Paper SAS102-2014

An Advanced Fallback Authentication Framework for SAS® 9.4 and SAS® Visual Analytics

Zhiyong Li, SAS Institute; Mike Roda, SAS Institute

ABSTRACT

SAS 9.4 and SAS Visual Analytics (VA) support a wide list of authentication protocols such as Integrated Windows Authentication (IWA), client certificate, IBM WebSEAL, CA SiteMinder and SAML 2.0. However, advanced customers may want to be able to use some of these protocols together and also have the flexibility to select which protocols to use. In this paper, we will focus on a "fallback" authentication framework that supports IWA as the primary authentication method. When IWA fails, it uses the X509 client certificate as the secondary authentication method, and when the client certificate fails, it uses the form-based username/password as the last option. The paper will first introduce the security architecture of SAS 9.4 and SAS VA. It will then review the three mentioned security protocols. Further, it will introduce the detailed "fallback" authentication framework and discuss how to configure it. Finally, we will discuss the use of this framework in the customer scenario from implementing the "fallback" authentication framework in a customer's SAS 9.4 and SAS VA environment.

INTRODUCTION

SAS 9.4 has changed its mid-tier infrastructure from using WebLogic, WebSphere and JBoss to SAS Web Application Server. In addition, SAS 9.4 also pre-packages a SAS Web Server and the management component called SAS Environment Manager. With this significant infrastructure change comes with the following security architecture and implementation challenges:

- 9.4 SAS Web Application Server environment should support all third party security products and protocols that are supported in SAS 9.3 by three Web Application Servers. Table 1 shows the security protocols that are supported in 9.3 for different Web Application Servers and the security protocols supported by SAS 9.4.

Security Protocols/Products	SAS 9.4 SAS Web (Application) Servers	SAS 9.3		
		WebSphere	WebLogic	JBoss
Host Authentication	X	X	X	X
Container Managed Security	X	X	X	X
SSL	X	X	X	X
FIPs compliance	X	X	X	X
Browser based IWA	X	X	X	X
Desktop client based IWA	X	X	X	X
Browser based IWA and pass credential to the server	X			
Desktop client IWA and pass credential to the server	X			X
CA SiteMinder	X	X	X	
IBM WebSeal	X	X		
SAML 2.0	X			
Fallback authentication	X			

Table 1. Authentication protocol / solutions supported by SAS 9.3 and SAS 9.4

- SAS 9.4 security architecture should consider both SAS Web Server and SAS Web Application Server.
- SAS 9.4 security architecture should support customers' advanced security requirements.

SAS 9.4 has also changed its single sign-on implementation or Logon Manager to use Central Authentication Server (CAS). This will require changes from the previous SAS releases on how we can support third party security products with the Logon Manager.

This paper will give you an overview of SAS 9.4 web application security architecture, which includes a detailed discussion of the above two-layers of authentication: SAS web application layer and the enterprise layer.

Further, some customers may have advanced requirements. They want to be able to use several authentication products or protocols in one authentication flow. In other words, they want to be able to authenticate using protocol A, when that authentication fails, they want to be able to continue to authenticate using protocol B, and so on. One typical example is using three authentication protocols: IWA, client certificate and form-based username/password. We have developed such a framework that can support this requirement. We will use the above three protocols as an example to demonstrate the use of the framework.

SAS 9.4 AND SECURITY ARCHITECTURE

SAS 9.4 ARCHITECTURE

To understand the security, we need to first understand SAS 9.4 architecture. SAS 9.4 enterprise platform employs a multi-tier architecture. Figure 1 shows the high-level of that architecture for the SAS 9.4 enterprise platform. The client tier consists of browser-based, Java desktop or MS office based applications. The middle tier has web applications that run in the SAS Web Application Server, which in turn is behind the SAS Web Server. SAS Web Server can be used to provide the load-balance for the multiple clustered Web Application Server instances. It can also be used as the reverse-proxy server when SAS web applications are deployed into multiple Web Application Server instances. The server tier consists of SAS application servers such as SAS metadata server, SAS workspace servers and SAS stored process servers. The data tier has SAS Web Infrastructure Platform (WIP) data server and / or third party database servers.

SAS Web Application Server is SAS OEM-ed Pivotal tc Server. Pivotal tc Server is based on Apache Tomcat. Thus, in the following discussion, when we mention that a function is available for Tomcat, you can safely assume that same functionality is also valid for the SAS Web Application Server.

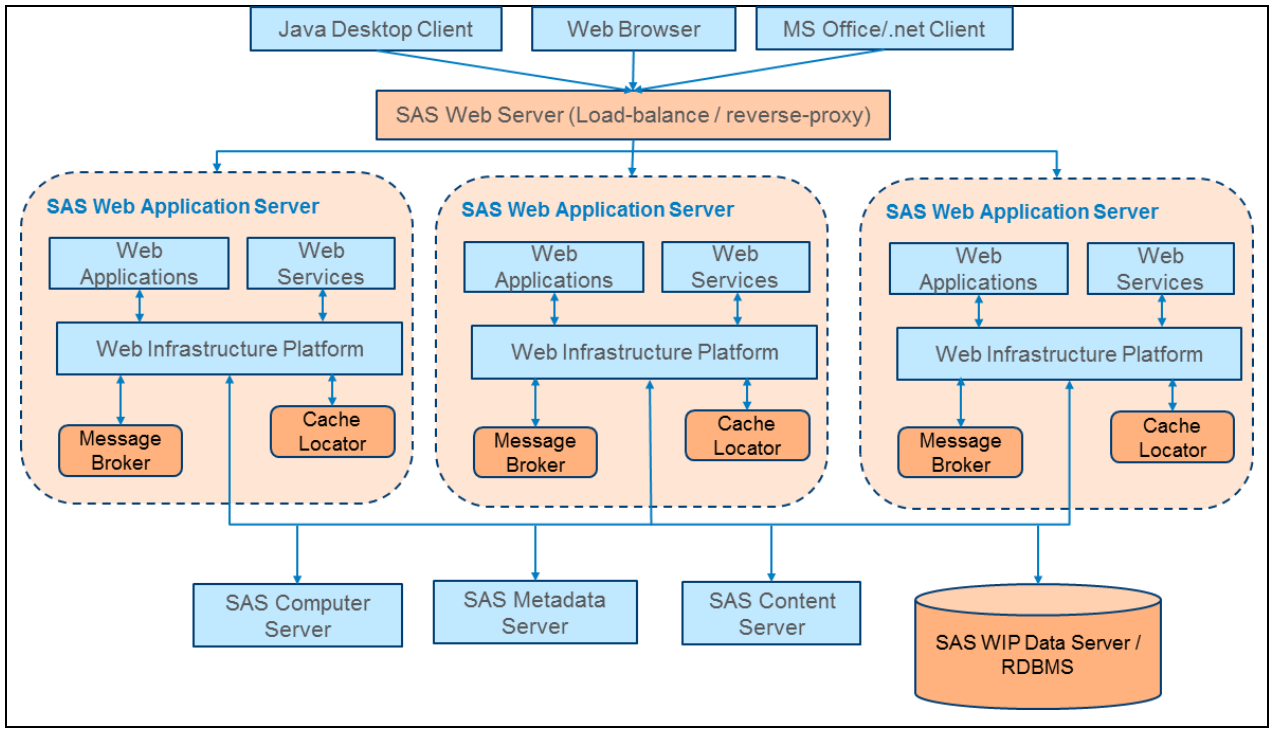

Figure 1. High-Level Architecture for SAS 9.4 Enterprise Application

SAS 9.4 AND VA SECURITY ARCHITECTURE

SAS 9.4, VA and Solutions include over eighty web applications. We provide a single logon manager that only requires an end-user sign-on to SAS web applications once and that user can access all other SAS web applications without being challenged to logon again. On top of that single sign-on to all SAS web applications, we also allow SAS web applications to participate with the enterprise authentication, which may have already been implemented by customers. This integrated authentication or single sign-on to customer environments may be implemented using container manager security and/or using customer's third party single sign-on products. In summary, SAS 9.4 offers the following layers of single sign-on authentication options:

- SAS web applications single sign-on: Use Logon Manager and CAS to single sign-on to all SAS web applications.
- Enterprise security Integration: Single sign-on to container's realm or third party authentication domains.
  - Use container managed security
  - Integrate with enterprise Single Sign-on (SSO) solutions such as IWA, CA SiteMinder, IBM WebSEAL, SAML, etc.

Figure 2 shows the high-level authentication architecture for SAS 9.4 web Applications. In that diagram, there are multiple web application server instances; however, Logon Manager is only on SASServer1_1 (one per cluster node). All SAS web applications will use that same Logon Manager to sign-on. When using container managed security, container managed security will be implemented at the SASServer1_1 instance that SAS Logon Manager is running on. When integrating with third party single sign-on products, the authentication will be again enforced at the SASServer1_1 instance that SAS Logon Manager is running on. Integration with the third party single sign-on products may use SAS Web Application Server only or use both SAS Web Server and SAS Web Application Servers. For example, to support CA SiteMinder, we need to have the Agent for SAS Web Server and the Agent for SAS Web Application Servers.

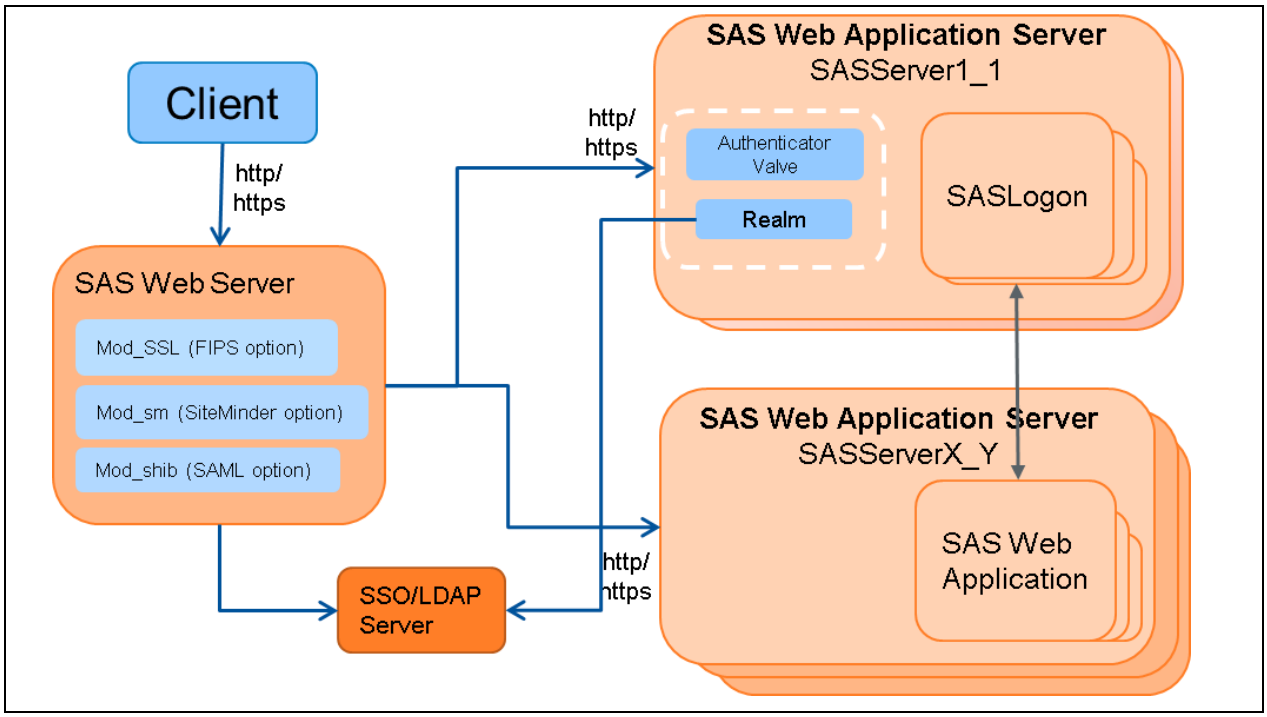

Figure 2. High-Level Security Architecture for SAS 9.4 Web Applications

SAS VA Web Application uses the same Logon Manager as the one used by other SAS web applications so that it can participate the single sign-on with other SAS Web Applications.

SAS Logon Manager and CAS

The SAS Logon Manager is a web application that handles authentication requests for SAS web applications. It is the (final) entry point for all authentication requests. As a result, users see the same logon page when they access the SAS web applications and will only need to logon once. By default, SAS Logon Manager uses host based authentication.

Logon Manager is implemented by using CAS. CAS is an open source authentication system originally created by Yale University to provide a trusted way for an application to authenticate a user. CAS can provide Single Sign-On to applications. CAS uses tickets and hence reduces the need for applications to pass credentials in order to authenticate a user. The heart of CAS is the CAS server, a Java servlet built on the Spring Framework. The SAS Web Infrastructure Platform packages its CAS server in sas.svcs.logon.war. Servlet mappings such as "/login" direct appropriate URLs to the CAS server. For example, one can logon using:

http://xyz.sas.com/SASLogon/login?service=http://xyz.sas.com:8080/SASHelloWorld/j_spring_cas_security_check

SAS applications engage with the CAS server by configuring the Spring SecurityFilterChain in their web.xml files, by employing additional configuration in their Spring application context as needed, and by using REST APIs. A SAS Spring namespace tag, <sas-sso:http />, adds filters to the SecurityFilterChain including a CasAuthenticationFilter.

Figure 3 shows the authentication process using the CAS based Logon Manager and how the tickets flow when a user logon to a SAS web application. There are four important actors in this scenario: a browser, a SAS web application (SASHelloWorld), the SASLogon application, and the CAS server.

1. The user types a URL such as http://localhost/SASHelloWorld into the browser.
2. Security processing in the Spring SecurityFilterChain for SASHelloWorld sends a response to the browser redirecting the request to SASLogon.
3. SASLogon's security processing communicates with the CAS server.
4. There is no authentication yet so SASLogon presents the SAS logon window in which the user enters credentials.
5. SASLogon uses the credentials to request a CAS Service ticket (ST) required for login and responds to the browser with that ticket.
6. The browser resends the initial login request to SASHelloWorld with the Service ticket.
7. This time the SASHelloWorld SecurityFilterChain sees the ticket, validates it with the CAS server, and login proceeds.

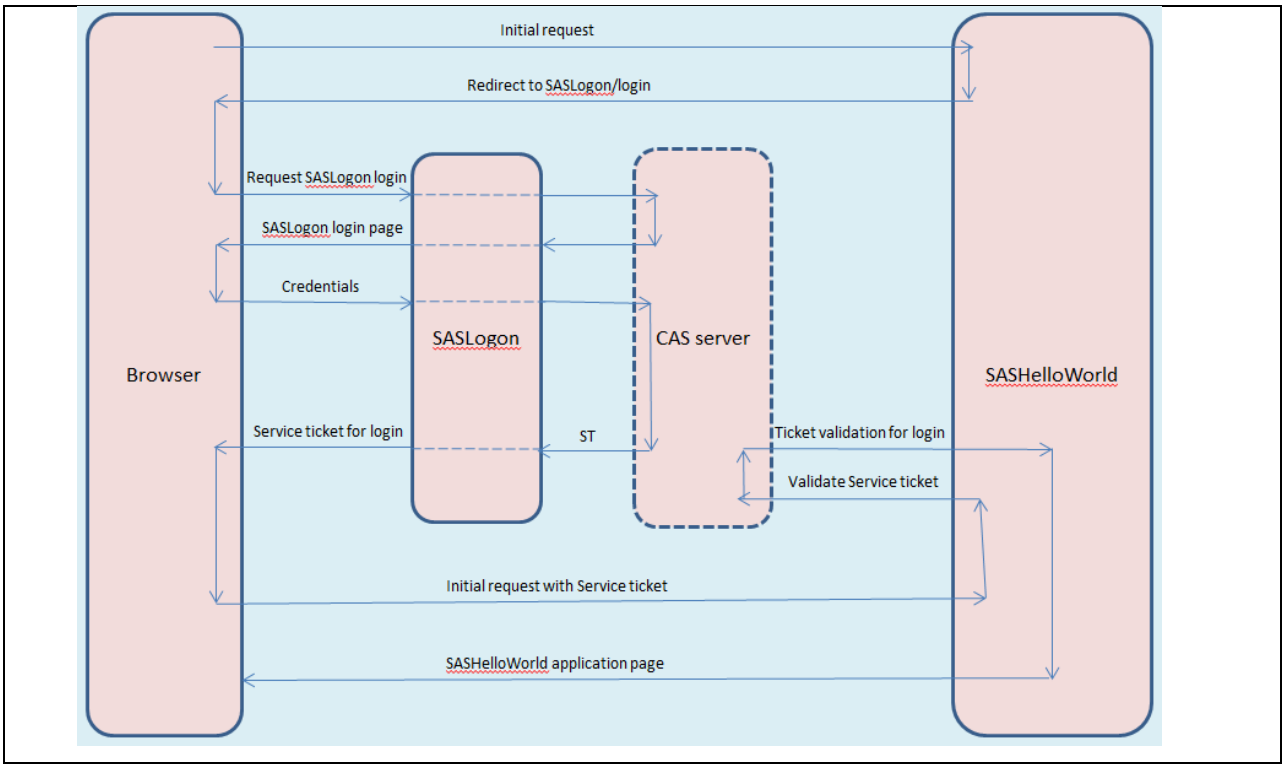

Figure 3. CAS and SAS Logon Manager

Container Managed Security

As we have mentioned, by default, SAS web applications use the form-based authentication that is provided by the SAS Logon Manager application. When credentials are provided to SAS Logon Manager, the credentials are sent to the SAS Metadata Server for authentication. The metadata server then authenticates the credentials against its authentication provider. The default provider is the host operating system.

As an alternative, you can configure the SAS web applications to authenticate on the middle tier. When users logon to a SAS web application, SAS Web Application Server handles the initial authentication for container-managed security. To use container managed security, we will need to make the following configuration changes:

- Configure CAS for Trusted Web Authentication: This will make CAS trust the container or third party authenticator for authentication.
- Secure SAS Logon Manager web application: This will make sure to allow the container to conduct the authentication when authentication request is sent to the SAS Logon Manager web application. To accomplish this, we need (1) set security constraint on /login for SAS Logon Manager web application; (2) select a Login method to authenticate and also make sure credentials or tokens can be made available, the Login methods could be a third party Login that negotiates with the user agent and collects credentials (this approach will be further discussed in the following Enterprise SSO Integration subsection) or the Web Application Server built-in support for BASIC, FORM, DIGEST, SPNEGO, and CLIENT-CERT methods; (3) define the security roles so that the user can be authorized to access the web application, for example, we can use * in conjunction with allRolesMode="authOnly“ on Realm.
- Realms: Realm is where the user is stored (also called the user registry) and the authentication is really happening. It is also there that the security role is checked.

Enterprise SSO Integration

As mentioned in the previous subsection, when using container managed security, the container may further be configured to use the third party Login. SAS 9.4 supports a wide variety of third party authentication protocols and products. Several popular ones are briefly discussed below.

Integrated Windows Authentication

Integrated Windows Authentication (IWA) is a Microsoft technology that is used in an environment where users have Windows domain accounts. With IWA, users will not be prompted for credentials; instead it will use the credentials associated with the user's Windows Login account. IWA typically uses Simple and Protected GSSAPI Negotiation Mechanism (SPNEGO) or Kerberos authentication protocols. On a typical environment, the key components of IWA are an Active Directory Controller machine (Windows 2000 Server or higher), a Kerberos Key Distribution Center (KDC) in a Domain Controller machine, a machine with a client browser or standalone client application, and a Web Application Server.

SAS 9.4 supports IWA in four different scenarios:

- Middle tier IWA: The IWA authentication happens between client and the middle tier. The middle tier and the server tier communication do not use the client IWA credential. Based on the type of the client application, this can be categorized into the following two types:
  - Client application is a web application.
  - Client application is a standalone Java application.
- Middle tier IWA and passing the client credential to the server tier: The IWA authentication happens not only between client and the middle tier, but also between the middle tier and the server tier. The middle tier will use the client delegation token to authenticate the client against the server tier. Similarly, based on the type of the client application, this can be categorized into the following two types:
  - Client application is a web application.
  - Client application is a standalone Java application.

CA SiteMinder

CA SiteMinder is used widely as the user authentication and single sign-on solution by many customers. It also provides a policy-based authorization, identity federation, and auditing to the protected web applications.

SAS 9.4 supports CA SiteMinder by using both SAS Web Server and SAS Web Application Server. It requires configuring a Web Agent to communicate with SAS Web Server and a custom Apache Tomcat valve for SAS Web Application Server. SAS provides this custom valve. Successful authentication results in a security token (SMSESSION) being set in the user's web browser cookies. The valve is part of the request pipeline. It receives the security token in the request and communicates with the policy servers through an API to decode the user credentials from the security token. This works in conjunction with web authentication to integrate with existing CA SiteMinder single sign-on environments.

IBM WebSEAL

IBM WebSEAL is an important component of IBM Tivoli Access Manager. Access Manager provides complete authorization, network security and policy management functions and it is used for end-to-end protection of resources over intranets and extranets. WebSEAL is a high performance, multi-threaded Web server that applies fine-grained security policy to resources in the Tivoli Access Manager protected Web object space. WebSEAL can provide single sign-on solutions and incorporate back-end Web application server resources into its security policy.

IBM Tivoli provides a WebSEAL solution to achieve SSO between Tivoli Access Manager and Apache Tomcat. The integration consists of a light weight identity assertion module. This integration passes the Tivoli Access Manager WebSEAL's authenticated user identity to the Apache Tomcat server, where the identity is established within Apache Tomcat using a custom Valve. A Valve is a filter that is inserted into the request processing pipeline. The custom Valve supplied with this integration creates a Principal for the user and adds roles according to the Valve's configuration.

The integration uses WebSEAL as a reverse-proxy. A WebSEAL junction is created and connected to Web application(s) deployed on the web application server. Users access and authenticate to WebSEAL, and their authenticated user ID is passed through the junction. A Valve is configured on Tomcat to read the user ID and create a Principal, which is used by the Web application in order to achieve SSO.

SAS uses this WebSEAL valve in its SAS Web Application Server to support WebSEAL. Customers need to first follow the steps in the "Web Authentication" procedure to configure SAS Web Application Server and SAS Logon Manager; it then needs to make the following further updates: (1) specify AMTomcatAuthenticated for the role-name element in the web.xml.orig file; and (2) do not add users to tomcat-users.xml or configure a Realm in the server.xml file.

SAML

In SAML 2.0 there is a conceptual distinction made between the roll of Identity Provider, or IdP, and Service Provider, or SP. Conceptually, the SAS Logon Manager acts as an IdP and the applications such as SAS Visual Analytics are service providers. In a SAML 2.0 environment, there is some external IdP, hosted by Salesforce.com for example, and the SAS Logon Manager is also a service provider, providing authentication services to the SAS web applications.

SAML 2.0 IdPs make 'assertions', for example asserting the identity of a user. The assertions, which are sent in XML and contain a digital signature for verification, piggyback on the HTTP requests made by the web browser and are consumed by the service providers. The different mechanisms of doing this are referred to as 'bindings', the two most common being redirect and POST bindings.

SAS 9.4 Maintenance release 1 can be used with SAML 2.0 authentication. Rather than work with the SAML assertions directly, SAS has built a solution around the open source Shibboleth Native SP module, which must be downloaded independently by the customer. The Shibboleth SP module does authentication in the Apache-based SAS Web Server. The authenticated user information is rewritten into the HTTP headers and transmitted to the SAS Web Application Server. Inside the application server, the SAS gets those credentials and trusts them. Technically this could be used with any form of Apache authentication; SAS is not tied in any way to Shibboleth but that is the only one that has been tested.

AUTHENTICATION METHODS FOR FALLBACK AUTHENTICATION

Before we discuss the fallback authentication framework, let us discuss three individual authentication methods that are used in the fallback authentication. For the IWA, we briefly mentioned it in the previous section. It will be discussed here again but we will focus on how it can be configured with the SAS Web Application Server and with the use of browser as the client.

INTEGRATED WINDOWS AUTHENTICATION

Integrated Windows Authentication (IWA) allows users to transparently login to applications using their Windows credentials without having to retype their username and password. In Internet Explorer this can be enabled by going into the Internet Options, Advanced, and checking the "Enable Integrated Windows Authentication" box under the Security settings. In Firefox you must enter about:config in the address bar, search for network.negotiate-auth.trusted-uris and add the address or domain of the middle tier.

For the SAS middle tier this happens over HTTP or HTTPS using the Simple and Protected GSSAPI Negotiation Mechanism (SPNEGO) and requires Kerberos v5 tickets. The SAS 9.4 middle tier application server runs VMware

vFabric tc Server, which is Apache Tomcat. Apache Tomcat has built-in support for SPNEGO. Setup requires a service account within Active Directory, a registered Service Principal Name, and a Kerberos keytab file containing the credentials of the service account. Configuration for IWA in the middle tier is covered in the SAS 9.4 Intelligence Platform: Middle-Tier Administration Guide. A link is provided at the bottom of this paper in the references.

When configured for IWA, normally the SAS Web Application Server is configured to protect only `/SASLogon/login`. When a user visits a SAS web application such as Visual Analytics and a session has not been established yet, the web browser is redirected to SASLogon for authentication. Upon the first request from the web browser for `/SASLogon/login`, a 401 status code (Unauthorized) is returned in the response, along with a challenge in the HTTP headers to negotiate:

```
HTTP/1.1 401 Unauthorized
Content-Type: text/html
Server: Microsoft-IIS/7.5
WWW-Authenticate: Negotiate
Date: Wed, 18 Dec 2013 19:29:57 GMT
Content-Length: 1293
```

If the web browser has been enabled for Integrated Windows Authentication, it will use the Security Support Provider Interface on Windows to contact the domain controller and get a Kerberos ticket for the middle tier service. Once it has a ticket, it makes a second request to the server with the ticket encoded in the Authorization header (truncated in the example below).

```
GET /SASLogon/login HTTP/1.1
Host: rdcesx14138.race.sas.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:25.0) Gecko/20100101
Firefox/25.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Authorization: Negotiate YIIQjQY…
```

The server validates the ticket and sets the username from the Kerberos principal. This is in form <user>@<realm> rather than the earlier Windows <domain>\<user> format. Apache Tomcat strips off the Kerberos realm by default but this behavior can be changed by setting the `stripRealmForGss="false"` option on the Realm.

CLIENT CERTIFICATE AUTHENTICATION

Client Certificate Authentication is another technology which allows users to login without entering a username and password each time. In the case of web browser clients, one or more certificates proving the users' identity is loaded into the web browser and presented when requested by the server. Like a server certificate, the client certificate is issued by a Certification Authority (CA), which digitally signs it with its own well-known certificate, effectively binding a public key to a distinguished name that the CA has verified in some fashion.

The server is configured with the CA certificate(s) that are to be trusted for issuing the client certificates. For the SAS 9.4 middle tier, this involves adding the CA certificate(s) to the SAS Private Java Runtime Environment `cacerts` truststore, and configuring SAS Web Server to point to the file or directory containing the CA certificates.

Certificates between the web browser and server are exchanged during the SSL handshake.

SAS Web Server Configured for HTTPS

SAS Web Server is an Apache HTTP server. As a prerequisite to client certificate authentication, SAS Web Server must be configured for HTTPS. This can be accomplished from the SAS Deployment Wizard, or it can be configured manually after the deployment. Then the following steps are taken to enable client certificate authentication:

1. Configure SAS Web Server for HTTPS (if not already done)
2. Configure Apache to request optional client certificates on /SASLogon/login and specify the location of the Certification Authority certificate(s)
3. Add Apache Rewrite rules to pass the certificate and verification information in the HTTP headers
4. Configure SAS Logon Manager for web authentication, except do not add the security constraint or roles to the web.xml in SASLogon
5. Add the SAS SSLAuthenticator Valve to the SASLogon context
6. Import the Certification Authority certificate(s) into the SAS Private JRE cacerts truststore.

The web server is configured to request client certificates only on `/SASLogon/login` so as not to degrade overall performance. If the client provides a certificate, Apache will verify it using the specified Certification Authority certificate(s). The client certificate and verification information are rewritten into the HTTP headers passed to the web application server.

The web application server is configured with the SAS SSLAuthenticator Valve. The Valve pulls the certificate information from the HTTP headers, making sure it was validated by the web server, and sets it in the request so it can be authenticated by the Realm in the same way as if the handshake occurred directly between the client and application server.

Since the SSL handshake between the client and the web server is the only way to verify the certificate came from the party with the corresponding private key, it is important to ensure clients cannot bypass the web server and send whatever certificate they want to the web application server via HTTP headers. Therefore it is recommended that 2-way SSL be configured between the SAS Web Server and SAS Web Application Server.

Standalone SAS Web Application Server

Deployments that elect not to run a web server may configure Client Certificate Authentication though built-in container-managed security. This involves the following steps:

1. Configure SAS Web Application Server to use HTTPS
2. Configure SAS Logon Manager for web authentication and container-managed security with authentication method CLIENT-CERT
3. Import the Certification Authority certificate(s) into the SAS Private JRE cacerts truststore.

Authentication Realm

With or without the web server in front, SAS Web Application Server must be configured with a Realm to authenticate the username pulled from the client certificate. Any of the standard Apache Tomcat Realms may be used, or as a simpler solution, SAS also provides a TrustedX509CertificateRealm to authenticate any trusted certificate that has not expired. By default, the standard Apache Tomcat Realms get the username from the Distinguished Name (DN) in the certificate, with SAS providing extensions to use the Common Name (CN) or thumbprint instead.

These configurations are documented in the SAS 9.4 Intelligence Platform: Middle-Tier Administration Guide, Second Edition.

FORM-BASED AUTHENTICATION

Perhaps the most common method, we cover two types of form-based authentication here. The first is the standard SAS Logon form that you get out of the box with any SAS middle tier deployment. Behind the SAS Logon form is CAS software from Jasig, and specifically CAS Authentication handlers. By default, SAS Logon is configured with a special authentication handler that authenticates users' credentials against Metadata. When we configure web authentication, for example with IWA or client certificate authentication, we add an authentication handler that trusts the principal set in the request. Other standard CAS authentication handlers are available from Jasig, but typically when we integrate with the Enterprise we will make use of container features and configure CAS for web authentication.

Using container managed security, we begin by editing the web.xml file of SASLogon. A security constraint is added to `/login` and in the login configuration we specify `FORM` as the authentication method. This type of login configuration allows us to specify the login page to use for collecting the users' credentials and an error page when authentication is unsuccessful.

Coupled with the security constraint, we define one or more Apache Tomcat Realms to authenticate the credentials entered on the form. This is usually where the integration with the enterprise security happens. For example we can use a `JNDIRealm` to access an external LDAP, or a `JDBCRealm` to access credentials stored in a database.

FALLBACK AUTHENTICATION

Fallback authentication is a hybrid approach that allows a site to use a primary method for authentication while transparently reverting to one or more alternate methods when the browser does not support the primary method.

There are three key components that enable this to work:

1. Apache Tomcat Authenticators with built-in support for SPNEGO, CLIENT-CERT, etc.
2. A new SasFallbackAuthenticationValve in SAS 9.4
3. A custom error page for 401 status codes

Authenticators are actually special instances of valves that are normally instantiated by the container as a result of placing a security constraint in a web application's `web.xml` file. They choreograph the authentication process, collecting credentials from the client through challenges in the headers, SSL handshaking, or other means, and calling the Realm to authenticate the credentials. One major limitation, however, is there can only be one authentication method specified in the web.xml login configuration.

The `SasFallbackAuthenticationValve` addresses this limitation. Designed to be used with any of the authentication methods supported by Apache Tomcat, the `SasFallbackAuthenticationValve` creates an instance of the Authenticator (e.g. `org.apache.catalina.authenticator.SpnegoAuthenticator`) during startup. As a Valve, it sits in the request processing pipeline for SASLogon. As requests come into the server, it decides whether to pass them to the Authenticator or let them fall through. This is controlled by a query string parameter in the request. Refer to Figure 4 below.

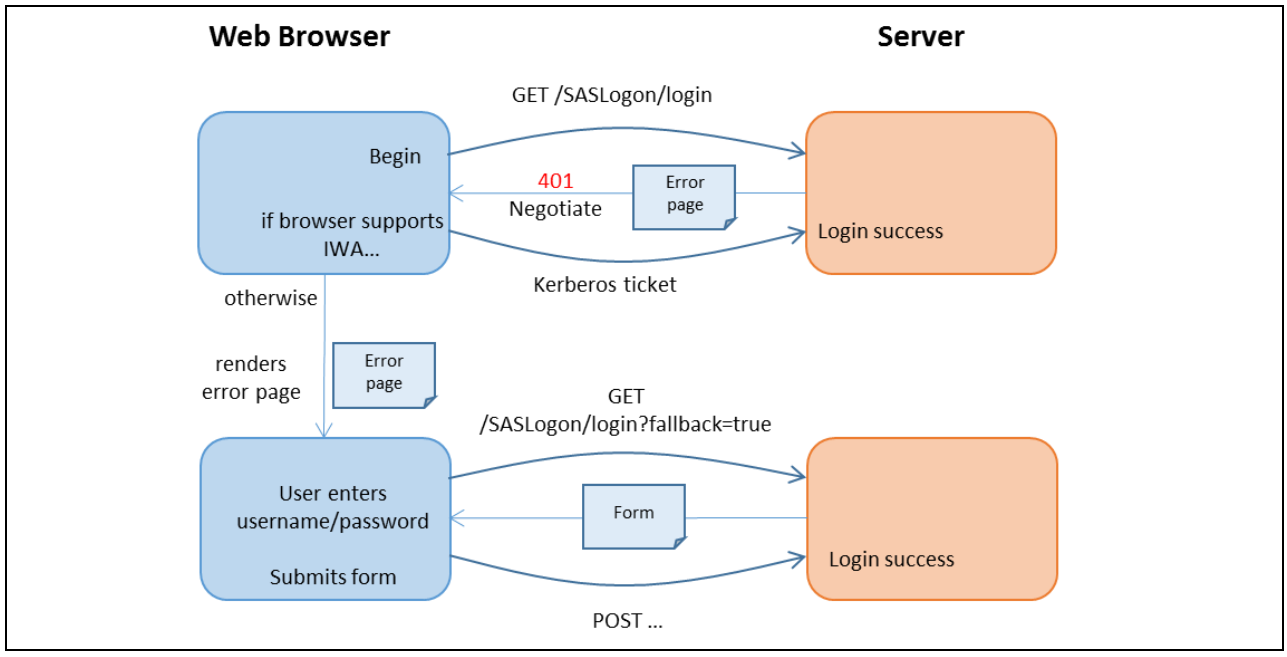

Figure 4. IWA with Fallback to Form-Based Authentication

To facilitate the fallback authentication, a custom error page for 401 status codes is setup in the `web.xml` file for SASLogon. Upon receiving the first request from the web browser to `/SASLogon/login`, a 401 (Unauthorized)

response is returned with a challenge for the primary authentication method. The custom error page is delivered in the content part of the response. If the web browser can handle the challenge, it ignores the error page and sends another request with the appropriate credentials and is authenticated.

If the browser does not support the authentication method specified in the challenge, it displays the error page. Javascript on the page does an immediate redirect back to the login page again, but this time adds a special `fallback=true` parameter on the query string. When the `SasFallbackAuthenticationValve` sees this parameter in the request, it skips the primary authentication method and calls the next valve in the request processing pipeline. Ultimately if the user is not authenticated and no other security is configured, the standard SASLogon form is displayed and the user can login there and be authenticated by Metadata.

To help illustrate how this works better, let's look at some specific examples.

IWA WITH FALLBACK TO THE SAS LOGON FORM

This is the most common use case. IWA is used as the primary authentication method, with the standard SAS Logon form as a backup. This allows, for example, users to authenticate with IWA using a suitable browser, or with a SAS internal account such as sasadm@saspw using a browser that is not setup for IWA. This extra flexibility is trivial to setup, making it an attractive option. In addition to configuring IWA, which is covered in the SAS 9.4 Middle Tier Administration Guide, the following steps are performed.

1. The following is added to `SASServer1_1/conf/Catalina/localhost/SASLogon.xml` inside the Context:

```
<Valve className="com.sas.vfabrictcsvr.authenticator.SasFallbackAuthenticatorValve"
       authMethod="SPNEGO" />
```

2. The custom error page for status code 401 must be uncommented in the `SASServer1_1/sas_webapps/sas.svcs.logon.war/WEB-INF/web.xml` file:

```
<error-page>
    <error-code>401</error-code>
    <location>/WEB-INF/view/jsp/default/ui/401Fallback.html</location>
</error-page>
```

   If a `security-constraint`, `login-config`, or `security-role` was added for web authentication, this should be removed.

The detailed flows are shown below.

Browser supports IWA

1. Client gets redirected to /SASLogon/login from some SAS web application
2. Server responds with 401 status code, header WWW-Authenticate: Negotiate, and error page
3. Client sends GET request for /SASLogon/login with Authorization header containing Kerberos ticket
4. Server authenticates the Kerberos ticket, sets a principal in the request
5. CAS gets the principal in the request and trusts it, login is successful, client is redirected back to the calling application

Browser does not support IWA

1. Client gets redirected to /SASLogon/login from some SAS web application
2. Server responds with 401 status code, header WWW-Authenticate: Negotiate, and error page
3. Client does not support IWA so it displays the error page instead
4. Error page automatically redirects back to the login page but with fallback=true
5. Client sends GET request for /SASLogon/login with Authorization header containing Kerberos ticket
6. Server skips IWA authentication
7. No principal is set in the request so CAS responds with the login form
8. Client sends POST on the login form with username and password
9. CAS authenticates username and password against Metadata, login is successful, client is redirected back to the calling application

WITH CLIENT CERTIFICATE AUTHENTICATION

The previous example can be extended to perform Client Certificate Authentication as a secondary choice. Refer to Figure 5 below. In the second series of interactions, the server requests the client provide an optional certificate during the SSL handshake. If the browser provides a certificate, it is authenticated and login is successful. If not, the server returns a form for the user to enter credentials.

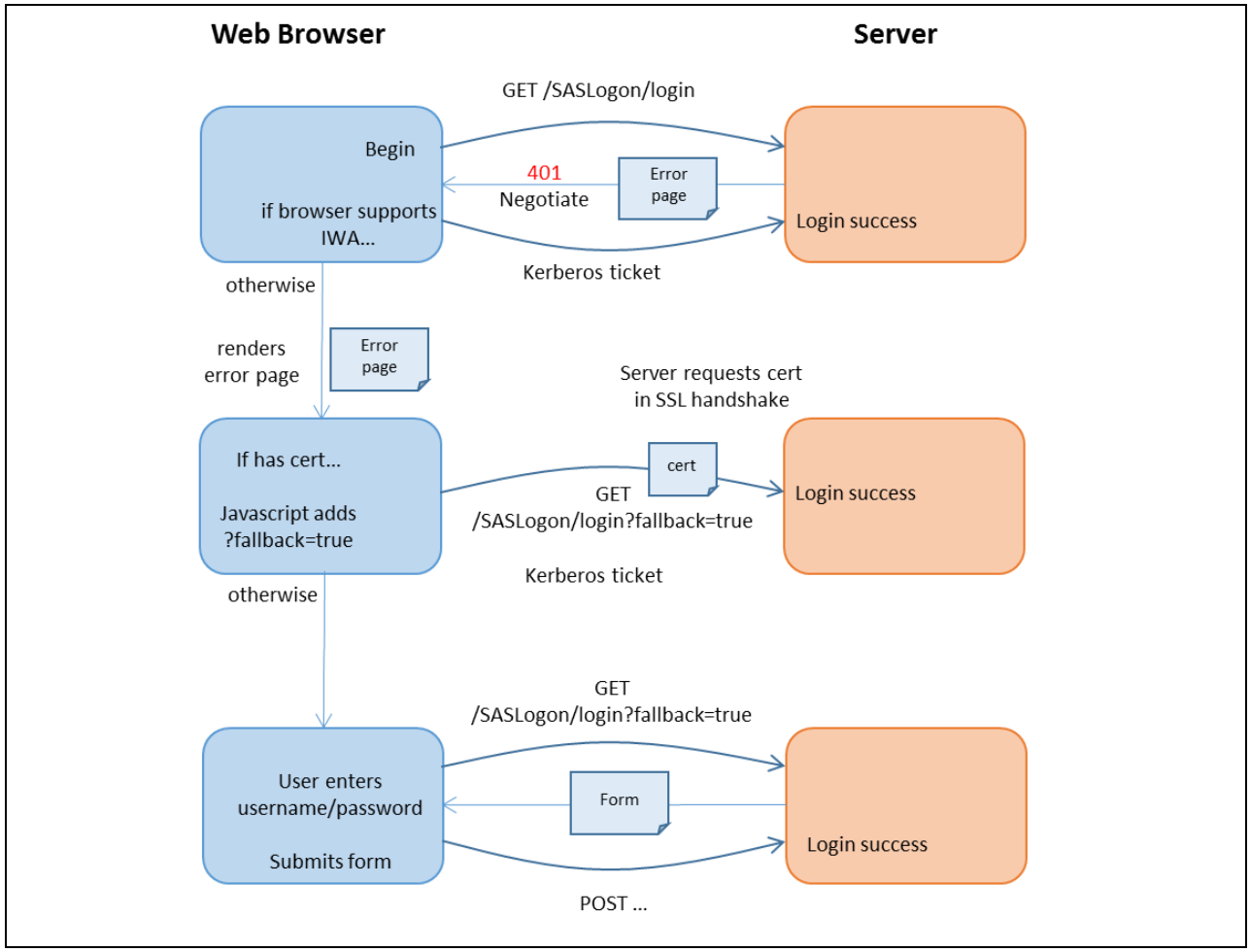

Figure 5. IWA with Fallback to Client Certificate and Form-based Authentication

As described earlier in the paper, this involves configuring the web server with the client CA certificates, and rewrite rules to pass the certificate information in the HTTP headers to the application server. On the application server, the certificate information is reassembled and authenticated by the SAS `SSLAuthenticator` valve. To use Client Certificate Authentication as a secondary authentication method after IWA, we must tell the valve to only process requests with the `fallback=true` query string parameter, leaving the rest to the primary authentication method (IWA in this case). This can be accomplished with the optional `uriPattern` parameter on the valve, which accepts a regular expression for matching request URIs. Additionally, we need to tell the valve to allow requests to fall through if no certificates is provided, so the login form is displayed as a last resort.

```
<Valve className="com.sas.vfabrictcsvr.authenticator.SSLAuthenticator"
       uriPattern="/SASLogon/login.*fallback.*$"
       fallThrough="true" />
```

Here is the detailed flow:

Browser does not support IWA, but has a trusted certificate

1. Client gets redirected to /SASLogon/login from some SAS web application
2. Client sends GET request for /SASLogon/login; web server requests client to provide a certificate and client provides one, which is passed to the application server
3. Server ignores the certificate, responds with 401 status code, header WWW-Authenticate: Negotiate, and error page
4. Client does not support IWA so displays the error page instead
5. Error page automatically redirects back to the login page again but with a fallback=true request parameter
6. Client provides a certificate to the web server again, which is passed to the application server
7. Server skips IWA authentication, valve processing is continued
8. Server authenticates the client based on the certificate, sets a principal in the request
9. CAS gets the principal in the request and trusts it, authentication is successful, client is redirected back to the calling application

CUSTOMER SCENARIO

Many companies use smart card readers to logon to their PC workstations. Microsoft Windows contain extensions that allow users to authenticate using the certificate read from the smart card. Combined with Integrated Windows Authentication, this provides a very secure and seamless way to access applications. SAS 9.4 has been successfully configured in such environments. This section describes one such deployment.

Like many companies, the customer uses Microsoft Active Directory with multiple trusted domain trees in a forest. Every employee has an entry in Active Directory and is also issued an X509 client certificate for their web browser, with the thumbprint of the certificate stored in their Active Directory entry. While most of the employees log into Windows PCs via the smart card reader, some employees use UNIX-based workstations and authenticate with web applications using their browser certificate.

The SAS middle tier was configured with a 3-tier authentication scheme whereby the primary authentication would be through IWA. If the browser did not support IWA, it would fallback to client certificate authentication, and finally to a form-based login. Each of the 3 authentication methods would use a separate Realm in the server configuration.

The primary authentication method, IWA is a special case where the Realm is redundant since the user has already provided a valid Kerberos ticket from the Active Directory-backed domain controller. For this reason, the SAS custom `GSSContextEstablishedRealm` was used. This Realm authenticates any user who has successfully established a GSS Context with the container – in other words has provided a valid Kerberos ticket.

Client certificate and the form-based authentication would each use a `JNDIRealm` to authenticate the credentials against the company's Active Directory using LDAP. As this customer had multiple Active Directory domain trees in a forest, the JNDI realms were configured with a high-level `userBase` and the `userSubtree` option was specified to search the entire subtree. Both of these are standard options with the Apache Tomcat `JNDIRealm`.

SAS 9.4 maintenance release 1 ships with a custom `com.sas.vfabrictcsvr.realm.JNDIRealm` containing some additional options:

`-usernameSuffix`	Suffix to append to the username.
`-userAccountName`	Attribute in the LDAP to set the username from

The `usernameSuffix` option was used to distinguish users in the logs based on how they authenticated. For example, users authenticating with a certificate would be seen by SAS as `<username>-cert` while users authenticating from the form would be seen as `<username>-form`. This required adding multiple web accounts for each user in Metadata.

By default, Apache Tomcat sets retrieves the username from the Distinguished Name (DN) of the certificate but allows you to specify a custom `X509UsernameRetriever` class in the Realm configuration. SAS provides a few such classes:

`X509SubjectCnRetriever`	Common Name (CN)
`X509SubjectThumbprintRetriever`	Unique thumbprint of the certificate

As this customer was a very large organization, the CN and DN were not guaranteed to be unique so the thumbprint was used instead. However, it was not desired to add thumbprints in Metadata or see them in the logs, so the `userAccountName` option was used on the custom realm to set the username from the `sAMAccountName` attribute in the user's Active Directory record.

As a last resort, rather than falling back to the SAS logon form, container-managed FORM-based security was used. The reason for this was two-fold. Importantly, it allowed employees to authenticate against Active Directory LDAP instead of SAS Metadata. Secondly, it also allowed us to display some information on the form about their certificate, indicating for example if it was expired.

CONCLUSION

The SAS 9.4 and SAS VA middle tier architecture provides a flexible and secure platform. Built around SAS Application Server, it consolidates and supports a wider range of authentication protocols than 9.3 and earlier releases. We began by looking at where middle tier security falls into the overall SAS 9.4 architecture. We discussed the SAS Logon Manager, which provides authentication services to SAS VA and SAS solutions using industry-standard CAS software. We reviewed how container security is configured in the SAS Application Server, and how SAS can be integrated with Enterprise single sign-on infrastructure.

As a new feature and for advanced users, we introduced a new authentication framework, Fallback Authentication, which can enable multiple security protocols to effectively be chained together in order of precedence. We looked at how the SAS 9.4 middle tier supports Integrated Windows Authentication and how this can be configured to 'fallback' to X509 client certificate authentication and finally form-based authentication. This framework is used by our customers and the summary of our experience from working with customers will help anyone who is interested in using the similar authentication strategy.

REFERENCES

SAS Institute Inc. (2013). "SAS 9.4 Intelligence Platform: Middle Tier Administration Guide, Second Edition." Available at http://support.sas.com/documentation/cdl/en/bimtag/66823/PDF/default/bimtag.pdf

SAS Institute Inc. (2013). "SAS 9.4 Intelligence Platform: Security Administration Guide, Second Edition." Available at http://support.sas.com/documentation/cdl/en/bisecag/67045/PDF/default/bisecag.pdf

Zhiyong Li, Rob Stephens, and Alec Fernandez, Migrating SAS® Java EE Applications from WLS/WAS/JBoss to SAS Web Server, SGF 2014.

Amy Peters, Bob Bonham and Zhiyong Li, Monitoring 101: New Features in SAS 9.4 for Monitoring Your SAS Intelligence Platform, available at: https://support.**sas**.com/resources/papers/proceedings13/463-2013.pdf

Zhiyong Li, IWA for a Spring Desktop and Web Application, Java Developers' Journal, http://java.sys-con.com/node/1326751.

Jan De Clercq, Microsoft. "Smart Cards." Accessed Jan 2014. Available at http://technet.microsoft.com/en-us/library/dd277362.aspx

The Apache Software Foundation (October 18, 2013). "Apache Tomcat 7: The Realm Component" Accessed Jan 2014. Available at http://tomcat.apache.org/tomcat-7.0-doc/config/realm.html

The Apache Software Foundation (October 18, 2013). “Apache Tomcat 7: Windows Authentication How-To” Accessed Jan 2014. Available at http://tomcat.apache.org/tomcat-7.0-doc/windows-auth-howto.html

ACKNOWLEDGMENTS

Thanks Casey Hadden and Alan Eaton for providing the SAS Logon Manager and CAS interaction diagram.

CONTACT INFORMATION

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

Name: Zhiyong Li
Enterprise: SAS Institute, Inc.
Address: SAS Campus Drive
City, State ZIP: Cary, NC 27513
Work Phone: (919)531-9068
Fax: (919)677-4444
E-mail: Zhiyong.Li@sas.com
Web: www.sas.com

Name: Mike Roda
Enterprise: SAS Institute, Inc.
Address: SAS Campus Drive
City, State ZIP: Cary, NC 27513
Work Phone: (919)531-5891
Fax: (919)677-4444
E-mail: Mike.Roda@sas.com
Web: www.sas.com