Securing Ground Data System Applications
for Space Operations

Michael J. Pajevski*, Kam S. Tso†, and Bryan Johnson‡
Jet Propulsion Laboratory, California Institute of Technology,
Pasadena, CA 91109

The increasing prevalence and sophistication of cyber attacks has prompted the Multimission Ground Systems and Services (MGSS) Program Office at Jet Propulsion Laboratory (JPL) to initiate the Common Access Manager (CAM) effort to protect software applications used in Ground Data Systems (GDSs) at JPL and other NASA Centers. The CAM software provides centralized services and software components used by GDS subsystems to meet access control requirements and ensure data integrity, confidentiality, and availability. In this paper we describe the CAM software; examples of its integration with spacecraft commanding software applications and an information management service; and measurements of its performance and reliability.

I. Introduction

The MGSS Program Office manages development of the Advanced Multi-Mission Operations System (AMMOS) software that is used in ground data systems of NASA/JPL robotic space missions. The AMMOS software suite includes applications for mission planning activities, commanding spacecraft, processing/displaying telemetry data, producing science and engineering data products from spacecraft and instrument data, and providing access to science data products (as some examples of AMMOS capabilities). The Deep Space Network (DSN) Project Office manages and is responsible for the entire life-cycle of the DSN. Two examples of the many services that the DSN offers are uplink and downlink capabilities that support spacecraft commanding and telemetry capture, respectively. Protecting capabilities of the AMMOS and DSN is very important to the security of NASA/JPL space missions. Controlling access to important functions and information is crucial to the protection of highly valued spacecraft, engineering data, and scientific information involved in robotic space missions.

To achieve a security solution that meets the needs of their evolving systems, MGSS and DSN management (sponsored by NASA) agreed to collaboratively support the creation and use of common security services. The first element delivered as part of that agreement is the Common Access Manager (CAM). By integrating the CAM into MGSS and DSN products, they can uniformly address the following access control related needs:

1) Suitable granularity – appropriate for each software application and its mission users.
2) Manageable – enable policy changes without development effort or service downtime.
3) Verifiable – facilitate the review of authorization policies to ensure their correctness.
4) Portability – support use of CAM and AMMOS/DSN applications at (or by) any NASA Center.
5) Minimal effort/cost – for application layer access control development, deployment, and operations.
6) Performance – keep up with usage rates of protected applications and information.
7) Reliability – provide availability needed to support time-critical spacecraft operations activities.

Section II describes the CAM software, how it is used to control access for various kinds of applications, and how it supports site portability of applications. Section III covers examples of CAM integration with software applications to minimize their access control related development, deployment, and operations effort/cost. Section IV shows results of performance and reliability testing and analyses performed for the CAM software.

* Systems Engineer, Cyber Defense and Information Architecture, Mission Control Systems, M/S 301-480.
† Software Engineer, Cyber Defense and Information Architecture, Mission Control Systems, M/S 301-480.
‡ Group Supervisor, Cyber Defense and Information Architecture, Mission Control Systems, M/S 301-480.

American Institute of Aeronautics and Astronautics
II. AMMOS Common Access Manager

The Common Access Manager (CAM) is a software assembly in the Application Security (ASEC) subsystem of the AMMOS. AMMOS and DSN software applications use the CAM software to control access to application services and information. Other GDS software applications can also be protected by the CAM software with minimal development effort.

The CAM software provides the following capabilities to protect application layer assets from unauthorized access:

1) Authorization policy management: The ability to manage (i.e., view, create, update, and delete) authorization policies is provided via graphical and command-line interfaces.
2) Single Sign-On (SSO): A capability that allows users to access all applications integrated with the CAM (with restrictions according to authorization policies) with a single application layer log in. Functions are also provided for validating SSO tokens, logging out of SSO sessions, administratively terminating SSO sessions/tokens, and enforcing configurable idle timeouts and maximum session lifetimes.
3) Authorization policy enforcement: The ability to intercept attempts to access protected resources and reject unauthenticated/unauthorized attempts is provided for Web/Application Servers.
4) Authorization checking: Mechanisms to check authorizations are provided for custom applications that include their own enforcement points, but rely on the CAM for authorization management, authentication (with single sign-on), and authorization checking.
5) Identity data retrieval: Applications can use the CAM software to request user information, including username, full name, and group memberships.
6) Integration with institutional and/or project identification and authentication services: Including LDAP, Kerberos, Active Directory, and RSA SecurID.

The authorization policy management capabilities of the CAM software enable customized policy based access control over application functions and information. Policies can be established for any identifiable resource, various kinds of “subjects” (i.e., individual user, group, role, all “authenticated users”, Web Services client, etc.), a customizable set of action types (e.g., read, update, or delete), and various conditions (such as client hostname/address, client network address range, day of week, or time of day). Section III describes the authorization policy schemes for two different applications.

Software applications integrate with the CAM, which provides a consistent interface to applications regardless of the venue specific details of the underlying identification and authentication services. The CAM supports the most commonly used standards-based forms of identification and authentication services (as noted above), and can be configured to use one or more sources of authentication credentials and identity information. The configurable nature of the CAM software, and the abstraction layer it provides for applications, simplifies the deployment of software applications in different venues with the differing identification and authentication services.

Integration with the CAM is made simple for application developers through the use of easy-to-use software components. No programming is needed to support Web Browser clients and most Web/Application Server hosted applications. Software library functions are provided to support integration with custom applications and to implement fine-grained access control (in addition to, or in lieu of, Web interface access control). The examples in Section III describe typical patterns for integrating the CAM software into applications.

The CAM software provides its capabilities through three major components shown in the figure below:

1) CAM Server provides functionality and service interfaces that software applications use to check authentication, get single sign-on (SSO) tokens, validate SSO tokens, check authorization, retrieve identity information, and log the results of access control processing performed by the CAM.
2) CAM Policy Agents intercept service requests (for Web Servers and Application Servers) and filter out unauthenticated/unauthorized access attempts.
3) CAM Software Libraries provide programming interfaces that custom software applications use to call on the capabilities of the CAM Server.

Figure 1 illustrates the major aspects of the CAM software. Protected applications use the CAM software, which is configured to use the identification and authentication services available for the venue.
The CAM software uses secure connections (i.e., SSL/TLS) for all of its service interfaces. The secure connections are provided to ensure that authentication data and other sensitive information cannot be inappropriately accessed in-transit (thus protecting authentication credentials from misuse). The CAM software also provides logging capabilities to capture the results of access control actions performed by the CAM software.

The CAM Server software and Policy Agents are provided by the OpenAM\(^1\) open source software product, managed and supported by ForgeRock. The CAM Software Libraries are developed by JPL to simplify the integration of custom applications with the CAM in a product neutral manner (allowing underlying product used in the CAM to be changed without needing to modify applications that use the CAM Software Libraries). More details of the CAM software can be found in Reference 4.

III. CAM Integration Examples

The CAM software has been successfully integrated with AMMOS and DSN subsystems to provide access control and single sign-on capabilities for a variety of Web, Java, and Python software client and server applications. The CAM reusable components have greatly reduced the effort to implement application layer access control in AMMOS and DSN subsystems.

In this paper we describe two examples of CAM integration with GDS software applications. One scenario demonstrates a typical use of URL-based access control in the Command Preparation & Delivery (CPD) Web Services. The second example shows how to augment URL-based access control with attribute based access control that is tailored to the specific needs of an Information Management Service.

The examples described here represent practical experience integrating the CAM software with applications used in the ground data systems of robotic space missions. The first case below is based on delivered capabilities that are being used by missions. The second case is based on design and development work for future capabilities in the AMMOS and DSN. Lessons learned and potential forward work items are summarized in the examples.

A. CPD Web Services

CPD Web Services are part of DSN’s modernization of its Telemetry, Tracking and Command (TTC) Services. CPD Web Services make it possible for missions to use Web Browsers or mission control tools in the AMMOS Mission Data Processing and Control System (AMPCS) to send commands to spacecraft via space communications ground stations. The CPD Web Services and AMPCS tools support commanding throughout a mission’s lifecycle, from as early as flight software development, through testbed, ATLO\(^1\), and operational phases.

The AMPCS software suite provides mission control, telemetry processing and storage, alarm processing, display of telemetry through graphical and text-based user interfaces, spacecraft data query services and reporting, and automation (as described in Reference 5). The mission control capabilities of AMPCS use CPD Web Services to send commands to spacecraft. The CAM software is used for authentication and authorization checking between AMPCS and CPD.

\(^1\) ATLO stands for Assembly, Test and Launch Operations.
CPD Web Services support manual and automated modes of radiation services. In manual mode, users load command files either from a file store or local disk to the staging area, create radiation requests for the files, and manually control the sending of spacecraft commands to DSN stations for radiation. In automated mode, users prepare radiation sessions that contain a list of command sequence files to be radiated. When the session becomes active, CPD automatically connects to the planned ground station and sends the files associated with the session.

CPD Web Services provide REST (Representational State Transfer) APIs that support inquiring various status parameters, uplinking radiation requests, setting up automated radiation (a.k.a., AutoRad) sessions, and controlling connections to ground stations.

The radiation requests contain the command sequences for operating the spacecraft. Although other safeguards are in place to reduce risks associating with the sending of incorrect commands, such events must be avoided. Thus, it is critical to protect the CPD services from unauthorized access. The AMPCS and CPD Web Services applications use the CAM software to control access to these critical capabilities. Figure 2 below shows the use of the CAM software by the two spacecraft command applications that make up Integrated Command.

![CAM Integration with Spacecraft Command Applications](image)

The figure above shows the use of Java-based client applications (e.g., AMPCS Chill_Up) that use a CAM Software Library (see arrow 2 in the diagram) to authenticate the user and get an SSO Token from the CAM Server (see Arrow 3). The CAM Server uses identification and authentication services provided by the institution (Arrow 4). The client application includes the SSO Token (as a cookie) in HTTP requests it sends to the CPD Web Services (Arrow 5). The Apache Tomcat Server hosting the CPD Web Services calls a CAM Policy Agent (Arrow 6), which uses a CAM Server to validate the SSO Token and check if the user’s request is authorized (Arrow 7). If the user is authorized to make the request, the Policy Agent allows it to proceed to the CPD Web Services. Otherwise, the request is rejected.

The CAM software supports user authentication for various kinds of applications and usage scenarios. The AMPCS custom software client applications need to get SSO Tokens for interactive users and when being used in an unattended/automated fashion. CPD’s Web interface also supports Web Browsers that need to get SSO Tokens.
The following approaches are used by the different CPD clients to authenticate users and get SSO Tokens:

• **Web Browser Client:**
  CPD Web Services provide a Web interface that supports Web browser clients. When someone uses a Web browser to access CPD Web Services, the request is intercepted by a Policy Agent and the browser is redirected to a login page on the CAM Server. Depending on its configuration, the CAM Server can accept RSA SecurID, Kerberos, and/or username/password authentication credentials. Upon successful authentication, a cookie containing the user’s SSO Token will be added to the browser. A browser will automatically include the SSO Token in requests it sends to the CPD Web Services.

• **AMPCS GUI Client Application:**
  The AMPCS GUI Client (Chill_Up) is a Java application that uses CPD to control connections to space communication stations and send selected commands to spacecraft. The client application uses an API in the CAM Java Software Library to get an SSO Token. The Software Library prompts the user for authentication credentials and interacts with the CAM Server to verify the credentials and get an SSO Token (if authentication succeeds). The client application includes the SSO Token in all requests to CPD.

• **AMPCS Automated Client Applications:**
  AMPCS includes two applications that are used in an automated/unattended manner. A status publisher application runs as a background process to continuously obtain status from CPD and provide updates to AMPCS GUIs via a Java Messaging Service (JMS) interface. An application called Chill_Send is executed by scripts (i.e., MPCS Test Automation Kit (MTAK), described in Reference 5) to send commands to spacecraft via CPD automatically to support testing or operations. When the automated applications call the CAM API to get an SSO Token, they specify a Kerberos keytab file to use for authentication. The CAM Software Library uses the keytab file to get a Kerberos ticket that is verified by the CAM Server in order to get an SSO Token. The AMPCS automated applications provide the SSO Token in all requests to CPD just like the other clients. Regardless of the type of client application or authentication credential used, the server-side application receives an SSO Token that verifies that the user/application has been authenticated.

As witnessed from experience with CPD, the CAM software can be integrated with client and server applications with minimal effort. No server side programming is needed for access control because the Policy Agents filter out unauthenticated/unauthorized requests. Web browsers are supported by using their native capabilities. Integration of the Java and Python clients involves only a few simple API calls.

The Policy Agent used by CPD is implemented as a plug-in filter for the Apache Tomcat Server. It intercepts all the HTTP requests as they are received by Tomcat. The agent uses the SSO Token, uniform resource identifier (URI), and HTTP method (e.g., GET or POST) information in each HTTP request to get an authorization decision from the CAM Server (or to find a previous decision in the agent’s cache).

The authorization decisions are based on the evaluation of authorization policies and information about the request and the user making the request. The authorization policies that are managed and enforced by the CAM software consist of three components — a subject, an action, and a resource.

- **The subject** of the policy is the entity being granted permissions to a resource. In this case, the subject is a Lightweight Directory Access Protocol (LDAP) group that represents a mission role. LDAP groups provide a simple and convenient way to assign users to roles, as noted in references 6 and 7.
- **The action** is one of the HTTP methods (i.e, GET, POST, PUT, and DELETE) that can be used in requests to access the CPD REST APIs.
- **The resource** is represented by a URI in the REST architectural style.

To test CAM software integration with AMPCS and CPD, authorization policies were developed for a set of roles and simulated resources. The policies used in testing are based on four roles in order to demonstrate the ability for the CAM software to control access to various CPD services with differing authorization policies.

The following four roles are used in the verification of CAM software integration with CPD:

1) **SCIENTIST:** This role is assigned to mission scientists who write command sequences for controlling the onboard science instruments. The requests from all scientists are stored in the same SCIENTIST pool. They can edit, delete, and flush the requests in that pool but not in the other pool.

2) **SEQUENCE:** This role is assigned to mission engineers who write command sequences for controlling the spacecraft, rovers, landers, etc. Again, they can edit, delete, and flush the requests in the SEQUENCE pool.

3) **ACE:** This role is assigned to mission controllers who are responsible for the radiation of command sequences. They can create, edit, delete, and flush requests in the ACE pool. They also have super power to delete and flush, but not edit, requests of all other pools. They are the only users who can control the DSN stations for radiation.
4) VIEWER: This is a read-only role which is assigned to users to who are only allowed to view the status of the stations and radiation requests.

The following set of policies were defined to specify the access privileges associated with each role:
1) All authenticated users are allowed to GET all resources.
2) The SCIENTIST users are allowed to GET, POST, PUT, and DELETE resources in the SCIENTIST pool.
3) The SEQUENCE users are allowed to GET, POST, PUT, and DELETE resources in the SEQUENCE pool.
4) The ACE users are allowed to GET, POST, PUT, and DELETE all resources.
5) The ACE users are denied the ability to POST or PUT resources in the SCIENTIST and SEQUENCE pool.

This part of the test policies demonstrates how to use “deny rules” to create exceptions to “allow rules”.

Applicable deny rules always take precedence over any applicable allow rules.

Testing of the CAM software integration with CPD has proven successful. The CAM software is able to enforce role-based authorizations for CPD services. The combination of AMPCS, CPD Web Services, and the CAM software has been successfully tested at subsystem and system levels. It has also been deployed into testbeds and ATLO environments for the Soil Moisture Active Passive (SMAP) mission, and will be used to support the mission’s operations.

B. Information Management Service

The Information Management Service in this CAM software integration example is a generalization of services being developed for the AMMOS and DSN to manage mission information and metadata. This example is based on aspects of the Information & Data Management (IDM) and Sequence Revitalization (SEQ-R) subsystems being developed for the AMMOS, and on aspects of the Information Management Service (IMS) being developed for the DSN. The approach described here is conceptual. Details of actual subsystems are beyond the scope of this paper.

An Information Management Service provides access to information and metadata in one or more forms. Information may be in the form of timelines (see references 8 and 9), non-timeline data, files, bundles, packets, etc. Metadata may be represented as attribute-value pairs, triplesets (denoting relationships), etc. Regardless of the form of the information and metadata, typical aspects of information management services such as arranging information in collections and associating metadata with information/collections provide ways to reference protected resources in authorization policies. Typical application integration patterns such as Web interfaces and software library interfaces provide simple integration points for using the CAM software.

The approach for integrating with the CAM software depends on the nature of the application being protected. Applications hosted in Web/Application Servers can be protected by Policy Agents as described in the CPD Web Services example. In some cases, the URL based authorization enforcement capabilities of the Policy Agents are not sufficient to provide fine-grained access control. This example illustrates one of those cases, and expands upon the Web interface access control approach by adding a layer of attribute based access control in order to provide fine-grained access control that meets the needs of an Information Management Service and its mission users.

The Information Management Service in this example arranges information objects hierarchically using namespaces\(^\text{10}\). The following kinds of namespaces are defined for this example:

1) Registered namespace - an administratively created namespace that represents a collection of information objects (and possibly other collections) and is associated with one or more authorization policies.
2) User defined namespace – a namespace that is created by a normal user (i.e., without administrative privileges) and resides within a registered namespace. Access to user defined namespaces (and information objects in them) is controlled by the authorization policy for its parent registered namespace.

Associating authorizations with registered namespaces provides a relatively simple way to define different collections of information in different security categories. If the registered namespaces are completely orthogonal to one another (i.e., there is no nesting of namespaces with different authorization policies) and the URL in each HTTP request sent to the Information Management Service Web interface contains the identifier(s) of the namespace(s) being accessed, then the URL based authorization enforcement capabilities of the Policy Agents can provide sufficient fine-grained control (down to the registered namespace level). However, the Information Management Service in this example supports the nesting of registered namespaces. Doing so does not rule out the possibility of crafting URL based authorization policies that can sufficiently control access to the information, but such policy sets may be complex and difficult to maintain.
Figure 3 illustrates the major aspects of this CAM integration example.

1. Web Interface Access Control
   The Web interface access control mechanisms provide course grained control (e.g., by mission/venue; and normal user vs. administrator) over the services provided by the Information Management Service. User requests coming in through the Web interface (see Arrow 1) are screened by a policy agent that enforces URL based authorization policies by using the services of a CAM Server (see Arrows 2 and 5), which uses Identification & Authentication Services provided by the institution (see Arrows 3 and 4). The Web Services provided by the Information Management Services are not accessed (see Arrow 6) unless the user is authenticated and is authorized to access the requested Web resource.

   The granularity of the authorization policies at the Web interface can vary depending on the needs of the mission. In this example, a simple set of authorizations is used as follows:
   1) All mission users can use service interfaces for accessing managed information and metadata.
   2) Information Service Administrators can use service management interfaces.

   The authorizations associated with user access can be very simple for the Web interface layer when fine-grained access control is performed by a layer of attribute based access control. Keeping the number of policies at the Web interface layer relatively low simplifies policy management (particularly verification and maintenance). Also, a smaller set of policies (for each policy agent) takes less time to be fetched from a CAM Server, consumes less memory in policy agent caches, and requires less time to process during authorization checking.

2. Attribute Based Access Control
   Attribute based access control (ABAC)\textsuperscript{11} is accomplished with the CAM software by using attributes of information/collections to identify resources in authorization policies. In this example, the identifiers of registered namespaces are used to reference protected resources. At run-time, a Policy Enforcement Point (PEP) in the Information Management Service determines the set of registered namespaces affected by a service request, and passes the identifiers of those namespaces (along with the type of action being performed on each namespace and the SSO Token of the user who made the request) to the CAM software in order to get an authorization decision.

American Institute of Aeronautics and Astronautics
When using registered namespace identifiers in authorization policies to refer to protected resources, a new policy (or set of policies) must be created whenever a new registered namespace is created. Doing so involves administrative actions in the Information Management Service to create the registered namespace and administrative actions on the CAM Server to create policies. The separation of the administrative actions provides greater control over the provisioning of new registered namespaces, but can also result in delays when setting up new registered namespaces. In cases in which the set of registered namespaces and authorizations can be defined up-front, registered namespaces and authorization policies can be set up at system deployment time. In cases in which new registered namespaces may be needed well into the operational life of a system, good procedures and people should be in place to facilitate the timely set up of new registered namespaces.

One way to avoid the need to create new authorization policies when new registered namespaces are created is to use an “access tag” instead of the registered namespace identifier in the authorization policies. Using access tags instead of registered namespace identifiers also lowers the number of authorization policies if there are more registered namespaces than access tags (i.e., multiple registered namespaces per information security category). Authorization policies for access tags can be set up in advance of setting up the registered namespaces. When a registered namespace is created, an access tag value is selected for it. At run-time, the access tag values for the registered namespaces accessed by a service request are determined and passed to the CAM software to check authorization. Using registered namespace identifiers to check authorization may be sufficient for most Information Management Service instances, but providing the option to use an access tag instead offers better support for systems that use registered namespaces for more than creating one separate collection for each security category.

Another approach for dealing with the potential need to create authorization policies when resources are created (or modify policies when information objects change state) is the implementation of a CAM software API that enables some integration of authorization management between applications and the CAM software. Implementing a CAM API for managing authorization and implementing capabilities in applications to use those new CAM APIs would require additional development effort, but could provide a more user friendly (and timely) way to manage authorizations for dynamically created and updated resources while maintaining the ability to centrally review, backup, restore, and update authorization policies for all applications that use the CAM. There is the potential for future work implementing policy management APIs and supporting their use by GDS software applications.

Figure 4 shows a sequence diagram for an information service using the CAM software to check authorizations based on resource attributes. Interactions between the CAM Software Library and CAM Server are omitted from the diagram for the sake of simplicity and to focus on the interactions between the application service and the CAM software.

Figure 4. Sequence Diagram for Attribute Based Access Control
After successful authentication and authorization at the Web interface, service requests that involve access to managed information are passed on to the Information Services layer (see Arrow 7 in Figure 3), where fine-grained attribute-based access control mechanisms are applied. The Information Services layer is comprised of software code that processes information management service requests. The request processing path includes a Policy Enforcement Point (PEP) that uses the CAM software to check authorizations associated with service requests. The PEP is a collection of code that is responsible for determining the set of protected resources associated with each request (and the actions being taken on the resources), obtaining the protected resource attributes that are needed to check authorization, constructing authorization decision requests, using the CAM software to get authorization decisions, and enforcing the decisions (i.e., rejecting unauthorized requests).

The PEP retrieves the necessary attributes (i.e., metadata) from the Information Store (see Arrow 8 in Figure 3 and Arrow 2 in Figure 4). The PEP constructs an authorization decision request that includes the SSO Token of the user who submitted the request and a set of “target resource – action pairs”. The “target resource” in each pair is a registered namespace identifier (or an access tag value) and the “action” is the kind of operation being performed (e.g., GET, POST, or DELETE). The PEP uses a CAM software interface (see Arrow 9 in Figure 3 and Arrow 4 in Figure 4) to submit the authorization decision request and get a decision. Optionally, the PEP may check for matching recent decisions in an internal cache; and use matching decisions instead of using the CAM Software Library to get new decisions. Such caching of decisions for some relatively short amount of time (e.g., 5 - 10 minutes) will greatly reduce latencies associated with using a CAM Server (when a user sends numerous requests involving access to the same information) while still being reasonably responsive to authorization changes.

The CAM Software Library uses a CAM Server to get an authorization decision (see Arrows 10 and 11 in Figure 3; not shown in Figure 4). The CAM Server uses Identification & Authentication Services to get identity information (such as group memberships) needed to make an authorization decision. The CAM Server caches identity information (for a configurable period) to minimize latencies associated with using identity information. Access control processing at the application’s Web interface will most likely result in the user’s identity information already being cached by the CAM Server by the time the attribute based access control processing occurs.

The PEP receives the authorization decision(s) from the CAM Software Library (see Arrow 12 in Figure 3 and Arrow 5 in Figure 4). If all of the resource – action pairs are permitted for the user, the request is allowed to be fulfilled. If any of the resource – action pairs are not allowed, the entire transaction should be rejected in order to prevent unexpected results. Allowing partially authorized requests to be fulfilled may be reasonable in some cases (e.g., read-only access) in which partial results are meaningful and not misleading. Examples of such cases are beyond the scope of this paper.

Requests that are authenticated and authorized are forwarded (see Arrow 14 in Figure 3 and Arrow 6 in Figure 4) to the Request Fulfillment code, which performs the requested operations on the information and returns a response back through the Web interface. The policy agent is not involved in the response path.

Registered namespace level access control is likely to cover most cases. A registered namespace can be created for each security category of information, and properly followed mission procedures can keep information associated with the correct namespaces and policies. However, in some cases an additional measure of protection may be necessary. The additional protection can be provided by object level access control.

The same basic mechanisms shown in the diagram above may be used to apply object level control by associating access control metadata with individual information objects, referencing the object level access control metadata in authorization policies, and using the object level metadata to make authorization decisions at run-time. In this case, access tags may be a better choice than registered namespace identifiers because there will be many information objects in each security category. Object level access tags can be used to enforce more restrictive authorizations on objects (e.g., to make them private) within a registered namespace. Such tags can also be used to denote mandatory authorization policies that must be checked in addition to any other applicable policies.

The potential performance impacts of using object level access control is a concern when many objects that have their own authorizations are accessed to fulfill a service request. Using object level access control sparingly can minimize the performance concern. Using separate registered namespaces for different authorization policies instead of enforcing object level authorizations avoids the concern, but constraints the way that information can be arranged. Using other mechanisms, such as database level access controls or a custom application-specific solution, may provide better performance for object level control; but implementing such mechanisms is likely to increase the cost and complexity of the software and its usage.

Performance and reliability are valid concerns for any software set that supports critical mission operations. These aspects of the CAM software have been characterized for a typical usage scenario. Results of that characterization are presented in Section IV of this paper.
IV. Performance and Reliability Evaluation

It is important that the CAM software does not impose unreasonable performance overhead or decrease the reliability of the applications that rely upon it. Thus, stringent requirements on performance and reliability are imposed on the CAM software. A test tool suite has been developed to verify the performance and reliability requirements. The suite consists of the following components:

- The **Load Generators** use multiple test users to make requests to the CAM Server or Agent at the specified rates. There are four distinct load generators each targets the CAM Java API, Python API, J2EE Agent, and Web Agent, respectively. When the test starts the load generator reads a list of test users and passwords and spawns a thread to run the tasks of each user. For the API load generators the tasks use the respective Java and Python CAM API to make requests to the CAM Server, thus measuring both the performance and reliability of the CAM API and the CAM Server. For the J2EE agent load generator the tasks make HTTP GET, POST, PUT, and DELETE requests to an agent-protected demonstration CPD server application hosted on Tomcat. Similarly, the Web agent load generator makes HTTP GET requests to static web pages hosted by an Apache server protected by a Web agent. The load is controlled by the number of test users and the time between requests. The length of the test depend on the number of iterations the tasks are run. The result and timing of the response to each request are logged in a data file for performance and reliability analysis.

- The **Test Analyzer** verifies the correctness of the results and collects the timing information to compute the throughput and average response time of each type of tasks and and overall result among all the tasks.

- The performance and reliability tests are run by a **Test Script** which varies the parameters for the number of users, user pace-in time, user think time, and number of iterations to obtain different settings of the tests.

A. Performance Evaluation

The timeliness requirements imposed on the CAM software are summarized in the Table 1.

<table>
<thead>
<tr>
<th>Request Type</th>
<th>Response Time (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication</td>
<td>500</td>
</tr>
<tr>
<td>Authorization (non-initial)</td>
<td>20</td>
</tr>
<tr>
<td>User Profile</td>
<td>50</td>
</tr>
<tr>
<td>Validate SSO Token</td>
<td>50</td>
</tr>
<tr>
<td>Invalidate SSO Token</td>
<td>20</td>
</tr>
</tbody>
</table>

For throughput, the CAM software must be capable of processing at least 500 subsequent (non-initial) authorization decision requests per second. It should be noted that achieving the specified performance depends on having hardware that meets the minimum system requirements — a dual-core CPU and 4GB of memory. The authorization API performance numbers are specified and measured for non-initial requests in order to characterize steady state performance and factor out the effects of loading classes, priming caches, and making initial connections.

Our test results show that the CAM Software Library APIs, Policy Agents, and Server meet the performance requirements. Figure 5 and Figure 6 show the timeliness and throughput performance of the Java and Python API tests with a setting of user pace-in time of 0.1 second and user think time of 0.05 seconds. The short think time was used to generate a high rate of requests to the CAM Server in order to achieve the specified throughput.

Figure 5 shows the average response time increases rapidly with increased number of users. The throughput shown in Figure 6 increases as more requests hitting the CAM Server but it starts to peak toward 100 users as the requests take longer to process. The results also show the Java and Python APIs have similar performance since the determining factor is the performance of the CAM Server, which is the same for both tests.
B. Reliability Evaluation

The reliability requirements imposed on the CAM software specify 99.9% availability and a mean-time between failure (MTBF) of 90 days. For testing, we use a compressed reliability test to shorten the test time. The intention of the compressed reliability test is to compress the 90-day operations into 3 days, implying a scaling factor of 30. The test assumes “nominal operation” to be a 16-hour work day with 20 users each making one request per second to a Web server application, and concurrently another 20 users each making one request per second to a Tomcat Java EE application.

With a scaling factor of 30, each test runs for 32 minutes with requests being made 30 times per second. After each test run is done, the test application sleeps for 16 minutes. The cycle repeats 90 times to simulate the 90 days of operations. After the test is completed the correctness of the responses of all the requests are verified.
The compressed reliability tests have been run a number of times, simulating well over a year of continued operation, and the reliability has been 100%. All errors identified in the test results were found to be the result of problems with the test set up, the network, or the LDAP service. That is not to say that our experience does not include any issues with the underlying product in our solution, but it has proven its ability to deliver the necessary capabilities (at minimal cost) and operate continuously for long periods of time.

V. Conclusion

The Common Access Manager (CAM) software is a useful tool for easily incorporating access control into software applications used in ground data systems. The CAM software is well suited for Web-based applications, and the CAM software capabilities can also be utilized to provide fine-grained attribute-based access control.

The capabilities of the CAM software meet the needs noted at the top of this paper. The cost effective solution provided by the CAM software offers centrally managed and reviewable control over access to protected resources, simplifies site portability of applications, and achieves required performance and reliability results. However, there are potential future work items for enhancing the capabilities and performance of the CAM software.

While the centralized authorization policy management mechanisms of the CAM software simplify the deployment and maintenance of authorization policies for a number of applications in a venue (or set of venues), the separation of authorization management from information management services can interfere with the timely creation of policies for new resources. There is potential future work in the creation of CAM APIs for managing authorizations so that applications can interface with the CAM to automatically establish policies for new resources (as permitted by the current set of authorization policies).

Custom software applications that use the CAM Software Library APIs may need to implement decision caching, if necessary to support very high timeliness and throughput rates. A potential future area of work is the caching of access control related information in the CAM software. Other potential future work related to performance and reliability includes the use of load balancing and hot swap capabilities.

Acknowledgments

The work described in this paper was funded by the JPL Multimission Ground Systems and Services Office and performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

References

Web Sites

Proceedings

Special Publications