelligence bundle supplies two JAAS login modules. One module is for SAS "host" authentication. The "host" is typically based on a local operating system, and the module is called OMILoginModule. The other module is for the "trusted" Web authentication and is called TrustedLoginModule. Clearly, the major difference between these two modules is the way each module receives the user credentials. JAAS login configuration for a SAS 9.2 Web application can be specified in the `login.config` file or by using the administration tool that is provided by the application server. At the end of the JAAS system login

module processing, the application server initializes the JAAS "Subject" object and adds the externally authenticated user information into the "Subject" as JAAS "Principal". In SAS 9.2, the SAS TrustedLoginModule gets integrated into the login stack for the application server and exploits the "Principals" in the "Subject" for SAS resource authorization.

After a successful JAAS authentication, the application server can create a special token (or cookie) to enable application server based SSO. The Lightweight Third Party Authentication (LTPA) token from IBM WebSphere is a good example. It will be covered more in detail in the application server based SSO section.

When a reverse proxy security server (such as CA SiteMinder or IBM Tivoli Access Manager WebSEAL) is configured in front of the application server, typically the application server delegates the responsibility of user authentication to the reverse proxy server or "trusts" reverse proxy server for user authentication. In this scenario, the application server requires a special JAAS login module that can handle the encrypted user information that is passed in from the reverse proxy server. This special JAAS login module should run before any other JAAS login modules and is implemented as "interceptor." For WebSphere application servers, this implementation is called "Trust Association Interceptor" (TAI). It is also known as "Identity Asserter" in other application servers.

This trust relationship between the reverse proxy server and the application server represents the SSO agreement between them.

SCOPES OF SINGLE SIGN-ON

SAS INFORMATION DELIVERY PORTAL WEB APPLICATION BASED SSO

By default, SAS 9.2 configuration and authentication for SAS Web application access is handled by the SAS Logon Manager and the JAAS login module that is supplied by SAS (OMILoginModule). The SAS 9.2 configuration is independent from the protection mechanism for the Web application on the application server. The user registry could be based on local operating system accounts or user identities based on the LDAP server. User permissions and other properties for SAS resources are defined and maintained in the SAS metadata. Through its variety of portlets, SAS® 9.2 Information Delivery Portal is designed to be a central access point to all SAS 9.2 Web applications. After a user is authenticated to access the SAS Portal, the SAS Portal keeps the user and session context. When you try to access other SAS Web applications that are defined as portlets, the SAS Portal passes in the user and session context to the target SAS Web application. This way, one can access multiple SAS Web applications with a single authentication to the SAS Portal, so you have achieved SSO. It is a very narrowly defined SSO for the family of SAS 9.2 EBI Web applications since all SAS Web application access goes through the SAS Portal. Note that SSO through the SAS Portal does not use application server approach that creates cookies. The session information created by the SAS Portal is valid only for the SAS Portal. After successfully logging in to the SAS Portal, typing in the URL for a SAS 9.2 EBI Web application in a browser will trigger another authentication challenge through SAS login page.

Figure 1 below shows the implementation of the SAS Information Delivery Portal using SSO. The SAS Information Delivery Portal maintains the authenticated user information and session information, and passes this information to other SAS Web applications, when they are invoked through the SAS Information Delivery Portal.

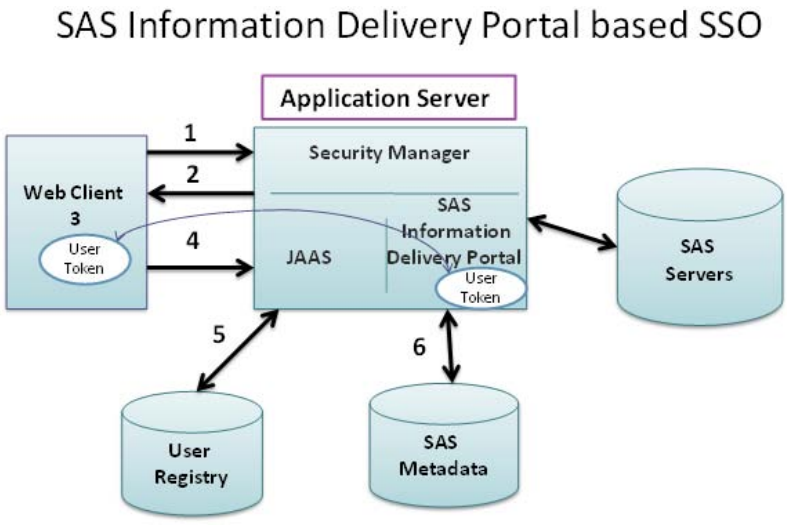

Figure 1. SAS Information Delivery Portal Using SSO

APPLICATION SERVER BASED SSO

In an enterprise-level configuration, an application server has its own user registry (mostly the LDAP server) and provides user authentication service for Web applications. To take advantage of the user authentication service that is provided by the application server, a Web application should have the `<security-constraint>` element in its deployment descriptor file (`web.xml`). This file designates the authentication method to use and the security role name for security role mapping. From a SAS 9.2 Web application perspective, this configuration is referred as trusted Web authentication because it delegates the responsibility of user authentication to the application server and trusts the user that is authenticated by the application server. As the application server authenticates the user, implementation of SSO is up to the application server. Not all application servers support SSO for that reason. When it supports SSO, the application server provides an option to enable or disable SSO capability. Use of special token or cookie created by the application server after initial user authentication is a typical SSO implementation. The Lightweight Third Party Authentication (LTPA) token for WebSphere application servers is a good example. The SSO token gets passed to the browser as a part of HTTP header. The access to the Web applications that share the same authentication mechanism, including Web applications outside of SAS, will not trigger any additional authentication challenge as the request carries the SSO token from the browser. Naturally, when the application server supports the SSO, the JAAS logon module that checks the SSO token is on top of the JAAS operation stack.

Figure 2 below depicts the implementation of application server based SSO. The application server creates the SSO token and passes it to the browser. The subsequent HTTP request from the browser carries SSO token that contains the authenticated user information. Any participating Web applications that are deployed under the same application server instance and use the same authentication method can be accessed without further authentication.

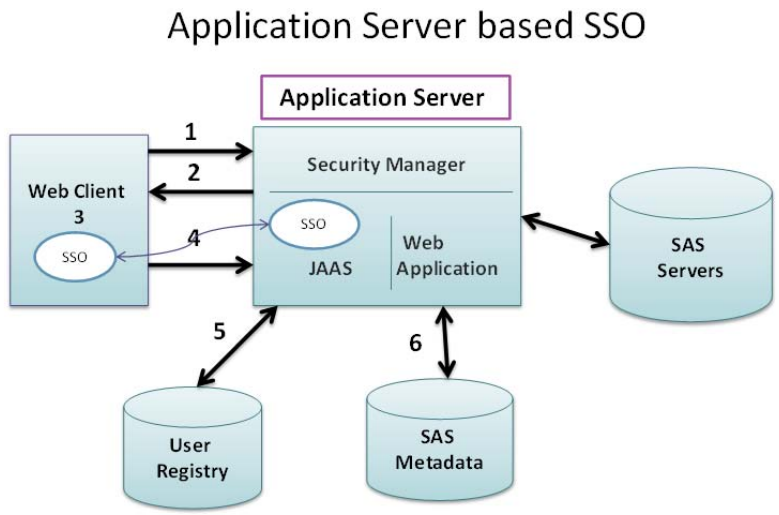

Figure 2. Application Server Based SSO

THIRD-PARTY SECURITY PACKAGE BASED SSO

Web infrastructure of large organizations typically includes a reverse proxy security server (RPSS) that authenticates users before a user can access the Web applications deployed under the application server. CA SiteMinder or IBM Tivoli Access Manager (TAM) WebSEAL belong in this category. After the initial user authentication is successful, the RPSS creates a heavily encrypted special token or special HTTP header that contains the user information. This mechanism is considered a lot safer than that of conventional minimally encrypted user name and password method.

Here is how a Web application works with trusted Web authentication through RPSS. A tricky part is the handling of user credentials between RPSS and the application server. Because the application server relies on the RPSS for user authentication, the configuration needs to set up an entity in the application server that could receive the request from the RPSS and place the user information into application server's data structure. For a WebSphere application server, this entity is called "Trust Association Interceptor" (TAI). For an Oracle WebLogic Server, this piece is called "Identity Asserter" (IA). In both cases, the user credential does not include a password since the user is already authenticated.

The "interceptor" module is a special case of JAAS login module that is placed on the top of the JAAS login module stack and is executed first during JAAS processing. The responsibility of this module is to decode the incoming token from the RPSS, to extract user name, and to initialize the JAAS "Subject" with the "Principal" that represents the

authenticated user. The “Principal” in the JAAS “Subject” would be consumed by other JAAS login modules including the SAS TrustedLoginModule.

As an authenticator, the RPSS supports SSO from its perspective. A special token (in case of CA SiteMinder, it is called SMSESSION cookie) is created after the successful user authentication is passed to the browser. The subsequent HTTP requests from the browser will carry the token which will be honored by the RPSS. From the application server perspective, after the interceptor module adds the JAAS “Principal” for the authenticated user, the application server creates its own SSO token (for example, the LTPA token for WebSphere application servers) as described in the previous section. This token also gets passed to the browser and allows SSO to the application server from that point on.

Figure 3 below shows the creation and the flow of the SSO token from the RPSS and application server.

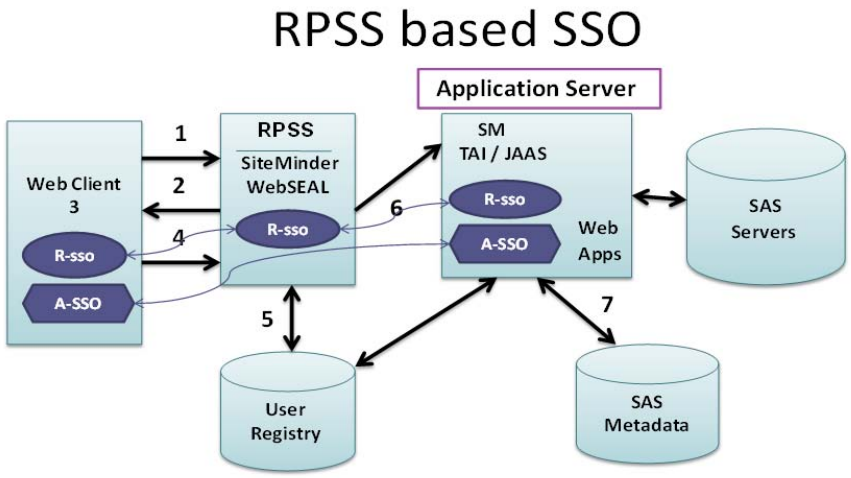

Figure 3. RPSS Based SSO

WINDOWS DOMAIN BASED SSO (INTEGRATED WINDOWS AUTHENTICATION)

Another popular SSO configuration is the Windows login based approach, which is also called Integrated Windows Authentication (IWA). This configuration assumes that clients and the application server for the SAS 9.2 Web applications are on the same Windows domain.

To properly comprehend the IWA, we have to understand the underlying protocols, such as the Kerberos protocol and Simple and Protected GSSAPI Negotiation (SPNEGO) protocols. The Kerberos protocol is the ticket-based mutual authentication mechanism developed by the Massachusetts Institute of Technology (MIT) and is the primary authentication protocol for the Windows network (since Windows Server 2003). When a user successfully logs into the Windows network, the user is represented by its Kerberos ticket. The Kerberos key distribution center (KDC) in conjunction with the domain controllers (DC) in Active Directory is the centerpiece of Kerberos ticket management. An application server (such as WebSphere) gets registered to the KDC with a service principal name (SPN) and becomes a legitimate Kerberos entity. A SPN gets mapped a user and uses its password to decode incoming service tickets. SPNEGO is the protocol supported by the browsers and application servers and is the wrapper of the underlying protocols, such as Kerberos or Microsoft NT LAN Manager (NTLM). Because application servers support only Kerberos, SPNEGO carries only Kerberos tickets. To access a Web application that is deployed in the application server through IWA, a browser requests a Kerberos service ticket for the target application server from the KDC. The data structure of the Kerberos service ticket is very complicated, but in essence, the SPN (application server) uses its password to decode the user information that is embedded in the service ticket, verifies this information, and accepts this information as an authenticated user who can access Web applications. From the application server perspective, this process is a part of its JAAS operation. In case of WebSphere, it is called SPNEGO Trust Association Interceptor (TAI), which is another “interceptor” approach. This approach is executed before any other standard JAAS login module is executed. Other application servers provide a similar approach, but the configuration process is specific to the application server.

Figure 4 below describes the IWA/SPNEGO support configuration for SAS 9.2 Enterprise Business Intelligence.

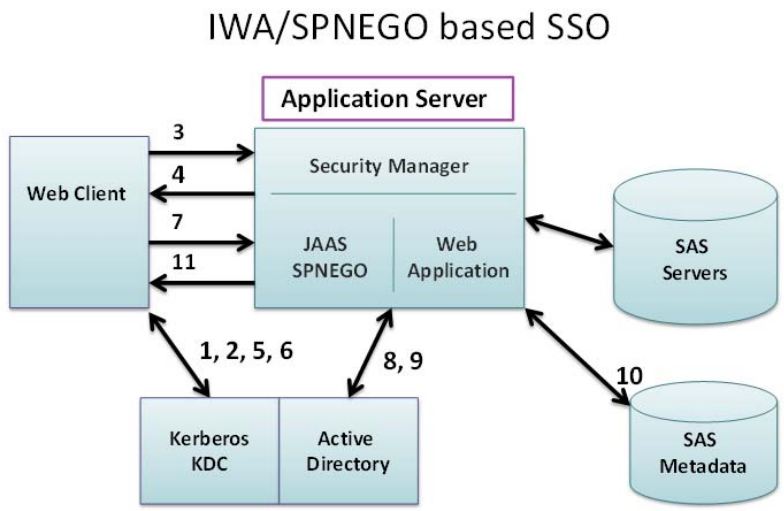

Figure 4. IWA/SPNEGO Configuration

1. User logs in to the Windows domain by requesting a Ticket Granting Ticket (TGT) from the KDC.
2. User receives a TGT from the Kerberos KDC.
3. User accesses a protected SAS Web application from the browser.
4. Application server sends a 401 error with Authentication-Negotiation option.
5. Browser requests a service ticket for the application server with service principal name (SPN).
6. Browser receives a service ticket for application server SPN.
7. Browser creates a SPNEGO token that contains the Kerberos service ticket and sends the token with the request.
8. SPNEGO login module in the application server decodes the SPNEGO token and the Kerberos service ticket to obtain the user name.
9. SPNEGO login module validates the user with the application server's registry (which is Active Directory) and adds the JAAS "Principal" into the JAAS "Subject".
10. SAS Web application produces output by accessing SAS servers and SAS resources.
11. Output is returned to the browser with session information.

    Note: The domain controller, client machine, and application server should be in the same Windows domain. The SAS server does not have to be in the same Windows domain.

There is significant difference in IWA/SPNEGO support between SAS 9.1.3 and SAS 9.2 Web applications. In essence, the authentication process in SAS 9.2 solely depends on the JAAS stack operation. The authentication process in SAS 9.1.3 grabs user information from the HTTP header before running the SAS TrustedLoginModule as a part of JAAS processing. The implication of the SAS 9.2 implementation is that IWA/SPNEGO should be supported by the application server itself. The SPNEGO support module in the application server decodes the SPNEGO token and the Kerberos ticket inside of it, extracts user information, and places the user "Principal" in the JAAS "Subject". This operation is basically the same as handling an encrypted SSO token from third-party security packages. The difference is the way a user gets authenticated initially.

TROUBLESHOOTING WEB AUTHENTICATION ISSUES

Attempting to troubleshoot Web authentication errors can be a daunting task. The previous section of this paper has detailed how the various components interact in an enterprise-level Web infrastructure. It is difficult to understand where to start when troubleshooting. This section of the paper aims to outline a process for troubleshooting as well as identifying which log files should be reviewed and how to enable additional logging. With the additional logging enabled, a tool will be presented to test the authentication process and some sample log messages and associated solutions presented.

The key principle for troubleshooting is to be systematic in your approach. The details in the previous section will assist in identifying the specific components within your enterprise-level Web infrastructure. These components should be addressed in order and one at a time so that the working components can be eliminated and the failing components can be diagnosed in further detail.

WHICH LOGS ARE IMPORTANT?

To be able to efficiently troubleshoot any issues with the single sign-on configuration, it is necessary to know which log files are important and what type information each log file contains. This section will outline the key log files and explain the type information to look for in each log file. Also this section will explain when during the authentication process information is written to the log files. From the SAS perspective, the two key log files to consider are the SAS Metadata Server log file and the SAS Remote Services log file.

SAS Metadata Server

The SAS Metadata Server log file will record the successful or unsuccessful authentication actions against the SAS Metadata Server. For example, a successful authentication action against the SAS Metadata Server with the middle tier configured for single sign-on will result in messages similar to Output 1.

```
INFO  [00006174] :sastrust@saspw - New client connection (185) accepted from server
port 8561 for user sastrust@saspw.  Encryption level is Credentials using
encryption algorithm SASPROPRIETARY.  Peer IP address and port are
[###.###.###.###]:4965.
INFO  [00006179] :sticdemo01 - New client connection (186) accepted from server
port 8561 for SAS token user sticdemo01.  Encryption level is Credentials using
encryption algorithm SASPROPRIETARY.  Peer IP address and port are
[###.###.###.###]:4968.
```

Output 1. Metadata Server Log Example

This example shows first the valid connection by the SAS Trusted user and then the subsequent connection by the end user. By default, the SAS Metadata Server log file can be found in

`SAS-configuration-directory`**`\Lev1\SASMeta\MetadataServer\Logs`**

SAS Remote Services

The SAS Remote Services log file can be found on the middle-tier server. By default, the SAS Remote Services log file will contain information messages regarding the Foundation Service. However, because single sign-on configuration requires changes to the SAS Remote Services class path, the log file can present any errors regarding failures to load the required classes. In addition, by making changes in the SAS® Management Console, it is possible to enable additional logging to present authentication messages in the SAS Remote Services log file. The details for enabling this additional logging are covered in the next section. Output 2 shows an example of the additional log messages for a successful single sign-on authentication.

```
[RMI TCP Connection(45)-149.173.96.253] DEBUG []
com.sas.services.user.UserContext.timing - Connecting to repository
omi://sgcwin065.race.sas.com:8561 took 31 milliseconds
[RMI TCP Connection(45)-149.173.96.253] DEBUG [] com.sas.services.user.UserContext
- Getting user credentials took: 0 milliseconds
[RMI TCP Connection(45)-149.173.96.253] DEBUG [] com.sas.services.user.UserContext
- All Credentials for domain WebAuth were requested.
[RMI TCP Connection(45)-149.173.96.253] DEBUG [] com.sas.services.user.UserContext
- domain: WebAuth  uid:  sticdemo01@!*(generatedpassworddomain)*!  pw: *******
```

Output 2. SAS Remote Services Log Example

Java Application Server

As we have seen from the previous section, the SAS components form only a small part of the overall enterprise-level Web configuration for single sign-on. So while understanding the two key log files from the SAS components is important, the key log files to consider when troubleshooting are the Java Application server log files. We have seen that the configuration for the Java application server is centered on the JAAS module configuration. These JAAS modules can be configured to provide additional logging information, and this additional information is in the Java application server's log files.

Depending on the Java application server used, the log information will be sent to different log files. By default, the JBoss Application Server sends the messages to the server log. . The location of the server log depends on your configuration. Here is an example location:

```
JBoss-Home\server\SASServer1\log\server.log
```

For an IBM WebSphere Application Server, the default log file is in

```
WAS_Home\profiles\<NodeName>\logs\SASServer1\SystemOut.log
```

Finally for an Oracle WebLogic Application Server, the default log file is in

```
SAS-configuration-
directory\Lev1\Web\SASDomain\servers\SASServer1\logs\SASServer1.out
```

Clearly, the Java application server log files will be the main source of troubleshooting information. However, in the default configuration quite often the log files will not contain sufficient information for effective troubleshooting. The next section will step through the process to produce additional information in the key log files.

HOW TO ENABLE ADDITIONAL LOGGING

The following sections provide detailed instructions for enabling additional logging to assist in troubleshooting single sign-on authentication issues. Once the issues have been successfully resolved, the additional logging should be disabled as large amounts of information will be added quickly to the log files and this will cause issues in a production system.

SAS Metadata Server

For troubleshooting single sign-on authentication issues, it is unlikely you will need to enable additional logging for the SAS Metadata Server. The default configuration will provide details of both successful and unsuccessful connection attempts. It is possible to enable further logging and provide details of the XML requests sent to the SAS Metadata Server. If you want to enable this level of logging for the SAS Metadata Server, contact SAS Technical Support for instructions.

SAS Remote Services

As outlined in the previous section, it is possible to increase the logging for SAS Remote Services such that the creation of user contexts is recorded in the log file. The increased level of logging will enable us to confirm the final step in the single sign-on process where the end user's contexts are created within SAS Remote Services. Use the SAS Management Console to enable this level of logging.

1. Log on to the SAS Management Console as an administrative or unrestricted user, for example sasadm@saspw.
2. From the **Plug-ins** menu, expand the **Foundation Services Manager**.
3. Expand **Remote Services** and then **Core**.
4. Right-click **Logging Service,** and select **Properties**. The Logging Service Properties window appears.
5. Click the **Service Configuration** tab, and click **Configuration**. The Logging Service Configuration window appears.
6. On the **Context** tab,click **New** to add a context, and enter the following information:

   Name com.sas.services.user

   Priority DEBUG

   Outputs SAS_LS_FILE
7. Click **OK** to return to the Logging Service Configuration window.
8. Click **OK** to save the new Logging Service configuration to the metadata repository.
9. Restart SAS Remote Services for the changes to take effect.

JBoss Application Server

JBoss Application Server makes use of the Apache log4j Java logging API. Log4j has four key objects; categories, priorities, appenders, and layouts. A category is a named object where the name is a case sensitive and has a hierarchical construct similar to the Java package namespace. The priority object represents the importance or level

of a message. By default, there are a small number of defined priorities which are known by the names: FATAL, ERROR, WARN, INFO and DEBUG. The appender is a logical message destination. Appenders are output targets for the Apache log4j API. An appender can be a file, an OutputStream, a java.io.Writer, a remote log4j server, a remote Unix Syslog daemon, or many other output targets. Finally, the layout object defines how the log message should be rendered into a string.

As with other parts of the JBoss Application Server configuration, the logging configuration is defined via XML files. The main logging configuration file for each server instance, `jboss-log4j.xml`, is located in

```
JBoss-Home\server\SASServer1\conf
```

The `jboss-log4j.xml` file contains definitions for the default appenders and logging contexts. The appender for the default JBoss server log file

```
JBoss-Home\server\SASServer1\log\server.log
```

is defined within the `jboss-log4j.xml` file along with the default priorities of a number of categories. The default configuration of the JBoss logging outputs a significant number of DEBUG and INFO messages to the log file. To make troubleshooting authentication issues more efficient, we can define an additional appender to output log messages relating to the authentication process. Adding the following appender to the `jboss-log4j.xml` file will allow us to send specific categories to this new log file.

```
<appender name="SECURITY_F"
class='org.jboss.logging.appender.DailyRollingFileAppender'>
    <param name="Append" value="true"/>
    <param name="DatePattern" value="'.'yyyy-MM-dd"/>
    <param name="File" value="${jboss.server.home.dir}/log/jboss.security.log"/>
    <layout class="org.apache.log4j.PatternLayout">
          <param name="ConversionPattern" value="%d{ABSOLUTE} %-5p [%c] %m%n"/>
    </layout>
</appender>
```

With the output defined, we can then add specific category definitions to the `jboss-log4j.xml` file to enable logging of the authentication process. One key option to include on all additional category definitions is `additivity="false"`. This option will ensure the log messages are sent only to the new appender. Without this option, the messages will go to both the default FILE appender and our new appender. To log messages about the creation of users' contexts, add the following to the `jboss-log4j.xml` file:

```
<category name="org.jboss.security.SecurityAssociation" additivity="false">
    <priority value="TRACE" />
    <appender-ref ref="SECURITY_F"/>
</category>
```

To log messages about reading the login-config.xml authentication definitions, add this category:

```
<category name="org.jboss.security.auth.login.XMLLoginConfigImpl"
additivity="false">
    <priority value="TRACE"/>
    <appender-ref ref="SECURITY_F"/>
</category>
```

To log messages specific to the JBoss SPNEGO Module, add this category:

```
<category name="org.jboss.security.negotiation.NegotiationAuthenticator"
additivity="false">
    <priority value="TRACE"/>
    <appender-ref ref="SECURITY_F"/>
</category>
```

However, only adding these categories will not provide a great deal of information regarding each of the login modules defined in `login-config.xml`. To obtain additional information regarding a specific module, it is necessary to enable the debug option for the specific module. This just requires adding the following option to the module definition:

```
<module-option name="debug">true</module-option>
```

Adding this option to the SAS TrustedLoginModule will cause the complete XML request and response to the SAS Metadata Server to be echoed to the default appender. Output 3 shows an example of such output.

```
INFO  [STDOUT] [SasLoginModule]  The login module is initialized.
INFO  [STDOUT] [SasLoginModule]  Connection established.
INFO  [STDOUT] [SasLoginModule]  TrustedLoginModule: closing trusted connection.
INFO  [STDOUT] [SasLoginModule]  Getting user identity from metadata.
INFO  [STDOUT] [SasLoginModule]  User identity retrieved from metadata;
INFO  [STDOUT] [SasLoginModule]  Client XML: <GetMetadata>
  <Metadata>
    <Person Id="A5ZBCPA7.AN000003" Name="" Desc="">
      <IdentityGroups/>
      <EmailAddresses/>
      <Locations/>
      <ExternalIdentities/>
    </Person>
  </Metadata>
  <NS>SAS</NS>
  <Flags>4</Flags>
  <Options>
    <Templates>
      <ExternalIdentity Name="" Context="" Identifier=""/>
      <Login Name="" UserId="" Password="" >
        <Domain/>
      </Login>
      <Email Name="" Address="" />
      <AuthenticationDomain Name="" />
      <IdentityGroup Name="" GroupType="">
        <IdentityGroups/>      </IdentityGroup>
    </Templates>
  </Options>
</GetMetadata>
INFO  [STDOUT] [SasLoginModule]  Server XML: <GetMetadata><Metadata><Person
Id="A5ZBCPA7.AN000003" Name="sasdemo"
Desc=""><IdentityGroups/><EmailAddresses/><Locations/><ExternalIdentities/></Person
></Metadata><NS>SAS</NS><Flags>4</Flags><Options><Templates><ExternalIdentity
Name="" Context="" Identifier=""/><Login Name="" UserId=""
Password=""><Domain/></Login><Email Name="" Address=""/><AuthenticationDomain
Name=""/><IdentityGroup Name=""
GroupType=""><IdentityGroups/></IdentityGroup></Templates></Options></GetMetadata>
INFO  [STDOUT] [SasLoginModule]  User identity retrieved from metadata;
```

Output 3. Debug Output from SAS TrustedLoginModule

Adding the debug option to the Kerberos module as part of the SPNEGO configuration will result in the main steps of the Kerberos Key Exchange being echoed to the default appender. Output 4 illustrates a successful authentication.

```
INFO  [STDOUT] Debug is  true storeKey true useTicketCache false useKeyTab true
doNotPrompt true ticketCache is null isInitiator true KeyTab is
C:\SAS9.2\appsrv05.keytab refreshKrb5Config is false principal is
HTTP/SGCWIN065.race.sas.com@RACE.SAS.COM tryFirstPass is false useFirstPass is
false storePass is false clearPass is false
INFO  [STDOUT] principal's key obtained from the keytab
INFO  [STDOUT] Acquire TGT using AS Exchange
INFO  [STDOUT] principal is HTTP/SGCWIN065.race.sas.com@RACE.SAS.COM
INFO  [STDOUT] EncryptionKey: keyType=23 keyBytes (hex dump)=0000: E1 9C CF 75 EE
54 E0 6B   06 A5 90 7A F1 3C EF 42  ...u.T.k...z.<.B
INFO  [STDOUT] Added server's keyKerberos Principal
HTTP/SGCWIN065.race.sas.com@RACE.SAS.COMKey Version 12key EncryptionKey: keyType=23
keyBytes (hex dump)=
E0 6B   06 A5 90 7A F1 3C EF 42  ...u.T.k...z.<.B
INFO  [STDOUT]          [Krb5LoginModule] added Krb5Principal
HTTP/SGCWIN065.race.sas.com@RACE.SAS.COM to Subject
```

```
INFO  [STDOUT] Commit Succeeded
INFO  [STDOUT]          [Krb5LoginModule]: Entering logout
INFO  [STDOUT]          [Krb5LoginModule]: logged out Subject
```

Output 4. JBoss Kerberos Log Example

To obtain a complete debug log for the Kerberos Key Exchange, it is necessary to set a further Java option. Unfortunately, this option needs to be set for the JVM, so this option needs to be included in the startup JVM arguments of the JBoss application server. When JBoss is installed as a Microsoft Windows Service, edit the file

`JBoss-Home`**`\server\SASServer1\wrapper.conf`**

and add the following additional parameter, ensuring the number scheme is consistent with the existing entries:

```
wrapper.java.additional.40=-Dsun.security.krb5.debug=true
```

When JBoss is run from a script on Microsoft Windows, UNIX, or Linux, then edit the file

`JBoss-Home`**`\bin\SASServer1.sh`**

and add the following to the end of the JAVA_OPTS line, ensuring there are no carriage returns within the line:

```
-Dsun.security.krb5.debug=true
```

With this additional debug option set, the full Kerberos Key Exchange will be echoed to the default appender. The information that will be written to the log will cover:

- Reading the Keytab file
- Loading values from the Keytab file
- Listing encryption types
- Creating the Kerberos message
- Type of encryption used on the Kerberos message
- Sending the Kerberos message to the Key Distribution Center (KDC)
- Receiving the message from the KDC
- Acquiring the Kerberos ticket
- Decrypting the service ticket
- Authenticating the end user

Each authentication attempt will generate approximately 150 lines in the log.

Clearly, enabling the TRACE log output creates a great deal of useful information for troubleshooting authentication issues. However, leaving this level of logging enabled for any amount of time will result in very large log files which will hamper any attempts to troubleshoot the authentication issues. Enable this level of logging only while testing the authentication issues.

IBM WebSphere

IBM WebSphere logs the messages that are most useful in troubleshooting to the defined server's Java Virtual Machine (JVM) log. By default, the log is located in

`WAS_HOME\profiles\<node_name>`**`\logs\SASServer1\SystemOut.log`**

Specific error messages are sent to

`WAS_HOME\profiles\<node_name>`**`\logs\SASServer1\SystemErr.log`**

The JVM logs are created by redirecting the System.out and System.err streams of the JVM to independent log files. The `System.out` log is used to monitor the health of the running application server. The `System.err` log contains exception stack trace information. One set of JVM logs exists for each application server and all of its applications.

In addition to the JVM logs, IBM WebSphere can generate diagnostic trace output. The diagnostic trace provides detailed information about how the application server components run within the managed process. Changes to the configuration of the diagnostic trace can be made to a running server or applied after a server restart. To configure the diagnostic trace in IBM WebSphere 7.0.x:

1. Access the **Integrated Solutions Console.**
2. Expand **Servers**. Then expand **Server Types** and select **WebSphere application servers.**
3. Select the desired application server from the table.

4. Select **Logging and tracing** from the **Troubleshooting** section.
5. Select **Diagnostic Trace.**
6. Click the **Runtime** tab.
7. Select **Change Log Detail Levels.**
8. Enter ***=info: com.ibm.ws.security.*=all** for trace information on the authentication process.
9. Select **OK.**

These changes take effect immediately and the new trace log file will be created in

`WAS_HOME\profiles\<node_name>`**`\logs\SASServer1\trace.log`**

Output 5 displays an example `trace.log` file with a successful authentication event using SPNEGO.

```
[1/18/11 3:30:20:010 EST] 00000064 EJSWebCollabo 3   Resetting invoked: Subject:
    Principal: sticdemo01@RACE.SAS.COM
    Principal: shqracedc01.race.sas.com:389/sticdemo01
    Principal: PFSPrincipal: sasdemo
    Principal: PFSPrincipal: sticdemo01@WebAuth
    Public Credential: com.ibm.ws.security.auth.WSCredentialImpl@73337333
    Private Credential:

token name:         com.ibm.wsspi.wssecurity.token.krbAuthnToken
version:            1
hashCode:           -1395948001
uniqueId:           sticdemo01
kerberos principal: sticdemo01
realm:              RACE.SAS.COM
expiration:         Tue Jan 18 05:20:38 EST 2011
renew until:        null
isReadOnly:         false
isAddressless:      false
isForwardable:      false
isRenewable:        false
KerberosTicket:     Kerberos TGT is null.
GSSCredential:      null
    Private Credential: com.ibm.ws.security.token.SingleSignonTokenImpl@f2a0f2a
    Private Credential: com.ibm.ws.security.token.AuthenticationTokenImpl@76b076b0
    Private Credential: com.ibm.ws.security.token.AuthorizationTokenImpl@ed00ed0
    Private Credential: com.sas.services.security.login.PFSCredential@19531953
 and received: Subject:
    Principal: sticdemo01@RACE.SAS.COM
    Principal: shqracedc01.race.sas.com:389/sticdemo01
    Principal: PFSPrincipal: sasdemo
    Principal: PFSPrincipal: sticdemo01@WebAuth
    Public Credential: com.ibm.ws.security.auth.WSCredentialImpl@73337333
    Private Credential:
```

Output 5. IBM WebSphere Debug SPNEGO Log Example

For IBM WebSphere to log the specifics of the SAS TrustedLoginModule, it is necessary to enable the debug option on the module itself. This option is read only at startup, so the application server must be restarted after the debug option is enabled. To enable the debug option:

1. Access the **Integrated Solutions Console.**
2. Expand **Security**, and then select **Global Security.**
3. Expand **Java Authentication and Authorization Service.**
4. Select **System logins.**
5. Select **WEB_INBOUND.**
6. From the table, select **com.sas.services.security.login.websphere.WSTrustedLoginModule.**

7. In the table of custom properties, either edit the value of **debug** to **true,** or if the property is not present, add the property with name **debug** and value **true**
8. Select **OK** and synchronize changes

With the debug option enabled, the `SystemOut.log` file will contain the debug messages from the SAS TrustedLoginModule. For a successful authentication process, the log will contain messages similar to those shown in Output 3.

To obtain debug messages for the SPNGEO configuration with IBM WebSphere, it is necessary to edit two custom properties to the Java Virtual Machine definition for the application server. To edit the required custom properties:

1. Access the **Integrated Solutions Console.**
2. Expand **Servers** and then **Server Types.**
3. Select **WebSphere application servers.**
4. Select the application server from the table.
5. Expand **Java and Process Management.**
6. Select **Process definition.**
7. Select **Java Virtual Machine.**
8. Select **Custom properties.**
9. Specify the following two properties set their value to **ALL:**

Name	Value
com.ibm.security.jgss.debug	ALL
com.ibm.security.krb5.Krb5Debug	ALL

Table 1. Required Debug Custom Properties

10. Synchronize changes.
11. Re-start the application server to read the two custom properties defined for the Java Virtual Machine.

These debug options will produce a large amount of output in the `SystemOut.log` file. For example, a successful authentication process will generate approximately 800 lines in the log. This output will include KRB_DBG_KTAB messages as the keytab is read, JGSS_DBG_MARSH messages showing the HEX value of tokens passed, KRB_DBG_CRYP messages showing the cryptography used for the tokens, and JGSS_DBG_CTX messages regarding the creation and updating of the context.

Finally to obtain even more detailed logging of the SPNGEO authentication process, enable diagnostic trace logging for `com.ibm.ws.security.spnego.*=all`. To enable this diagnostic trace logging:

1. Access the **Integrated Solutions Console.**
2. Expand **Servers**. Then expand **Server Types** and select **WebSphere application servers.**
3. Select the desired application server from the table.
4. Select **Logging and tracing** from the **Troubleshooting** section.
5. Select **Diagnostic Trace.**
6. Click the **Runtime** tab.
7. Select **Change Log Detail Levels.**
8. Enter ***=info: com.ibm.ws.security.spnego.*=all** for trace information on the authentication process.
9. Select **OK.**

These changes take effect immediately, and the new trace log file will be created in

`WAS_HOME\profiles\<node_name>`**`\logs\SASServer1\trace.log`**

This level of diagnostic trace will generate even more logging output than the previous JVM custom properties. For example, a successful authentication attempt will generate approximately 1200 lines in the log. This level of

diagnostic trace logging can also be enabled without the two custom JVM properties. In this case, a successful authentication attempt will generate approximately 600 lines of log messages.

Oracle WebLogic

Oracle WebLogic sends logging output to the defined server log. By default, all messages go to the log file. The server message log does not contain HTTP requests, JMS messages, or JTA transaction messages. In addition to maintaining its local message log, by default each server forwards messages of severity NOTICE and higher to the domain log. By default, the server log file is located in

```
SAS-configuration-directory
\Lev1\Web\SASDomain\servers\SASServer1\logs\SASServer1.log
```

In addition to the `SASServer1.log`, the standard output of the JVM is redirected to the following file:

```
SAS-configuration-directory
\Lev1\Web\SASDomain\servers\SASServer1\logs\SASServer1.out
```

Additional debug messages can be sent to the server log. This is enabled through the WebLogic Server Administration Console:

1. Access the **Administration Console.**
2. Expand **Environment** and select **Servers.**
3. Select the server name from the table.
4. Click the **Debug** tab.
5. Select **Lock & Edit.**
6. Expand **weblogic.**
7. Select the check box against **security** and select **Enable.**
8. Select **Active Changes.**

These changes can be made while the server is running and all additional log messages are written to the server log. This will generate a very large amount of additional log messages. The server log file is configured to wrap the log file at 5MB and leave the additional debug log setting enabled for five minutes. With all the SAS Web applications deployed, four log files are generated.

As with both JBoss and IBM WebSphere, there is a separate option to enable debug log output for the SAS TrustedLoginModule. To enable the debug log output for the SAS TrustedLoginModule:

1. Access the **Administration Console.**
2. Select **Security Realms** and select **myrealm** from the table.
3. Click the **Providers** tab and select the **SASTrustedAuthenticator** from the table.
4. Click the **Provider Specific** tab.
5. Select **Lock & Edit.**
6. Edit the value of **debug** and set it to **true.**
7. Select **Save** and select **Activate Changes.**
8. Restart the application server for the changes to take effect.

The debug option causes messages to be sent to the standard output of the JVM. Hence, the `SASServer1.out` file will contain the debug messages from the SAS TrustedLoginModule. For a successful authentication event, the log will contain lines similar to those shown in Output 3.

To obtain additional logging information when using SPNEGO, it is necessary (as with IBM WebSphere) to add two JVM options to the start-up arguments of the server. Depending on how the server is started will depend where these two options need to be placed. If the server is started via the Administration Console, the options should be added in the following manner:

1. Access the **Administration Console.**
2. Expand **Environment**, and select **Servers.**

3. Select the server name from the table.
4. Click the **Configuration** tab and click the **Server Start** tab.
5. Select **Lock & Edit.**
6. Add the following two arguments to the **Arguments** field:

```
-Dsun.security.krb5.debug=true -Dweblogic.StdoutDebugEnabled=true
```

7. Click Save and select **Activate Changes.**
8. Restart the application server for these changes to take effect.

If the server is started as a Microsoft Windows Service or via scripts, the arguments will need to be added to the common environment script that is located in

`SAS-configuration-directory`**`\Lev1\Web\SASDomain\bin\commEnvSAS.cmd`**

or

`SAS-configuration-directory`**`\lev1\Web\SASDomain\bin\commEnvSAS.sh`**

Within this file, the options `-Dsun.security.krb5.debug=true` and `-Dweblogic.StdoutDebugEnabled=true` should be added to the JAVA_OPTIONS line. Finally, if the server is started as a Microsoft Windows Service, the service should be removed and reinstalled before the changes will take effect.

This additional logging will write messages regarding the following parts of the SPNGEO authentication process:

- Reading the Keytab file
- Transactions with the Kerberos Key Distribution Center
- Cryptography types used for the Kerberos tickets

TESTING AUTHENTICATION

The previous sections have provided detailed instructions for enabling additional logging to provide useful information for troubleshooting authentication issues. Enabling some of the TRACE level of logging can produce large amounts of information very quickly. This information can become unwieldy to use efficiently in diagnosing the authentication issue. Therefore, it is important to have a testing strategy in place before enabling any of the additional logging.

The testing strategy must aim to isolate the authentication process from other parts of the system. This will minimize the additional logging messages which can distract from the troubleshooting process. A simple means of achieving this isolation is to test using only a single application. SAS Technical Support is able to provide a simple testing Web application that can be deployed to any of the supported Java application servers and is independent of the authentication mechanism. Testing using this application will simplify the troubleshooting process.

The testing strategy that we propose is

1. Identify the log files.
2. Make changes to the logging configuration to add more details to each log.
3. Temporarily make the deployed SAS Web applications unavailable.
4. Use the SAS testing application to run a test.
5. Review the log files to isolate error messages.
6. Solve any issues using the proposed solutions later in this paper.

Ideally the testing application should be the only application deployed to the server. This application can be easily configured with JBoss Application Server. Just rename the `deploy_sas` directory and create a new `deploy_sas` directory. Then place the testing application in the new `deploy_sas` directory. However, it should be noted that the Java classes used by the SAS TrustedLoginModule are part of the SAS Logon Manager, and if this is not deployed, the SAS TrustedLoginModule will not be available. A work around for this is to copy the required JAR files from the SAS Logon Manager to the JBoss `\lib` directory. Specifically, the files should be copied from

`JBoss-Home`**`\server\SASServer1\deploy_sas\sas.wip.apps9.2.ear\sas.svcs.logon.war\WEB-INF\lib`**

to

`JBoss-Home\server\SASServer1\lib`

The following files need to be copied:

- `sas.core.jar`
- `sas.oma.omi.jar`
- `sas.security.sspi.jar`
- `sas.svc.connection.jar`
- `sas.svc.sec.login.jar`
- `sas.svc.sec.login.jboss.jar`

With these files copied, the SAS TrustedLoginModule will be available for testing without any of the SAS Web applications deployed.

With IBM WebSphere, it is possible to stop all the deployed SAS Web applications using the IBM Integrated Solutions Console and then deploy the SAS testing application. With the testing application successfully deployed, it is straightforward to enable the diagnostic trace output for the server and test the authentication process. IBM WebSphere will already have access to the Java classes required for the SAS TrustedLoginModule. (This access is a step in the standard configuration of Web authentication with IBM WebSphere.)

A similar process to the one used with IBM WebSphere can be used with Oracle WebLogic. Oracle WebLogic allows the administrator to stop individual applications. However, the additional log settings require you to restart the application server. Therefore, the testing application should be deployed; additional logging options should be enabled; the server started; the SAS Web applications stopped; and then the authentication process tested.

With the testing application from SAS deployed, running the test is performed by accessing the URL for the specific Java application server:

- JBoss: http://<servername>:<port>/sec_con/sec_con_jboss.jsp
- IBM WebSphere: http://<servername>:<port>/sec_con/sec_con_92was.jsp
- Oracle WebLogic: http://<servername>:<port>/sec_con/sec_con_92wls.jsp

A page is returned displaying the Principal values set by the JAAS configuration. Figure 5 shows the result of a successful test on IBM WebSphere with SPNEGO configured.

JAAS Subject Test

Access CallerSubject from WSSubject

```
Actual code:
Subject callersubject = com.ibm.websphere.security.auth.WSSubject.getCallerSubject();

Printing callerSubject....
Subject:
        Principal: sticdemo01@RACE.SAS.COM
        Principal: shqracedc01.race.sas.com:389/sticdemo01
        Principal: PFSPrincipal: sasdemo
        Principal: PFSPrincipal: sticdemo01@WebAuth
        Public Credential: com.ibm.ws.security.auth.WSCredentialImpl@20682068
        Private Credential:

token name:          com.ibm.wsspi.wssecurity.token.krbAuthnToken
version:             1
hashCode:            -1395948001
uniqueId:            sticdemo01
kerberos principal:  sticdemo01
realm:               RACE.SAS.COM
expiration:          Thu Jan 20 07:22:08 EST 2011
renew until:         null
isReadOnly:          false
isAddressless:       false
isForwardable:       false
isRenewable:         false
KerberosTicket:      Kerberos TGT is null.
GSSCredential:       null
        Private Credential: com.ibm.ws.security.token.SingleSignonTokenImpl@25412541
        Private Credential: com.ibm.ws.security.token.AuthenticationTokenImpl@23e523e5
        Private Credential: com.ibm.ws.security.token.AuthorizationTokenImpl@24e724e7
        Private Credential: com.sas.services.security.login.PFSCredential@2c322c32

Printing callerPrincipal....
sticdemo01


-----------------------------------------------
RemoteUser = sticdemo01
-----------------------------------------------
Is user in role "webuserRole"?: no
```

Figure 5: Testing Authentication

SAMPLE LOG MESSAGES AND SOLUTIONS

Error Message	`Error 401--Unauthorized`
Cause	This error in the client browser shows an error with the authorization part of JAAS.
Solution	Confirm that the role mapping is defined with the Java application server. Either the role mapping is not functioning, or the user is not part of the correct groups.

Error Message	`Client Browser: Error 403—Forbidden` `Java App Log: [SasLoginModule] Adding Principal com.sas.services.security.login.weblogic.WLServerPrincipal : unknown.`
Cause	This error in the client browser shows that the end user does not exist in the SAS metadata.
Solution	Add the login for the user in the SAS metadata.

Error Message	`SPNEGO authentication is not supported on this client.`
Cause	This error in the client browser shows that you are trying to access the SAS Web applications deployed to IBM WebSphere with SPNGEO configured using a browser that is not configured for Single Sign-On.
Solution	Update the browser configuration to allow Single Sign-On.

Error Message	`Error 401 - This request requires HTTP authentication ().`
Cause	This error in the client browser shows that you are trying to access the SAS Web applications deployed to JBoss Application Server with SPNGEO configured using a browser that is not configured for Single Sign-On.
Solution	Update the browser configuration to allow Single Sign-On.

Error Message	`ERROR []`
	`com.sas.svcs.authentication.client.AuthenticationServiceInterface - No principal was found to initialize the Information Service connection.`
	`com.sas.services.ServiceException: No principal was found to initialize the Information Service connection.`

Cause	This error in the Java application server points to a failure of the connection to SAS metadata.
Solution	Confirm the user name and password for the SAS Trusted User.

Error Message	`ERROR []`
	`com.sas.svcs.authentication.client.AuthenticationServiceInterface - RemoteException occurred in server thread; nested exception is:`
	`java.rmi.UnmarshalException: error unmarshalling arguments; nested exception is:`
	`java.lang.ClassNotFoundException: org.jboss.security.SimpleGroup (no security manager: RMI class loader disabled)`

Cause	This error in the Java application server log points SAS Remote Services being unable to load required classes.
Solution	Confirm that the class path for SAS Remote Services contains the required for the Java application server.

TROUBLESHOOTING KERBEROS ISSUES

The configuration of SPNEGO for Single Sign-On within the supported Java application servers is relatively straightforward. Therefore, the most common authentication errors with these configurations are with the Kerberos setup. This section of the paper identifies methods to test the Kerberos configuration and some sample Kerberos error messages with associated proposed solutions.

KERBEROS TESTING

The majority of the Kerberos and Microsoft Windows domain configuration can be tested outside the Java application server. The following tests will confirm the configuration prior to testing with the Java application server. From the detailed description of the SPNEGO setup earlier in the paper, one of the key stages in the configuration is the registration of a service principal name (SPN) for the Java application server. This name is required for a service ticket to be granted to the end users. Microsoft Windows has a built-in command which can be used to list the SPNs registered against a user account. Issuing the following command as a domain user will list the SPNs registered:

```
setspn –l <username>
```

Without this registered name, the SPNEGO configuration will fail.

For the Java application server to connect to Kerberos, a credential is supplied for the mapped user in the form of a Kerberos Keytab. The keytab file will contain an entry for the registered service principal name, and this name must match the value returned from the SETSPN command. A Java utility klist can be used to list the entries in a keytab file. Issuing the following command:

```
c:\Program Files\Java\jdk1.5.0_15\bin\klist.exe" -k -t <keytab>
```

will result in the output shown in Output 6.

```
Key tab: C:\SAS9.2\appsrv05.keytab, 1 entry found.

[1] Service principal: HTTP/SGCWIN065.RACE.SAS.COM@RACE.SAS.COM
         KVNO: 12
         Time stamp: Dec 31, 1969 19:00
```

Output 6. Java Utility klist Example Output

Once the service principal name and keytab have been tested, the Kerberos process of obtaining a ticket can also be tested. Again a Java utility kinit can be used to carry out the test. The following command will request a new ticket:

```
"c:\Program Files\Java\jdk1.5.0_15\bin\kinit.exe" -k -t <keytab> <SPN> -J-
Djava.security.krb5.conf= <krb5.conf>
```

If the new ticket is granted, a message will be echoed stating the ticket was added to the cache. This command can include a debug option for more output. Adding the option `-J-Dsun.security.krb5.debug=true` will result in debug messages being displayed for each step in the process. The next section will present some of the common Kerberos error messages returned by this kinit test and by the Java application servers.

SAMPLE KERBEROS ERROR MESSAGES

This section of the paper will present some common Kerberos error messages, their possible causes, and some solutions. These errors might be seen in either the debug logs from the Java application servers or from the results of testing the Kerberos process with the Java utility kinit.

Error Message	`javax.security.auth.login.LoginException: KrbException:: Pre-authentication information was invalid (24) - Preauthentication failed`
Cause 1	The password entered is incorrect.
Solution 1	Verify the password.
Cause 2	If you are using the keytab to get the key, then the key might have changed since you updated the keytab.
Solution 2	Consult your Kerberos documentation to generate a new keytab and use that keytab.
Cause 3	Clock skew - If the time on the KDC and on the client differ significantly (typically 5 minutes), this error can be returned.
Solution 3	Synchronize the clocks (or have a system administrator do so).

Error Message	`GSSException: No valid credentials provided (Mechanism level: Attempt to obtain new INITIATE credentials failed! (null)) . . . Caused by: javax.security.auth.login.LoginException: Clock skew too great`

Cause	Kerberos requires the time on the KDC and on the client to be loosely synchronized. (The default is within 5 minutes.) If that's not the case, you will get this error.
Solution	Synchronize the clocks (or have a system administrator do so).

Error Message	`javax.security.auth.login.LoginException: KrbException: Null realm name (601) - default realm not specified`
Cause	The default realm is not specified in the Kerberos configuration file `krb5.conf` (if used), the realm is not provided as a part of the user name, or the realm is not specified via the `java.security.krb5.realm` system property.
Solution	Verify that your Kerberos configuration file (if used) contains an entry specifying the default realm, specify the realm by setting the value of the `java.security.krb5.realm` system property, or include the realm in your user name when authenticating using Kerberos.

Error Message	`javax.security.auth.login.LoginException: java.net.SocketTimeoutException: Receive timed out`
Cause	Cannot communicate with the KDC.
Solution	Verify that the Kerberos KDC is up and running.

Error Message	`GSSException: No valid credentials provided (Mechanism level: Failed to find any Kerberos Ticket)`
Cause	This error may occur if no valid Kerberos credentials are obtained. In particular, this error occurs if you want the underlying mechanism to obtain credentials, but you forgot to indicate this by setting the `useSubjectCredsOnly` system property value to false (for example, by including `DuseSubjectCredsOnly=false` in your execution command).
Solution	Be sure to set the `useSubjectCredsOnly` system property value to false if you want the underlying mechanism to obtain credentials.

Error Message	`javax.security.auth.login.LoginException: Could not load configuration file <krb5.conf> (No such file or directory)`
Cause	To use a `krb5.conf` file, you either set the `java.security.krb5.conf` system property (instead of the realm and KDC properties) to specify the location of the file, or you don't set any of these properties and therefore an attempt is made to locate the `krb5.conf` file in a default location. You will get the error "`Could not load configuration file <krb5.conf> (No such file or directory)`" if the file could not be found.
Solution	Verify that the Kerberos configuration file `krb5.conf` is available and readable. Check Kerberos Requirements for information about how to specify the location of the `krb5.conf` file and where such a file is searched for if you don't explicitly indicate the location.

Error Message	`javax.security.auth.login.LoginException: KrbException: KDC has no support for encryption type (14) - KDC has no support for encryption type`
Cause	Your KDC does not support the encryption type requested.
Solution	Please choose an encryption type that is supported by the KDC you are using. Microsoft Windows 2008 R2 no longer supports Data Encryption Standard (DES). You will need to use Rivest Cipher 4 (RC4).

Error Message	`KDC reply did not match expectations`
Cause	The KDC sent a response that cannot be understood by the client.

Solution	Verify that you have set correctly all the configuration parameters in the `krb5.conf` file, and consult your KDC vendor's guide.

CONCLUSION

Single Sign-On (SSO) can mean different things to different people. This ranges from SSO from one application to other applications (as represented by the SAS Information Delivery Portal based SSO) to SSO from the desktop to any application (as represented by the Windows Domain based SSO). Across all types of SSO, the configurations of enterprise-level Web infrastructures are complex and involve a number of third-party components. Troubleshooting authentication issues is complicated by the involvement of these additional components. The SAS 9.2 dependence on JAAS simplifies the SAS configuration part.

Troubleshooting authentication issues in these enterprise-level configurations needs to be undertaken as a step-by-step logical process. The other SAS Web applications should be temporarily undeployed to simplify the log messages which need to be reviewed. The SAS testing utility should be deployed and used to provide a simple and straightforward test. With the testing application deployed, you should review the additional logging to ascertain where the issues are occurring. Finally, with the number of sample error messages in this paper you should be able to resolve your authentication issues.

One point to note is Windows Domain based SSO authentication issues are mostly with the Kerberos part and not the other parts of the configuration. Therefore, with these types of enterprise-level configurations, it is more efficient to test the Kerberos part before troubleshooting the Java application server configuration.

If you have any doubts when troubleshooting authentication issues, contact SAS Technical Support. At a minimum, this paper will have made you aware of the type of logging information that SAS Technical Support will require in order to assist you. When communicating with SAS Technical Support, consider providing them a diagram like those in this paper to help explain your configuration.

REFERENCES

- Internet Engineering Task Force (2005). “The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism.” Available at http://tools.ietf.org/html/rfc4178.
- Microsoft Corporation. (2010). “Windows Authentication.” TechNet. Available at http://technet.microsoft.com/en-us/library/cc755284%28WS.10%29.aspx.Accessed on January 28, 2011.
- Microsoft Corporation. (2005). “Troubleshooting Kerberos.” TechNet. Available at http://technet.microsoft.com/en-us/library/cc728430%28WS.10%29.aspx. Accessed on January 28, 2011.
- Microsoft Corporation. 2004. Microsoft Corporation white paper. “Troubleshooting Kerberos Errors.” http://go.microsoft.com/fwlink/?LinkID=93730.
- Massechuttes Institute of Technology. (2010). “Kerberos: The Network Authentication Protocol.” Kerberos. Available at http://web.mit.edu/Kerberos/. Accessed on January 28, 2011.
- Oracle Corporation (2010.) “Java Authentication and Authorization Service (JAAS) Reference Guide.” Available at http://download.oracle.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html.
- IBM Corporation. (2010). “Creating a single sign-on for HTTP requests using the SPNEGO TAI.” Available at http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/tsec_SPNEGO_tai.html.
- IBM Corporation. (2010). “Single sign-on for HTTP requests using SPNEGO web authentication.” Available at http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/csec_SPNEGO_explain.html.
- IBM Corporation. 2009. IBM Corporation white paper. “WebSphere with a side of SPNEGO.” http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101065.
- Oracle Corporation. (2008). “Configuring Single Sign-On with Microsoft Clients.” Available at http://download.oracle.com/docs/cd/E12840_01/wls/docs103/secmanage/sso.html.
- SAS Institute Inc. (2010). “Configuring IBM WebSphere Application Server 6.1 for Web Authentication with SAS 9.2 Web Applications.” Available at http://support.sas.com/resources/thirdpartysupport/v92/appservers/ConfiguringWASWebAuth.pdf

- SAS Institute Inc. (2010). "Configuring IBM WebSphere Application Server 7.0 for Web Authentication with SAS 9.2 Web Applications." Available at http://support.sas.com/resources/thirdpartysupport/v92m3/appservers/ConfiguringWAS7WebAuth.pdf.
- SAS Institute Inc. (2010). "Configuring Integrated Windows Authentication for IBM WebSphere with SAS 9.2 Web Applications." Available at http://support.sas.com/resources/thirdpartysupport/v92m3/appservers/IWAWebSphere.pdf.
- SAS Institute Inc. (2010). "Configuring BEA WebLogic Server 9.2 for Web Authentication with SAS 9.2 Web Applications." Available at http://support.sas.com/resources/thirdpartysupport/v92m3/appservers/ConfiguringWLSWebAuth.pdf
- SAS Institute Inc. (2010). "Configuring Integrated Windows Authentication for WebLogic with SAS 9.2 Web Applications." Available at http://support.sas.com/resources/thirdpartysupport/v92m3/appservers/IWAWebLogic.pdf.
- SAS Institute Inc. (2009), "Configuring JBoss Application Server 4.2.0 for Web Authentication with SAS 9.2 Web Applications," Available at http://support.sas.com/resources/thirdpartysupport/v92m3/appservers/ConfiguringJBossWebAuth.pdf.
- SAS Institute Inc. (2010), "Configuring Integrated Windows Authentication for JBoss with SAS 9.2 Web Applications." Available at http://support.sas.com/resources/thirdpartysupport/v92m3/appservers/IWAJBoss.pdf.
- JBoss Community. "User Guide for JBoss Negotiation (inside of Negotiation-2.0.3.GA module)." Available at https://jira.jboss.org/jira/browse/SECURITY-343.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Stuart Rogers
SAS Institute Inc.
SAS Campus Drive
E-mail: stuart.rogers@sas.com

Heesun Park
SAS Institute Inc.
SAS Campus Drive
E-mail: sashsp@sas.com