

Access Control of Web and Java Based Applications

Kam S. Tso, Michael J. Pajevski, and Bryan Johnson
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109

Abstract—Cyber security has gained national and international attention as a result of near continuous headlines from financial institutions, retail stores, government offices and universities reporting compromised systems and stolen data. Concerns continue to rise as threats of service interruption, unauthorized access, stealing and altering of information, and spreading of viruses become ever more prevalent and serious. Controlling access to application layer resources is a critical component in a layered security solution that includes encryption, firewalls, virtual private networks, antivirus, and intrusion detection. In this paper we discuss the development of an application-level access control solution, based on an open-source access manager augmented with custom software components, to provide protection to both Web-based and Java-based client and server applications.

I. INTRODUCTION

Cyber security has gained national and international attention as a result of near continuous headlines from financial institutions, retail stores, government offices and universities reporting compromised systems and stolen data. Concerns continue to rise as threats of service interruption, unauthorized access, stealing and altering of information, and spreading of viruses have become more prevalent and serious [1].

The Multimission Ground Systems and Services (MGSS) Program Office at Jet Propulsion Laboratory (JPL) has funded the DISA Security Service Task to develop capabilities for protecting ground data systems as part of addressing cyber security threats identified in a recent security risk assessment [2]. This paper describes a prototype constructed to prove out concepts for providing a common access control solution that protects both Web-based and Java-based client and server applications.

A. Background

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 planning missions, commanding spacecraft, processing/displaying telemetry data, producing science and engineering data products from instrument data, providing access to science data products, and many other

functions [3]. Controlling access to these important functions and information is crucial to the protection of highly valuable spacecraft, engineering data, and scientific information involved in robotic space missions.

The Deep Space Network (DSN) Program Office at JPL manages the DSN space communications infrastructure. The DSN includes the uplink and downlink capabilities that support spacecraft commanding and telemetry processing. Protecting these capabilities are is very important to the security of NASA/JPL space missions.

The MGSS and DSN Program Offices are jointly defining the Deep Space Information Systems Architecture (DISA) [4], which describes an application software integration architecture and infrastructure services that include the DISA Security Service. The goal of DISA is to modernize the DSN and AMMOS through the application of industry best practices in software architecture. The goal of the DISA Security Service is to provide a robust and cost-effective security infrastructure that supports software applications and the other infrastructure services (e.g., messaging, registries, and information repositories). The goal of this paper is to describe first steps in the development of access control capabilities to be provided by the DISA Security Service.

B. Access Control

Access control seeks to restrict access to protected resources by enforcing policies that state which subjects can perform defined actions under known conditions on approved resources [5]. Access control works at a number of levels: hardware, operating system, middleware, and application. In this work, our focus is on protecting network-accessible application layer resources, which include application, web content, and web services. Two common access control models are the role-based access control (RBAC), which restricts access based on the roles assigned to users [6], and attribute-based access control (ABAC), which is based on attributes associated with users [7]. In both models, users are authenticated, their roles or attributes are retrieved, and the retrieved roles/attributes are compared against authorization rules in order to make authorization decisions that are enforced by the access control system.

As access control plays an important role in protecting business applications, many access management products are marketed by enterprise software vendors and are also developed by the open-source community. They provide authentication and authorization services that usually include:

- centralized management of policy information,
- policy enforcement and policy decision,
- immediate effectiveness of role, attribute, and policy changes,
- single sign-on over heterogeneous applications and services, and
- federated sign-on across trusted networks of partners.

II. DISA SECURITY SERVICE ARCHITECTURE

The DISA Security Service (DISA-SS) provides common access control capabilities for AMMOS software applications through a set of application programming interfaces (APIs) and network-accessible security services for authentication, single sign-on, authorization checking, and authorization policy management.

The DISA-SS utilizes on institutionally provided credential and identity services in order to leverage well-defined and robust processes for vetting identities and issuing credentials that meet the exacting standards required by NASA. During this first phase of prototyping the DISA-SS architecture, only one identification and authentication service is employed. Future prototyping will demonstrate the use of identification and authentication services provided by multiple institutions through the use of trust relationships and federation techniques.

A. System Composition

The DISA-SS consists of the following components.

- 1) *DISA-SS Core* provides most of the DISA-SS functionality. The core is comprised of two major parts:
 - Access Management Core provides functionality and network-accessible interfaces that software applications can use for checking authentication, checking authorization, retrieving identity information, getting single sign-on (SSO) tokens, and validating SSO tokens issued by the DISA-SS. The Access Management Core also provides an administration console (i.e., graphical user interface) and command line tools for configuring the Access Management Core and for managing authorization policies.
 - Audit Management Core provides network-accessible interfaces for accepting event data from software applications (and the other parts of the DISA-SS). This part of the DISA-SS also provides functionality for automatically analyzing event data (i.e. audit data) as it arrives at the DISA-SS and event data that is stored by the DISA-SS. It also provides graphical user interfaces for

configuration of analysis processing and event notification, managing audit data, and creating audit data reports.

- 2) *Software Libraries* provide application layer security capabilities for custom software applications via a set of application programming interfaces. The software libraries provide interfaces that applications can use to call on the capabilities of the DISA-SS Core for authentication, SSO, authorization, and logging.
- 3) *Policy Enforcement Agents* intercept service requests (for Web Servers and Application Servers) and filter out unauthenticated/unauthorized access attempts.

B. Example Topology

Figure 1 illustrates an example conceptual topology that includes the DISA-SS components described above, protected resources (on Web Servers, Application Servers, and in a Thick Client and Service), and institutional services that include one or more credential and identity services.

The personnel shown in Figure 1 include:

- 1) Application Users: Human users of software applications.
- 2) Security Service Administrator: Personnel responsible for configuring DISA-SS to use the appropriate institutional services, monitoring its operational status, and using its management capabilities for authorizations, event notifications, and audit data management.
- 3) Resource Manager: Personnel responsible for setting up authorizations used at run-time to control access to application resources. Developers may perform this role during development of their software while operations personnel perform the role in an operations environment.

III. DISA SECURITY SERVICE PROTOTYPE

A prototype was constructed to prove out the architecture described above. The prototype is based on an open-source access manager (which provides the Core and Policy Agent portions of the solution) and custom-developed software libraries.

Many access management products are designed with a focus on web portals and web applications, as shown in Figure 2. The web application is hosted either on a web server such as Apache HTTP, or an application server/container such as Apache Tomcat. A policy agent, which is a component of the access management product, is installed on the server host to protect the web application. Thus the policy agent acts as the policy enforcement point (PEP) in the typical access control architecture.

When a user or an external application requests access to the web application, the policy agent intercepts the request and enforces authorization policies. The enforcement is facilitated with a single sign-on token which travels with the request in a cookie. If the authentication token is not found

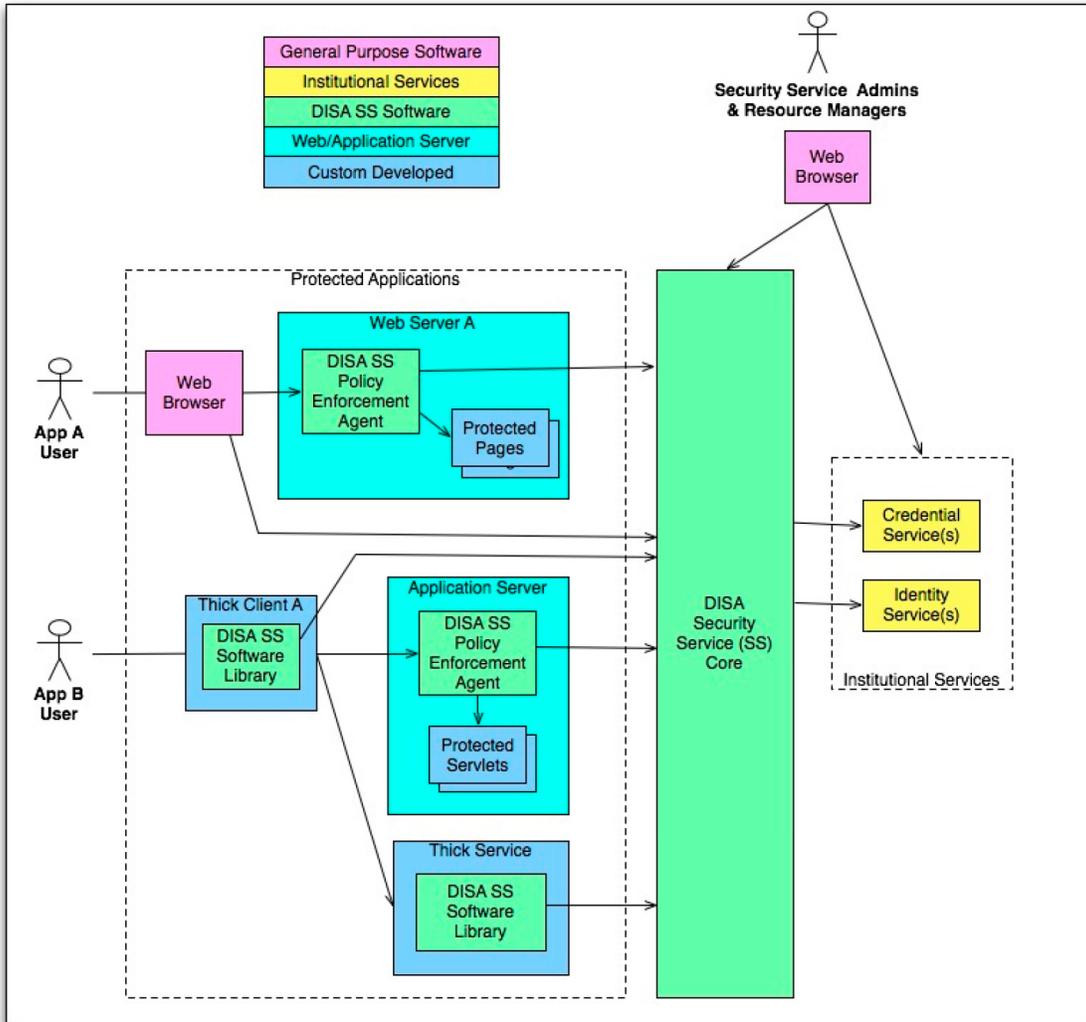


Figure 1. DISA Security Service Example Topology

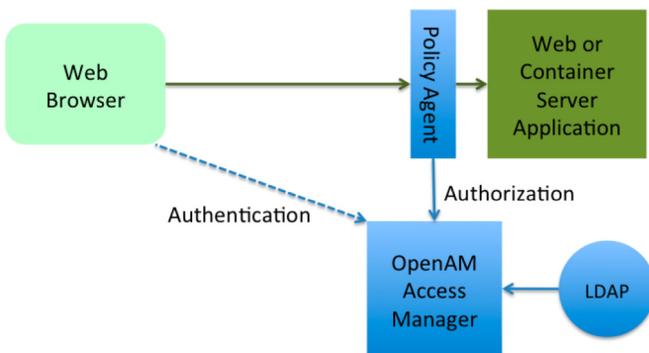


Figure 2. Access Control of Web-Based Application

Access Manager to obtain authentication for the request to proceed.

In Figure 2 a Lightweight Directory Access Protocol (LDAP) server is used for authentication. Access managers can also be configured to use other authentication modules such as Kerberos and Active Directory. Upon successfully authentication a cookie with the SSO token is returned to the browser. This cookie will then be available to the policy agent in subsequent requests. The policy agent sends the SSO token, requested URI, and action to the Access Manager to obtain an authorization decision. This design allows access control decisions to be made by the Access Manager as a policy decision point (PDP) instead of that logic being coded into applications. The Access Manager evaluates the request against the set of defined policies in making the decision of allowing or denying access to the

in the request, the policy agent redirects the request to the

resource.

Most access managers also act the policy administrative point (PAP) in that they provide a web console for administrators to manage authorization policies. A policy defines the rules that specify a user's access privileges to a protected resource. A policy usually comprises the following items:

- **Rules:** A rule contains the universal resource identifier (URI) of the protected resource, an action which operates on the resource (e.g. GET and POST), and a value defines the permission for the action (e.g. Allow or Deny).
- **Subjects:** A subject could be a user, group, role, or application requesting the access.
- **Conditions:** A condition allows constraints to be defined on a policy, such as time of day and IP address from which access is made.

IV. DISA SECURITY SERVICE REUSABLE COMPONENTS

Access managers work out-of-the-box for Web applications but not for thick clients or standalone servers developed in C/C++ or Java. Unlikely Web browsers, thick clients do not support HTTP redirection nor handle cookies unless they implement those web browser functionalities. On the other hand, no known policy agents are available for protecting standalone servers. In anticipation of the needs of these custom applications, most access managers provide a Software Development Kit (SDK) for applications to access their authentication and authorization services. Some of them also expose the services as RESTful web services. REST, which stands for Representational State Transfer, is an architectural style which uses the HTTP operations, GET, POST, PUT, and DELETE, in order to act on resources, represented by individual URIs [8].

The DISA Security Service makes use the RESTful web services to develop reusable components to enable think clients and standalone servers for access control. The DISA-SS components not only simplify the interface to the underlying Access Manager, but also simplify the ability to swap it out with another similar access management product.

The DISA Security Service reusable components are implemented in Java and include the following packages:

- **Credential**
The Credential package is used to establish the user's identity. Currently it supports the basic username and password credential, as well as X.509 certificates and Java Keystores.
- **Connection**
The Connection package is used to establish secure SSL/TLS HTTP connection to the application servers and REST client connection to the Access Manager. The package also include socket and database connections over SSL/TLS. The secure connections ensure transmitted data are encrypted using the cipher suites

recommended by the National Institute of Standards and Technology (NIST) [9].

- **Access Control**
The Access Control package provides Java application programming interfaces (APIs) for authentication and authorization services. It uses the Credential and Connection packages to communicate with the Access Manager to provide such services.

The DISA Security Service components have been used in prototyping two types of applications, custom Java thick clients for accessing resources protected by a policy agent, and standalone Java servers protecting network resources.

A. Thick Client Application

Most of the applications in JPL Advanced Multi-Mission Operations System (AMMOS) are not web-based but custom Java applications [3]. Some of the server side applications provide access to data through RESTful web services or HTTP protocol and they are hosted by Java EE containers. This type of server applications provides the opportunity for resources to be protected by the policy agent. However, the clients are usually written as thick Java applications to meet the requirements of rich graphical user interface and visualization capabilities [10]. Unlikely web browsers, these client applications do not have built-in capabilities for responding to HTTP redirection, handling cookies, and displaying login prompts via a web page as expected by the access management products. The DISA-SS reusable components allow these clients to access the protected resources without implementing those browser functionalities.

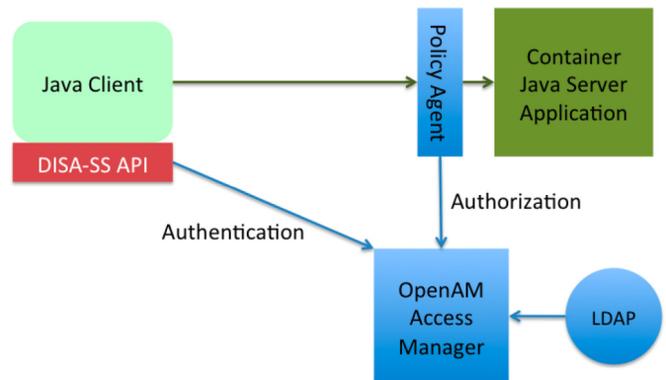


Figure 3. Access Control of Think Client and Container Server

Figure 3 shows how a Java client makes use of the DISA-SS APIs to work under the access control environment. In this configuration the server application is protected by a policy agent. The benefit of using the policy agent is that access control to resources is externalized, thus the server application does not need to make any code changes for or even aware of access control as needs change. It

is all handled by the agent and the access management infrastructure.

In this use case the main flow of events are:

- 1) The client obtains credential from the user, which currently is the username and password.
- 2) The client uses the DISA-SS authentication API to send the credential to the Access Manager for authenticating the user.
- 3) Upon successfully authentication an SSO token is returned to the client.
- 4) A HTTP cookie is created from the SSO token and sent with the request to access protected resources.
- 5) The policy agent obtains authorization decision from the Access Manager using the cookie.
- 6) If access is granted, the request is executed by the server application and responds with the result. Otherwise, the policy agent sends back an error message.

B. Standalone Server Application

Standalone server applications that do not run under Java EE application containers or servers are commonly used in AMMOS. There is no policy agent supporting these standalone server applications as access management products only provide agents to certain Java EE application containers and servers.

Figure 4 shows how a standalone Java server application uses the DISA-SS reusable components to protect itself from unauthorized access by a thick Java client application. In this use case the main flow of events are as follows.

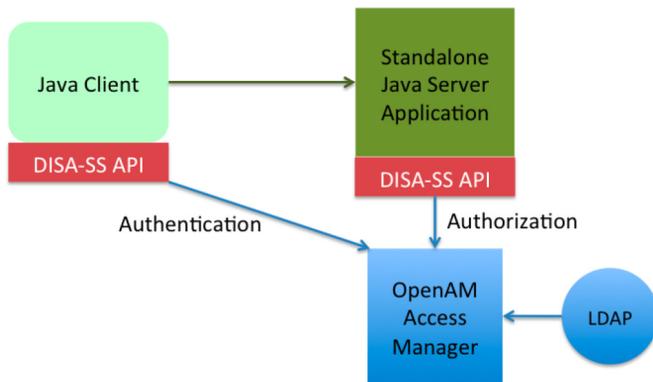


Figure 4. Access Control of Thick Client and Standalone Server

- 1) The user of the client will first be authenticated with the Access Manager using the DISA-SS authentication API, as described in the thick client use case.
- 2) After obtaining the SSO token upon successful authentication, the client passes the token to the server application whenever a request is made. How the token passes to the server is dependent on the type of the connection between the client and the server. If it is a HTTP connection the token can be passed as a

cookie. For other types of connection, such as socket or Remote Method Invocation (RMI) calls, the token can be included as part of the data sent to the server application.

- 3) The server application, after receiving a request with the SSO token, can get an authorization decision from the Access Manager using the DISA-SS authorization API. This API takes three parameters: SSO token, resource, and action. The resource is usually represented by a URI but in this case it can be any string value since the server is not a web application. The action can also be any string value decided by the server application based on the request.
- 4) The Access Manager makes the decision based on the subject, resource, and action. The subject is obtained from the SSO token. If the Access Manager can find a policy that matches the subject, resource, and action, it can grant access to the protected resource. Since the matching is based on comparing string values of the requested resource and the policy resource, we can assign any value appropriate to the application.

V. CONCLUSIONS

In this work we successfully demonstrated that we can extend an open-source access manager designed for web applications to meet the needs of Java thick clients and standalone servers that are commonly used in the JPL AMMOS environment. The DISA-SS reusable components will greatly reduce the effort for each AMMOS subsystem to develop its own access control strategy.

Through this prototyping effort we have found several shortcomings in the open-source access manager we evaluated:

- The Access Manager compares the policy resource URL to the requested resource URL to determine policy decision. Although wildcards are supported in matching the URL strings, variables and operations are not supported, making it impossible to define sophisticated policies. For example, if we want a policy to allow only the user who created a resource to delete that resource, we have to create a policy for each user.
- The performance of the Access Manager is a concern for some applications, such as message queue that could be required to process thousands of messages in a second. Fast policy decisions cannot be met if round-trips to the Access Manager are required frequently. Policy decisions can be cached by the policy agent, but under the current implementation, even a policy that is applied to a group of users still needs a new decision when accessing the same resource.

Despite of these shortcomings, our evaluation has shown the open-source Access Manager to be a promising access management product. We plan to continue the evaluation with other authentication methods such as Kerberos, digital

certificates, and RSA SecurId. We also plan to develop the DISA-SS reusable components in other programming languages such as Python and C++.

VI. ACKNOWLEDGEMENTS

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. The authors would like to acknowledge contributions from members of the DISA-SS team: Dave Childs, Gary Ramah, Henry Hotz, and Jonathan Jaffe.

REFERENCES

- [1] T. Kellerman, "Cyber-threat proliferation: Today's truly pervasive global epidemic," *IEEE Security and Privacy*, vol. 8, no. 3, pp. 70–73, May 2010.
- [2] A. Ko, K. Tan, and F. Cilloiz-Bicchi, "Cyber threat assessment of uplink and commanding system for mission operation," in *Proc. 17th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2011)*, Pasadena, CA, Dec. 2011.
- [3] A. Ko, P. Maldague, D. Lam, T. Bui, and J. McKinney, "The evolvable Advanced Multi-Mission Operations System (AMMOS): Making systems interoperable," in *Proc. AIAA SpaceOps 2010 Conference*, Huntsville, AL, Apr. 2010.
- [4] Jet Propulsion Laboratory. (2011) DISA: Deep space information services architecture. [Online]. Available: <http://disa.jpl.nasa.gov/>
- [5] R. Sandhu and P. Samarati, "Access control: Principle and practice," *IEEE Communications Magazine*, vol. 32, no. 9, pp. 40–48, Sep. 1994.
- [6] D. Ferraiolo and D. Kuhn, "Role based access control," in *Proc. 15th National Computer Security Conference*, Baltimore, MD, Oct. 1992, pp. 554–563.
- [7] D. Kuhn, E. Coyne, and T. Weil, "Adding attributes to role based access control," *IEEE Computer*, vol. 43, no. 6, pp. 79–81, 2010.
- [8] F. Rosenberg, F. Curbera, M. J. Duftler, and R. Khalaf, "Composing RESTful services and collaborative workflows," *IEEE Internet Computing*, vol. 12, no. 5, pp. 24–31, Sep. 2008.
- [9] C. M. Chernick, C. Edington III, M. J. Fanto, and R. Rosenthal, "Guidelines for the selection and use of transport layer security (TLS) implementations," National Institute of Standards and Technology, Gaithersburg, MD, NIST Special Publication 800-52, Jun. 2005.
- [10] L. Hall and P. Francel, "Multi-mission technical subsystem management measures taken and lessons learned," in *Proc. Aerospace Conference*, Big Sky, MT, Mar. 2011.