Paper 1761-2014

Test for Success: Automated Testing of SAS® Metadata Security Implementations

Paul Homes, Metacoda

ABSTRACT

SAS® platform installations are large, complex, growing, and ever-changing enterprise systems that support many diverse groups of users and content. A reliable metadata security implementation is critical for providing access to business resources in a methodical, organized, partitioned, and protected manner. With natural changes to users, groups, and folders from an organization's day-to-day activities, deviations from an original metadata security plan are very likely and can put protected resources at risk. Regular security testing can ensure compliance, but, given existing administrator commitments and the time consuming nature of manual testing procedures, it doesn't tend to happen.

This paper discusses concepts and outlines several example test specifications from an automated metadata security testing framework being developed by Metacoda. With regularly scheduled, automated testing, using a well-defined set of test rules, administrators can focus on their other work, and let alerts notify them of any deviations from a metadata security test specification.

INTRODUCTION

Many organizations use SAS metadata security to secure valuable resources that are made available to SAS platform users. Those resources might include servers, libraries, tables, information maps, stored processes, and reports for example. After an initial SAS metadata security implementation, it needs to be thoroughly tested to verify that those resources are appropriately secured. SAS platform installations are very dynamic: changes to content, the user base, and access controls occur continuously. In the face of these changes, it is important to repeat metadata security testing on a regular basis to ensure resources remain adequately secured. However, metadata security testing is usually performed manually and is very time intensive and error prone. Manual testing with good coverage can take several days of effort. Consequently, manual testing is usually performed very infrequently and with very limited coverage. Planned or accidental changes can cause resources to become inappropriately secured. Without regular testing, these issues might not be detected for days, weeks, or even months after the event. This paper will show how metadata security testing can be automated. Automated testing enables an organization to perform extensive metadata security testing as frequently as they like and to detect metadata security issues significantly faster. By scheduling a large suite of automated metadata security tests, an organization can be made aware of metadata security problems within minutes or hours.

This paper will explain the need for automated testing of metadata security implementations, discuss the types of tests that usually need to be done, and show, through examples, how those tests can be specified in a simple XML format. The paper will also explain how a metadata security testing engine can take those test specifications and run them over an existing SAS environment. Furthermore, in situations where there are existing environments, explain how those XML test specifications can be generated from those environments. Generated test specifications can be used for testing against the same environment at a later date (to detect changes), or used for testing against a different target environment which should have a similar metadata security setup to the source environment (to confirm no relevant differences). This might include a test environment that should mirror a production environment or a single environment being migrated from an older SAS version to a newer SAS version.

The primary audience for this paper are SAS platform administrators, but it may also be of interest to project managers, team leaders, system testers and I.T. security auditors. SAS platform administrators implement metadata security and are often also those who perform the testing (albeit manually). Non-administrators may not implement metadata security themselves, but may be responsible for teams that test the implementations.

SAS metadata security implementations represent just one of several security layers involved in SAS platform installations. Other security layers include database, web, and operating system authorization facilities. This paper focuses purely on the SAS metadata authorization layer. Of course, a good security plan will utilize all relevant authorization layers and incorporate redundancies to protect any required assets.

The paper assumes some familiarity with the SAS platform authorization concepts of users, groups, roles, capabilities, identity hierarchies, inheritance paths, Access Controls Templates (ACTs), explicit and indirect permissions. An explanation of these concepts can be found in the *SAS 9.4 Intelligence Platform: Security Administration Guide* as well as other resources listed in the *Recommend Reading* section at the end of the paper.

THE NEED FOR AUTOMATED METADATA SECURITY TESTING

Over the years, the breadth and complexity of SAS software has grown immensely. When I started using SAS there was no metadata server and no metadata security. The use of SAS software within many organizations has grown significantly too. We have seen small departmental installations of SAS software, with very simple security requirements, grow into much larger, shared, enterprise-level installations, supporting many different group of users with differing needs and access levels for the organizations assets. Security has become much more important, especially when dealing with more sensitive content such as credit, medical, and human resources data.

The roles and responsibilities for SAS platform administrators have grown too. When I started using SAS there were no administrators. Back then, SAS software administration, not that we called it that, was a tiny aspect of your normal role: install the software on your PC and upgrade sometime in the future. These days we can have teams of administrators that manage SAS installations full time. SAS platform administrators are usually very busy people and managing metadata security for the shared SAS platform is just a small, but often very important, part of their overall responsibilities.

SAS platform administrators are usually involved in all phases of a metadata security implementation, from the initial gathering of security requirements, the design and implementation of the SAS metadata groups, roles, capabilities, folder structures and associated access controls, testing of the implementation, and finally support and troubleshooting of security issues in production systems. Of course, they are also involved in the promotion of ongoing changes from development systems to production systems and ensuring each of those changes has no detrimental impact on the security of the platform.

Over the years, we have seen SAS administrators spend a significant amount of time testing, re-testing, and troubleshooting metadata security implementations to make sure that all of the carefully crafted groups, roles, folders and access controls remain intact, over time, as changes are made by multiple internal administrators, developers and external consultants alike. At Metacoda, we also need to do this metadata security testing ourselves to ensure consistency in our own development and testing environments.

When troubleshooting SAS metadata security issues, often to determine why people have more or less access than expected, it often comes down to invalid assumptions on the state of applied metadata security. There can be expectations that users are still members of certain groups, (directly or indirectly), still members of certain roles (directly or indirectly), or still have certain application capabilities (directly or indirectly). New hires or consultants might have an expectation that pre-defined access controls still exist, unmodified, and continue to be applied in the standard locations. There are expectations that the culmination of group memberships, and the application of explicit permissions and ACTs on parent objects, will result in a certain set of effective permissions on lower level objects for certain users. It is often during troubleshooting, and the required manual verification of these assumptions, that these assumptions or expectations are shown to be no longer valid: someone may have been removed from a group, or a nested group is no longer nested under an expected parent group. It may be that an ACT has been unexpectedly modified, or quick-fix explicit permissions have been set on objects and not revisited later. Manual discovery of these breakages may not occur for days, weeks, or months, and the investigation and rectification is usually required immediately once identified.

Typically metadata security testing and troubleshooting is done manually. Manual metadata security testing is intensely time consuming. Manual testing might involve the administrator examining the properties dialogs for large numbers of objects. It might involve impersonating users to see what they see in target SAS applications. It might involve coordinating time with the user to get them to test scenarios themselves. Any subsequent changes to applied metadata security, such as ACTs, explicit permissions, group and role memberships, would invalidate this testing, necessitating a fresh round of testing. It is little wonder that metadata security testing tends to be done only after large changes or during important projects. The reality is that small changes are always being applied in between these projects. Users move in and out of groups as they join, move within, and leave an organization. New content is being developed continuously. Some of that content may be sensitive and for the consumption of specific groups of users. If metadata security is important then security testing needs to happen regularly so mistakes or inconsistencies can be detected early. In our experience, manual and time consuming testing is rarely done regularly enough, and sometimes not at all.

LEARNING FROM SOFTWARE DEVELOPMENT PRACTICES

As administrators we might learn from the experience of software developers. They handle the enormity of regular testing of large complex systems through the practices of *Unit Testing* and *Continuous Integration*. The expected behaviour of the system is encapsulated in a series of unit tests which will either pass or fail depending on whether correct behaviour is seen. Often these tests are written before the software itself: the test starts out failing, the software is written, and when it is operating as expected the tests pass. With continuous integration, whenever small changes are introduced into a system, an entire army of unit tests is re-run to verify that the change has not

unexpectedly impacted on another part of the system. These tests are run regularly, often when a change is committed to a version control system to be shared with the rest of the team. Problems are picked up very quickly, before they cause significant problems, and can be either fixed or reverted for further investigation.

The similar approach of unit testing and continuous integration could be applied to SAS metadata security. Metadata security *Unit Tests* can be written as a formal contract on the expected state and behaviour of a SAS metadata security implementation. The tests can be run to verify that the current state of a SAS environment matches the expectations, as encapsulated in the tests. The result of running all tests is either a pass or a fail, together with details of where any failures have occurred. By scheduling the tests to run in batch, and on a regular basis, we get the benefit of early alerts as seen in *Continuous Integration*. The administrator can deal with them as they arise, resolve any urgent issues immediately, and perhaps modify any tests as required.

As a side benefit, the tests, if written in an easily understood format, and given enough coverage, can also act as necessarily accurate documentation on the state of the SAS metadata security implementation. In making sure tests continue to pass, the "documentation" will be kept up to date.

Armed with the knowledge that any deviation from the SAS metadata security specification, as embodied in the tests, will be rapidly tested and trigger alerts, SAS administrators can have a great deal more confidence in the changes that they make.

Of course, writing tests takes time, but it is a small investment that will pay dividends many times over as changes are made then immediately, and automatically, verified. An existing SAS installation could also be used as the basis for the automated generation of baseline tests.

Another software development concept that could help in this area is *Test Driven Development*, where tests are written first, before the software itself is written. Similarly, SAS metadata security tests could be developed immediately after the design step and just before implementation. The tests will start out failing, then continue to fail until such time as the metadata security implementation is complete and correct, at which point the tests will finally pass.

GOALS FOR METADATA SECURITY TESTING

There are many goals and requirements when conducting regular testing of SAS metadata security implementations. Let's consider a few.

ENSURE BASELINE PRE-DEFINED SECURITY REMAINS IN-PLACE

New SAS platform installations come with a pre-defined metadata security implementation to serve as the baseline for customer specific extension and customization. This pre-defined security gives a new installation a secure starting point. It consists of some key users, groups, roles, capability assignments, ACTs & access control applications. Much of this pre-defined security must remain in place for an installation to remain secure, and regular testing can highlight any changes to this baseline security. We have seen situations where some of these pre-defined ACTs and access control applications have been modified or deleted, causing problems or unexpected access to content. Regular testing would quickly identify such broad reaching changes.

ENSURE A MINIMUM LEVEL OF CUSTOMIZED SECURITY IS CONTINUOUSLY MAINTAINED

Whilst pre-defined security provides a good baseline implementation, SAS customers further invest in requirements gathering, design, and implementation of additional metadata security, to logically partition and secure their content for their diverse user communities. They might create metadata folders for different business units and secure those folders using groups and ACTs. Sometimes there is a need to create logical servers for different business units and secure those servers in metadata to ensure they are only available to the right groups. Regular testing can be used to ensure those access controls are maintained and not accidentally, or deliberately, removed or modified. Regular testing can ensure the correct effective permissions on those resources for candidate users and groups. Effective permission testing further helps to protect against unexpected changes due to higher level modifications to indirect resources, such as repository ACTs, group structure reorganizations, and ACT definition changes.

ENSURE CONSISTENCY OF SECURITY ACROSS MULTIPLE SAS ENVIRONMENTS

Many SAS customers have more than one SAS platform installation or environment. These are often independent, isolated, environments for development, testing, staging, and production purposes. Each environment can have a unique metadata security implementation to cover the different ways in which the environments are used. Different classes of users, such as developers and business users may have different access levels to each environment. Whilst the entirety of the metadata security implementation may be different across every environment, there will usually be some aspects which need to be applied consistently across each environment. This might include the list of groups and roles, memberships for key groups and roles, key access control definitions and their application to key

objects. Test specifications should be able to be different for each environment, but also reference, or include, common test specifications that should apply across multiple environments. Some environments might be required to almost mirror each other. In those situations it would be useful to be able to export the source environment's metadata security implementation as a set of baseline test specifications which, with potential modifications, could be run against a target environment to ensure consistency.

SUPPORT MIGRATION DURING SAS UPGRADES

In addition to testing for metadata security consistency between environments with the same SAS version, it is also useful to be able to test for consistency between environments with different SAS versions. This can help when upgrading an environment from SAS 9.3 to SAS 9.4 for example. The older versioned environment could be used to generate a set of test specifications to run against the newer versioned environment, ensuring relevant aspects of the existing metadata security implementation have been correctly migrated to the new environment.

VERIFY APPROPRIATE ACCESS FOR CANDIDATE USERS

When implementing SAS metadata security, administrators will configure groups, roles, capability assignments, ACTs & access control applications to ensure that appropriate users have appropriate access levels to appropriate objects. On some objects they should have no access, on some they might have read-only access and on others they might have update access. Whilst the implementation specifies the intent, it is testing of the actual effective permissions, on key objects for key users, which will provide confidence that the implementation is performing as expected. Effective permissions are sensitive to indirect changes at multiple levels; including repository ACTs, identity hierarchy changes through group modifications, and changes in ACT and explicit permission applications on parent objects in object inheritance paths.

The ability to regularly and quickly perform a battery of effective permission tests, for real or candidate users, on reasonably sized collections of key metadata objects, can provide administrators with a high level of confidence that small changes have not had unintended consequences elsewhere.

SUPPORT SCHEDULED TESTING WITH FAILURE ALERTS

For computer backups we don't rely on people to manually run backups. Instead we establish formal automatic scheduled processes to ensure reliable and regular backups. Similarly, for reliable, regular metadata security testing we should be able to schedule the tests to run in batch without the requirement for any normal involvement by administrators when the tests are working as expected.

When metadata security tests do fail we want administrators to be promptly notified about these deviations from the metadata security plan. They can be alerted to test failures through email notifications which should include details, or links to reports, about the test failures detected. The failures can be investigated and resolve as required. This might require updating the tests to match new requirements in the implementation, or a reversal of the metadata changes to bring the implementation back in line with the tests.

Additionally, successful test results should also be available to support auditing that tests have been run.

SUPPORT GOOD PRACTICES

Over the years, based on the experience of many administrators, good practices have emerged that can help administrators avoid typical problems in the area of metadata security. Some of these practices include:

- Applying access controls using groups rather than individual users.
- Only denying permissions to the implicit groups, PUBLIC and SASUSERS, making sure that explicit groups are only used for granting permissions.
- Limiting the number of members in highly privileged groups and roles.

Where required, additional metadata security tests can be used to ensure these good practices are followed.

A METADATA SECURITY TESTING FRAMEWORK

We have spent some time at Metacoda considering how best to provide automated metadata security testing for both ourselves and our customers. We are now incorporating our work in this area into the next major version of our Metacoda Security Plug-ins product. This includes the ability to export an existing metadata security implementation from a metadata server as a set of test specifications, to interactively run test specifications from within the SAS Management Console, as well as the ability to run test specifications in batch so they can be scheduled using an organizations preferred scheduler.

To remove the requirement for complex SAS metadata API coding, and an in-depth understanding of the underlying SAS metadata model, this testing framework is based on an XML test specification. XML test specification files describe the key aspects of a site's metadata security requirements: ACT definitions, applications to objects, group members, effective permissions on key objects for candidate users etc. Defining tests in XML has multiple benefits.

- The tests can be written by a wider audience, without requiring skills and experience in Java, .NET or SAS language interfaces to metadata.
- The tests will be much more succinct and understandable than code, and can be read by many.
- XML editors can be used to ensure a correct well-formed document that adheres to a schema.
- The tests can be checked into a version control system (VCS) such as CVS, SVN, Git and many others. This makes it easy to track changes to the tests over time. Additionally, as a text based format, existing VCS tools can readily show the detailed differences between versions of the tests.

A testing engine takes the XML test specifications and evaluates the current state of metadata security in an existing SAS environment. Metacoda is developing such an engine but the types of tests and the XML test specification described in this paper are generic and could equally be processed by a custom or third party testing engine.

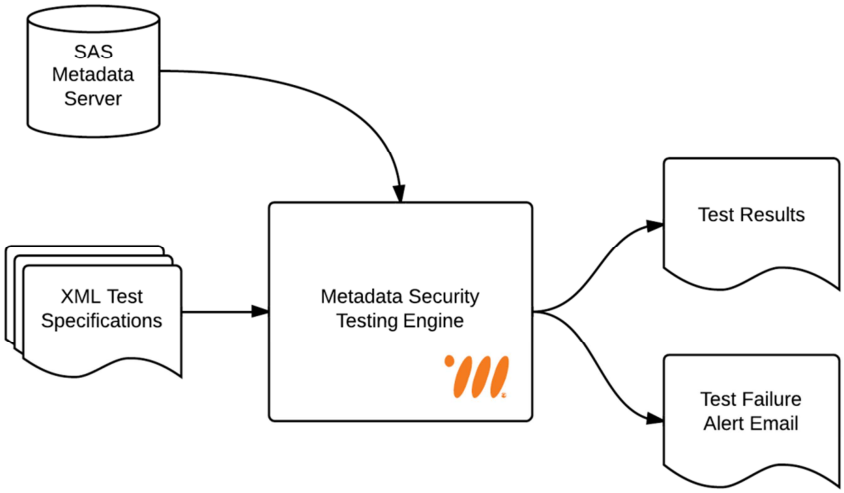

Figure 1 Metadata Security Testing Engine

The output of the testing engine is a report on test results with detailed descriptions of any failures encountered where the metadata security in place doesn't match the test specification. The engine can send alerts to administrators in the event of any failures. The testing engine can run the tests in batch with no intervention so the tests can be scheduled to run regularly.

To help with the generation of test specifications, based on an existing metadata security implementation, an export engine can connect to a metadata server, analyse its security metadata, and generate XML test specifications that describe that implementation. Options are used to control what should be exported and the level of detail to be included. This removes the requirement to completely hand-craft these XML test specification documents. In some situations the exported XML specifications might be used unmodified, or partially modified, to regularly re-test the same environment and ensure it has not been significantly modified.

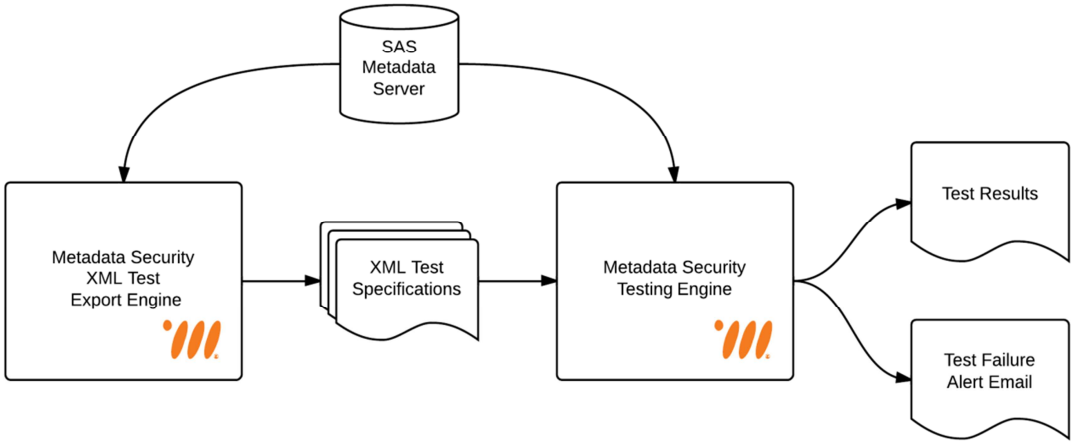

Figure 2 Metadata Security XML Test Export Engine (same environment)

Alternatively, the export engine can be used to generate XML test specifications from a source metadata server that are then used as inputs into the testing engine and run against a different target metadata environment. How much of the exported XML test specification is used unedited will depend upon how similar the source and target environments are expected to be.

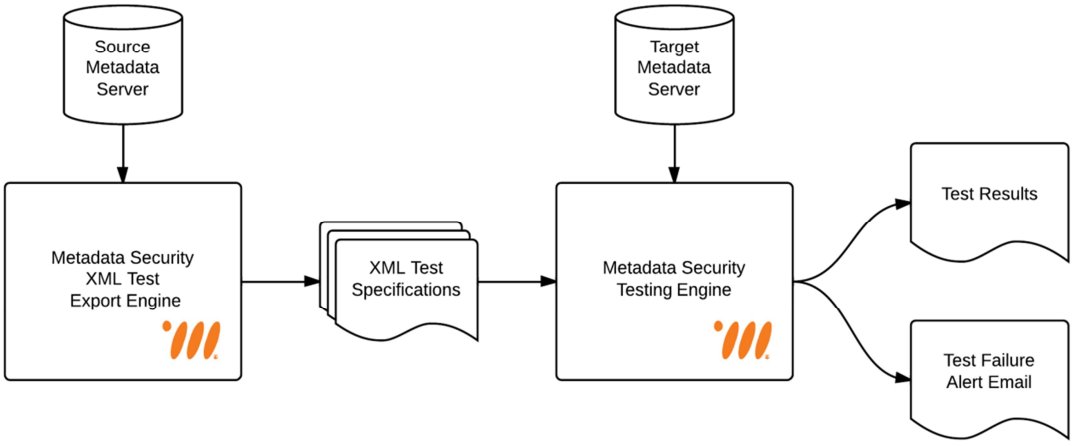

Figure 3 Metadata Security XML Test Export Engine (different environments)

SAMPLE TESTS

This section provides several examples of XML test specification fragments to show the types of tests that can be specified and the simplicity of using XML in lieu of writing complex metadata API code. All sample tests are based on the expectation of pre-defined security in a new SAS platform installation, so the reader should already be familiar with the users, groups, roles, capabilities, ACTs, and the application of ACTs and explicit permissions on key objects.

IDENTITIES EXIST

The first sample test, and one of the simplest, is to verify that key identities (users, groups, roles) still exist in a SAS metadata repository. We can also verify that some of their key attributes have not been modified post-installation.

The following XML test specification fragment is used to ensure that the pre-defined users *SAS Administrator* and *SAS Trusted User* still exist along with some of their basic attributes.

```
<Users complete="false">
  <User required="true" name="sasadm" displayName="SAS Administrator"
        internal="true" external="false"/>
  <User required="true" name="sastrust" displayName="SAS Trusted User"
        internal="true" external="false" login="false"/>
  ...
</Users>
```

The number of users is likely to be very variable and so the *Users* collection is marked as *complete="false"* meaning that there is an expectation that additional users will be found in metadata that have not been specified in the test. Both of the specified users have *required="true"* - they are not optional and must be found in metadata for the test to succeed. The *displayName* attributes have been specified so will be compared with what is found in metadata. Both users have *external="false"* – they must not be found to be linked to any external identities (such as via Active Directory or LDAP sync). Both users must have SAS internal accounts and additionally the *SAS Trusted User* must not have any (non-internal) logins of its own (other tests can be used to verify it has indirect access to the *sassrv* login).

The next sample XML fragment shows a similar test for the existence of the expected groups *SAS Administrators*, *SAS System Services* and *SAS General Servers*.

```
<Groups complete="false">
  <Group required="true" repository="Foundation" name="SASAdministrators"
         displayName="SAS Administrators" external="false" login="false"/>
  <Group required="true" repository="Foundation" name="SAS General Servers"
         displayName="SAS General Servers" external="false" login="true"/>
  <Group required="true" repository="Foundation" name="SAS System Services"
         displayName="SAS System Services" external="false" login="false"/>
  ...
</Groups>
```

Notice that the *SAS General Servers* group differs from the others in that it has *login="true"* and must have at least 1 login (which is the normal *sassrv* login). Later in the paper we will extend these samples to show how we can specifically test for the presence of a *sassrv* login which would be accessible to the verified *SAS Trusted User* member.

In addition to testing the existence of users and groups, we can also test for the presence of expected roles, such as *Metadata Server: Unrestricted* and *Web Report Studio: Report Viewing*.

```
<Roles complete="false">
  <Role required="true" repository="Foundation"
        name="META: Unrestricted Users Role"
        displayName="Metadata Server: Unrestricted"/>
  <Role required="true" repository="Foundation"
        name="Web Report Studio: Report Viewing"
        displayName="Web Report Studio: Report Viewing"/>
  ...
</Roles>
```

IDENTITY HAS CORRECT GROUP/ROLE MEMBERSHIPS

The prior tests were simple tests for the existence of identities. We can further enhance those tests for users and groups by also verifying that they are still members of required groups and roles. Whilst we might not verify memberships for all users and groups, it can be very useful to verify that a few test/candidate users have the correct group/role memberships. We can ensure that key service identities still have the same group/role memberships they had at installation time, and have not been modified post-installation. Membership can be partial or complete. Partial means they might also be members of other groups/roles not listed, Complete means they must only be members of the specified groups/roles.

This next XML fragment is used to verify that the *SAS Administrator* user is a direct member of both the *SAS Administrators* group and the *Metadata Server: Unrestricted* role. As direct membership tests, the user must be a direct member and cannot get its membership indirectly via other groups (other tests can be used if you want to test indirect membership or membership regardless of level). These are also complete membership tests - if the user is actually found to be a member of any other unspecified groups or roles the test will fail and generate an alert.

```
<User name="sasadm">
  <DirectGroupMemberships complete="true">
    <Group required="true" name="SASAdministrators"/>
  </DirectGroupMemberships>
  <DirectRoleMemberships complete="true">
    <Role name="META: Unrestricted Users Role"/>
  </DirectRoleMemberships>
  ...
<User>
```

The next XML fragment verifies that the *SAS Trusted User* is a direct member of a few required groups, including the *SAS General Servers* group which provides access to the *sassrv* login. By specifying a complete and empty direct role memberships list, the test also ensures the user is not a member of any roles.

```
<User name="sastrust">
  <DirectGroupMemberships complete="true">
    <Group required="true" name="BI Dashboard Administrators"/>
    <Group required="true" name="SAS General Servers"/>
    <Group required="true" name="SAS System Services"/>
  </DirectGroupMemberships>
  <DirectRoleMemberships complete="true"/>
  ...
<User>
```

IDENTITY HAS CORRECT APPLICATION CAPABILITIES

In addition to group and role memberships, you may want to confirm that key test/candidate identities (users, groups, roles) have, or do not have, access to capabilities, or features, in SAS applications. These effective capability access tests take into account direct, indirect, contributed and implicit capability access. You can verify access without being concerned about the mechanism of access. Additional tests are used if you do need to ensure the mechanism by which a capability is acquired or not.

The following XML fragment can be used to verify the *SAS Administrator* user has access to the SAS Management Console *Server Manager* plug-in, and that the *SAS Demo User* does not.

```
<User name="sasadm">
  <Capabilities complete="false">
    <Capability required="true" name="Server Manager" granted="true"
                parentFolder="Management Console 9.4:/Plug-ins" />
    ...
  </Capabilities>
  ...
<User>
<User name="sasdemo">
  <Capabilities complete="false">
    <Capability required="true" name="Server Manager" granted="false"
                parentFolder="Management Console 9.4:/Plug-ins" />
    ...
  </Capabilities>
  ...
<User>
```

Due to the number of capabilities, it is unusual to test complete capability sets. It is more likely a few candidate users and a few key capabilities would be chosen.

IDENTITY HAS CORRECT ACCOUNTS/LOGINS

Sometimes it is necessary to verify that a user or group has access to a login. These logins might be directly owned by the identity itself, or indirectly accessible through the identities group memberships. You could use this to test the accessibility of shared database logins.

The following XML fragment is used to verify that the *SAS Trusted User* (as used by the SAS Object Spawner) has no direct logins of its own but does have indirect access to the *sassrv* login so it can spawn SAS services such as stored process servers and pooled workspace servers. Notice that the *sassrv* login must also have a password present (which can be verified from SAS 9.3 onwards).

```
<User name="sastrust">
  <DirectLogins complete="true"/>
  <IndirectLogins complete="true">
    <Login required="true" authDomain="DefaultAuth"
           userId="sassrv" hasPassword="true"/>
  </IndirectLogins>
  ...
<User>
```

GROUP/ROLE HAS CORRECT MEMBERS

Some of the tests above have shown how memberships for identities can be tested. In other situations you may want to approach testing from the other perspective and instead verify the members for specific groups and roles. Members can be *partial* or *complete*. Partial means that it is ok if additional members are found that are not specified in the test. Complete means the test must specify all members and the test fails if it finds additional members not specified. You might not want to specify complete members for all groups, particularly those containing variable sets of users, but sometimes you want to verify that certain key users or groups are members of particular groups or roles. This can be particularly useful for verifying the correct organization for nested group and role structures.

This XML fragment shows how to verify that the *SAS Administrators* group has the user *SAS Administrator* as a member (with others allowed).

```
<Group repository="Foundation" name="SASAdministrators">
  <DirectMembers complete="false">
    <User required="true" name="sasadm"/>
  </DirectMembers>
  ...
<Group>
```

This next XML fragment verifies that the *SAS System Services* group has the *SAS Trusted User* as the only member allowed.

```
<Group repository="Foundation" name="SAS System Services">
  <DirectMembers complete="true">
    <User required="true" name="sastrust"/>
  </DirectMembers>
  ...
<Group>
```

You might also want to verify that the *Metadata Server: Unrestricted* role has the *SAS Administrator* user as the only member.

```
<Role repository="Foundation" name="META: Unrestricted Users Role">
  <DirectMembers complete="true">
    <User required="true" name="sasadm"/>
  </DirectMembers>
  ...
<Role>
```

This next test specifies that the *Enterprise Guide: Advanced* role must have the *PUBLIC* group as the only member.

```
<Role repository="Foundation" name="Enterprise Guide: Advanced">
  <DirectMembers complete="true">
    <Group required="true" name="PUBLIC"/>
  </DirectMembers>
  ...
<Role>
```

Occasionally you might want to verify that a role, such as the *Enterprise Guide: Analysis*, doesn't have any members.

```
<Role repository="Foundation" name="Enterprise Guide: Analysis">
  <DirectMembers complete="true" />
  ...
<Role>
```

ACCESS CONTROL TEMPLATES EXIST

All of the tests shown so far have been about identities and their relationships with each other. These next series of sample tests relate to access controls – how they have been defined and where they have been applied. The first of these tests is to simply verify the existence of Access Control Templates (ACTs). These can be used to verify that pre-defined ACTs, automatically created during installation, have not been removed post-installation. You can also verify that any required custom ACTs are also present. Later variations on these tests will show how to confirm the ACTs have the expected definition (permission pattern) and have been applied to various objects as expected.

This test verifies the pre-defined ACTs *Default ACT*, *SAS Administrator Settings*, and *Private User Folder ACT* must exist and the *Portal ACT* may exist. It also ensures *Default ACT* is the repository ACT.

```
<ACTs complete="true">
  <ACT required="true" repository="Foundation" name="Default ACT"
       desc="..." repositoryACT="true" />
  <ACT required="true" repository="Foundation" name="SAS Administrator Settings"
       desc="..." />
  <ACT required="true" repository="Foundation" name="Private User Folder ACT"
       desc="..." />
  <ACT required="false" repository="Foundation" name="Portal ACT"
       desc="..." />
  ...
<ACTs>
```

As a "*complete*" ACTs collection, any additional ACTs found in the repository that are not specified here will cause the test to fail. Unlike identities, which can be highly variable, there are often a relatively small number of ACTs which, once defined, are quite static and can be more rigorously tested.

ACCESS CONTROL TEMPLATE HAS CORRECT IDENTITIES & PERMISSIONS

The previous test only checked for the existence of ACTs. It is also important to test that they have the expected definition, or permission patterns, and that this collection of identities and associated permissions has not been accidentally or deliberately modified. For pre-defined ACTs you can test that they still have the same definition they had at installation time, and have not been modified post-installation.

The following test verifies that the pre-defined *SAS Administrator Settings* ACT still has its original and complete definition.

```
<ACT repository="Foundation" name="SAS Administrator Settings">
  <PermissionPattern complete="true">
    <Group required="true" repository="Foundation" name="SASAdministrators"
           permissions="+RM,+WM,+CM,+A"/>
    <Group required="true" repository="Foundation" name="SAS System Services"
           permissions="+RM"/>
  </PermissionPattern>
  ...
<ACTs>
```

ACCESS CONTROL TEMPLATE HAS BEEN APPLIED TO OBJECTS

After ACTs have been applied to key objects in a repository, regular testing can ensure those ACT applications are maintained. This can include pre-defined ACT applications, automatically configured during the initial installation of the SAS platform, as well as custom ACT applications done post-installation.

The next XML fragment shows how to verify that the *SAS Administrator Settings* ACT is still applied to the following objects at a minimum:

- The root metadata folder: often seen in SAS clients with the name *SAS Folders*, but actually found in metadata as *BIP Service*.
- The folder /*User Folders*
- The ACT named *Default ACT*
- The ACT named *SAS Administrator Settings*

```
<ACT repository="Foundation" name="SAS Administrator Settings">
  <Objects complete="false">
    <ACT required="true" name="Default ACT"/>
    <ACT required="true" name="SAS Administrator Settings"/>
    <Object required="true" publicType="RootFolder" name="BIP Service"/>
    <Object required="true" publicType="Folder"
            parentFolder="/" name="User Folders"/>
    ...
  </Objects>
  ...
<ACTs>
```

As a non-complete *Objects* collection, the *SAS Administrator Settings* ACT can be applied to other objects without failing the test, but must be applied to these objects, as a minimum, for the test to pass.

OBJECT HAS ACCESS CONTROLS APPLIED

This next sample test takes a different perspective. The previous test was used to verify an ACT was applied to a set of objects. This test is used to verify that a set of access controls, consisting of both ACTs and explicit permissions, have been applied to an object. It can be used to confirm a partial, incomplete, or minimum, set of access controls, where additional access controls may be found. Alternatively, it can be used to test for a complete set of access controls, where any additional unspecified access controls will cause the test to fail.

The XML below verifies that the root folder has explicit permissions, denying the WriteMetadata (WM) and CheckinMetadata (CM) permissions to the *PUBLIC* group, in combination with use of the *SAS Administrator Settings* ACT.

```
<Objects>
  <Object required="true" publicType="RootFolder" name="BIP Service">
    <AccessControls complete="true">
      <ACT required="true" name="SAS Administrator Settings"/>
      <Group required="true" name="PUBLIC" permissions="-WM,-CM"/>
    </AccessControls>
    ...
  <Object>
  ...
<Objects>
```

USER HAS CORRECT EFFECTIVE PERMISSIONS ON AN OBJECT

Most of the previous tests have been concerned with testing the metadata security implementation, in terms of the relationships between users, groups and roles, and the application of access controls on objects. This next test can be used to confirm the expected results of that implementation in terms of the expected effective permissions for users and groups on metadata objects.

Considering the dynamic nature of users in an organization, and the sheer number of metadata objects in a repository, it is impractical to specify and test the effective permissions for all users on all objects. However, a few candidate users can often be identified, in order to test the implementation by ensuring those users have an expected

set of permissions on key objects, such as folders, libraries, tables, stored processes, information maps, reports etc. Sometimes special test users are created specifically for these purposes. Effective permission tests can be thought of as testing the end result of a complex system of access controls and group memberships by saying we expect this user to be able to do these tasks on this object. These tests can help ensure sensitive folders, and their contained objects, are only visible to appropriate users and groups. If these folders and their contents are accidentally exposed through a change in access controls (directly or indirectly), the effective permissions tests will quickly alert administrators of the potential issue.

This final XML fragment shows how to verify expected effective permissions for a few sample users (*SAS Administrator*, *SAS Demo User* and *SAS Trusted User*) on the */Shared Data* folder.

```
<Objects>
  <Object required="true" publicType="Folder" parentFolder="/" name="SharedData">
    <EffectivePermissions>
      <User name="sasadm" default="granted" permissions=""/>
      <User name="sasdemo" default="denied" permissions="+RM"/>
      <User name="sastrust" permissions="+RMi,-WMi,-CMi,-Ri,-Wi,-Ci,-Di,-Ai"/>
    </EffectivePermissions>
    ...
  <Object>
  ...
<Objects>
```

The simplest specification is for the *SAS Trusted User*, with name *sastrust*, which should have the ReadMetadata permission granted indirectly (RMi) and all other permissions denied indirectly. For the other users, the permissions have been abbreviated with the use of *default="granted"* or *default="denied"*. This *default* attribute indicates the expected state of unspecified permissions. The *permissions* attribute is then used to specify only those permissions which are different from the default. The specification for the *SAS Demo User*, with name *sasdemo*, has the ReadMetadata permission granted and all other permissions denied. The unrestricted *SAS Administrator* user, with name *sasadm*, has a specification that all permissions should be granted.

CONCLUSION

The process of manual metadata security testing is too time-intensive or not done at all. At the same time metadata security can be critical for a business to appropriately secure digital assets made available through the SAS platform. It's not practical to manually test metadata security with enough depth, coverage and regularity to confidently and continuously know that the platform is still secure in accordance with a defined security plan.

Automated metadata security testing makes much better use of a SAS platform administrator's limited availability. They can instead focus their metadata security testing efforts on writing metadata security test specifications that can much more frequently test more aspects of a metadata security implementation than is possible through manual procedures. By scheduling tests to run in batch, extensive testing can be conducted as regularly as required. Through alerts, unexpected changes to metadata security can be detected and resolved quickly, before they become a potential problem.

Finally, to aid in the area of compliance, metadata security test specifications can form necessarily accurate documentation of what metadata security is required and also, through regular testing, known to be accurately implemented. The logs and reports of test runs can provide auditable evidence that metadata security testing has being extensively and regularly performed.

ACKNOWLEDGMENTS

I'd like to thank Michelle Homes, Ronan Martorell, and Greg Nelson for their generous help in reviewing my paper, and the excellent feedback they provided.

RECOMMENDED READING

- *SAS® 9.4 Intelligence Platform: Security Administration Guide*
  http://support.sas.com/documentation/cdl/en/bisecag/65011/PDF/default/bisecag.pdf
- *SAS Global Forum Paper 376-2011: Best Practice Implementation of SAS® Metadata Security at Customer Sites in Denmark*
  Cecily Hoffritz and Johannes Jørgensen, SAS Institute Inc., Copenhagen, Denmark
  http://support.sas.com/resources/papers/proceedings11/376-2011.pdf
- *SAS Global Forum Paper 324-2010: Be All That You Can Be: Best Practices in Using Roles to Control Functionality in SAS® 9.2*
  Kathy Wisniewski, SAS Institute, Cary, NC
  http://support.sas.com/resources/papers/proceedings10/324-2010.pdf
- *SAS Forum Australia & New Zealand 2010: Best Practices with SAS® 9 Metadata Security*
  Paul Homes, Metacoda
  http://platformadmin.com/blogs/paul/2010/08/best-practices-with-sas-9-metadata-security-sfanz2010/

CONTACT INFORMATION

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

Name: Paul Homes
Organization: Metacoda
Email: paul.homes@metacoda.com
Web: http://www.metacoda.com/
Blog: http://platformadmin.com/
Twitter: http://www.twitter.com/PaulAtMetacoda
LinkedIn: http://au.linkedin.com/in/paulhomes
sascommunity.org: http://www.sascommunity.org/wiki/User:PaulHomes