Using XML For Platform Security

Profiling SAML and the GSS-API

Gerald **Beuchelt**

Abstract

The Security Assertion Markup Language (SAML) [SAML] has various capabilities to provide authentication, attribute and limited authorization information: Users that authenticate to a SAML Identity provider (either transparently or directly) receive an XML token that carries authentication and other information. This token can be used to authenticate against other services that trust the respective Identity Provider.

While SAML is typically being used for Web Single-Sign-On (SSO) scenarios, there is no principal restriction of SAML to such application. If a SAML assertion (or more generally speaking any XML security token or assertion) was presented appropriately to the authentication and authorization subsystem of an e.g. operating system, the token could be used for identity operations within the OS and its applications. .

The Generic Security Service (GSS) API [GSS-API] defines a generic token for binary security tokens. This token format can be used to transparently provide identity related information to a service provider within the OS. GSS-API technology is currently implemented to a varying degree in Windows and some brands of UNIX like e.g. Solaris. By defining a GSS-API binding for XML security tokens in general and SAML 2.0 Authentication Statements in general, one can start utilizing such tokens for platform-level authentication.

Table of Contents

1. Introduction

SAML [SAML] provides a rich and extensible set of assertions that are being used for web application and web services authentication and authorization.

Single Sign On and transparent authentication between client and server systems is a high priority in today's computing environments: users and administrators have a hard time dealing with isolated and disconnected user databases which result in proliferation of user accounts, insecure passwords, and many lost or forgotten passwords and account names. In an ideal world, users would simply authenticate once to the local computer system and this authentication result could be reused for all following authentication requests from other systems, such as file servers, portals, etc.

Today there are already a few technologies which allow such transparent authentication and SSO within the limits of controlled environments such as the workgroup or enterprise network. Most solutions fail however when they are required to scale beyond the enterprise.

On the other side, there exist a few mechanisms and products (such as e.g. Liberty Alliance based technologies) that allow SSO to web sites on an Internet scale. Unfortunately, these technologies are not used to perform platform (operating system) level authentication, but only for web based applications and services.

Combining the ubiquitous cryptographically strong existing security frameworks like the GSS-API (or SASL - the Simple Authentication and Security Layer) with the features that SAML can offer, a large number of new use cases can be addressed.

2. Protocols

2.1. Security Frameworks

In order to enable widespread use of the distributed computing model, the required connections between to separate systems need to be trusted by the participating parties. Such trust can only be warranted, if such connection fulfills a few minimal requirements:

Integrity
: Messages or streams transported on the communication channel must not be altered or interfered with during transmission. This requirement addresses a number of threats, among them the man-in-the-middle attack scenario. It is typically satisfied by cryptographically signing the content.

Confidentiality
: The content might need to be protected from being read during transmission. Especially sensitive information must never be available in cleartext. An interesting problem that arises in the context is the problem that end-to-end protected information (i.e. only the initiator and the final recipient are capable of decrypting the ciphertext) can - obviously - not be examined for routing information. A possible solution to this problem can be to rely on two layers of protection.

Frequently other security requirements may arise (such as e.g. non-repudiation), but these will not be addressed in this paper.

In order to provide security for distributed transactions, a variety of security mechanisms has been established. All of these have in common that they can use different cipersuites and provide for proper key management. As a result of the proliferation and adoption of a number of different security mechanisms, the Internet Engineering Task Force (IETF [http://www.ietf.org/]) provided from early on security frameworks that can utilize different security mechanisms in a unified way.

The GSS-API [GSS-API] is a standard mechanism for presenting tokens in a client-server environment. It allows transparent authentication and authorization for network based applications. While its original emphasis is on the API calls used to perform the token exchange, it also defines the actual token format that is used in such an exchange. This token header format is based on the ASN.1 [X.680] mechanism, using the DER encoding rules, while the payload can be encoded in any way.

The GSS-API is implemented in a very large number of operating systems. This includes Solaris, other UNIX based operating systems and Linux. Also, while Microsoft Windows does not support the actual API calls, it implements the token exchange protocol and marshaling mechanism in its own SSPI interface.

There is already a large variety of mechanisms defined for use within the GSS-API framework. Also, newer extensions of the core GSS-API framework allow the "stacking" of different mechanism and - as such - a very high degree of composability.

2.1.1. GSS-API

The GSS-API [GSS-API] is a standard mechanism for presenting tokens in a client-server environment. It allows transparent authentication and authorization for network based applications. While its original emphasis is on the API calls used to perform the token exchange, it also defines the actual token format that is used in such an exchange. This token header format is based on the ASN.1 [X.680] mechanism, using the DER encoding rules, while the payload can be encoded in any way.100%

The GSS-API is implemented in a very large number of operating systems. This includes Solaris, other UNIX based operating systems and Linux. Also, while Microsoft Windows does not support the actual API calls, it implements the token exchange protocol and marshaling mechanism in its own SSPI interface.

There is already a large variety of mechanisms defined for use within the GSS-API framework. Also, newer extensions of the core GSS-API framework allow the "stacking" of different mechanism and - as such - a very high degree of composability.

2.1.2. SASL

The Simple Authentication and Security Layer (SASL - [SASL]) is another security framework by the IETF that addresses similar needs as the GSS-API. SASL is, however, stream-oriented, while the GSS-API is message oriented.

A very interesting feature in SASL is that it has a GSS pseudo mechanism, that allows it to use any security mechanism defined in the GSS-API. As such, SASL can be treated as being on a higher level in the protocol stack, effectively using the GSS-API below.

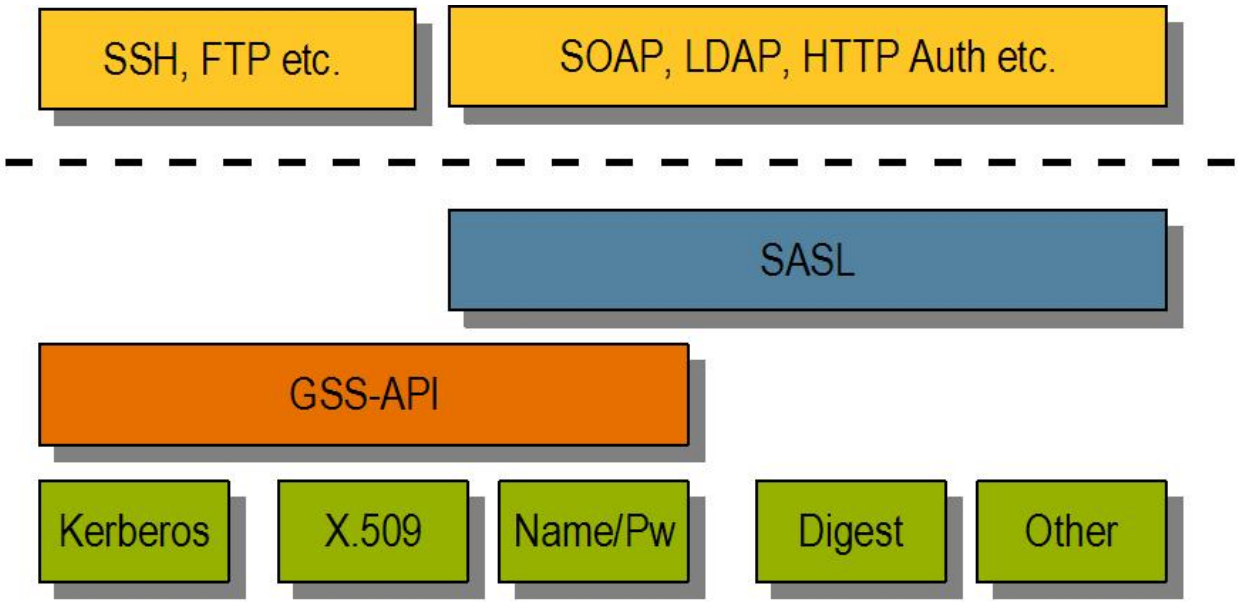

Figure 1. Network Protocol Stack

2.2. SAML 2.0

SAML (Security Assertion Markup Language) is a XML based protocol that defines a security token format and some default transport protocol bindings. SAML's primary application is in the web service and web application area, where it is typically used to achieved Single-Sign-On (SSO) for different web sites. SAML is the preferred token format for the Liberty Alliance specifications.

SAML defines three primary security assertions:

Authentication Statement	The Authentication Statement confirms that the holder of this claim was authenticated in some way at a defined time. It may contain cryptographic keys in the form of XML Signature `<ds:keyInfo>` elements. The SAML Authentication Statement can - when properly profiled - replace the security token of traditional security services, such as Kerberos v5.
Attribute Statement	The attribute statement relates one or more attributes to an assertion subject.
Authorization Decision Statement	While the authorization decision is still contained within the SAML 2.0 specification, it is marked as deprecated. Authorization decisions should be handled by XACML (eXtensible Access Control Markup Language) which also has a authorization policy rules engine specified in the core protocol.

3. Motivation and Use Cases

3.1. Motivation

Over the past few years, XML processing has become very common and is - in many cases - the enabling technology for a new interoperability level on a higher layer in the protocol stack. This manifests itself in a large variety of technologies, including SOAP web services, ebXML business-to-business protocols and many more. This explosion of

new extranet and Internet technologies with sometimes very loose coupling made clear that the industry needs an interoperable, federated identity system, that scales beyond the enterprise or home networks.

At the same time, platform level security interoperability has not seen any radically new approaches or improvements over the past few years: simple authentication interoperability is possible for a variety of mechanisms. Yet, these security protocols are not at all extensible or only in a fairly limited way (Kerberos v5) or have other limitations (X.509).

Combining the flexibility and extensibility of the XML based security tokens with the speed and efficiency of the traditional security mechanisms can lead to a fresh wave of platform level security interoperability technologies and products, that can finally span across different system architectures without loss.

3.2. Use Cases

3.2.1. Cross-Technology Use of Security Tokens

When creating complex distributed applications, the developer has a wide choice of technologies. In today's world, he can e.g. employ XML based web services, RPC mechanisms and more traditional client/server approaches. Often enough, applications use a mix of these technologies, mixing e.g. XML web Services and RPC mechanisms.

While efforts to achieve a reasonable Single Sign One (SSO) solution for XML Web Services are well progressing, SSO through technology tiers (*e.g.* from Operating System to Web Applications or Web Services and *vice versa*) is currently not on the roadmap in most cases. This results in different authentication and authorization databases for the respective technology domains and in multiple authentication processes during a single transaction. Besides being computationally expensive, this issue also raises security issues like the need to protect multiple databases, synchronization of data etc.

GSS-SAML would be able to address at least some of these issues by enabling the XML based SAML token to provides sufficient information to perform all authentication and authorization across technology boundaries. In a likely scenario, a web application requires SAML based authentication. The end-user would through e.g. Liberty Alliance technologies be able to provide the web application execution environment with a SAML token that allows authenticated and authorized access to the web application. If during execution the web application needs to access e.g. a file on a remote file share or a database, the SAML user token can be properly wrapped within the GSS-API framework are presented to the back-end service. The token could then be evaluated for authentication and authorization privileges by the back-end application. The obvious benefit in this scenario is the need for only one account database and the use of a single security token for the entire transaction.

3.2.2. SASL Scenarios

There is a growing interest in using SAML in highly distributed grid applications (see http://grid.ncsa.uiuc.edu/GridShib/). In this scenario SASL is used as the security framework for exchanging the token. Also, LDAP and the Liberty Alliance specifications make heavy use of SASL as the framework for security token exchange.

3.2.3. Using SAML Assertions as Platform Tokens

Security tokens play a crucial role in securing distributed computing by communicating authentication and authorization information for user processes within a computing system. Solaris (similar to practically all other UNIX-like operating systems) frequently use name/password mechanisms for authentication and a simple UID/GID mechanism for authorization. Windows establishes a more complex authorization scheme by using SIDs for identifying group membership. Security tokens (such as e.g. Kerberos tickets) can carry name information and authorization data. SAML tokens could be used through the GSS-API framework to perform seamless authentication and authorization to the operating system. E.g. instead of sending a Kerberos ticket through the GSS-API, one could send a SAML token for identity purposes. Since SAML tokens are highly extensible, a single token could be used for authentication to platforms are different as Solaris and Windows.

SAML assertions carrying attributes beyond the name can be used in a large variety of cases. Additional attributes are ideally suited to denote group membership or other authorization relevant data. When using a limited subset of the available attributes for a particular identity, such a SAML assertion can be used in scenarios where privacy is important, effectively providing platform level support for authenticated and authorized, yet pseudonymous access to system resources.

4. Toward a GSS-SAML Mechanism

There are a variety of options on how to combine the GSS-API with SAML, ranging from a fresh start (defining a completely new GSS mechanism) to an embellishment of existing GSS mechanisms with SAML constructs.

4.1. Decoration

In this case, one would use existing GSS-API mechanisms to do the actual authentication, key exchange and per-message tokens, but also transport SAML assertions protected by the underlying mechanism. This would still allow transporting SAML authentication statements and additional attributes derived from the authentication process such as e.g. group membership, images, roles etc. This can be achieved in two ways:

External Decoration
: Using a newly proposed extension to the GSS-API, GSS mechanisms are stackable. This means that an existing GSS-mechanism (such as e.g. Kerberos) could provide the basic key-exchange requirements and other required GSS functions. SAML constructs of any type (authentication statements, attribute statements or authorization decisions) can be attached to an existing credential and protected by the underlying security mechanism.

Internal Decoration
: Some of the existing GSS mechanisms already provide extensions points within the structure of the token. For example, Kerberos has the AUTH_DATA field for authorization data which is used by DCE and Windows. X.509 allows for an even more generic extension of existing certificate structures.

Decoration is particularly useful in the context of OS authentication and authorization. Especially in scenarios where existing authorization mechanisms are poor (like the UID/GID authorization in Unix and Unix-like systems), the use of SAML attribute statements through the decoration approach can quickly lead to a significant improvement of the authorization system. Also, as already mentioned, by using unnamed keys and a restricted subset of the attributes available for a particular identity, a high degree of privacy can be guaranteed.

A key benefit of either decoration approach is that the underlying mechanism has already an established and secure way of dealing with keys and other sensitive information. Thus, key negotiation or exchange are problems that are already solved.

4.2. Native SAML Tokens

In order to combine the GSS-API token exchange natively with SAML based authentication statements, it is necessary to define a GSS-SAML profile that performs the necessary encoding of the SAML XML based token into the GSS compliant token exchange format. Credentials and context tokens may rely partially on XMLenc, XMLDsig (see discussion below) and SAML.

The GSS-API per-message tokens are the same as for the Kerberos GSS mechanism. Authentication databases can be SAML native (e.g. a Liberty IdP) or be based on an existing authentication mechanism like e.g. Kerberos, LDAP, or a flat file. XML based tokens are quite long and carry a lot of redundant information. As such it makes sense to use SAML Artifacts within the message exchange framework and have those Artifacts resolved by the accepting party. The message flow can look possibly like this:

1. The GSS-API Initiator (client) calls GSS_Init_sec_context() to connect to the Service Provider (i.e. the GSS-API Acceptor)

2. The Acceptor returns along with a GSS_S_CONTINUE_NEEDED a SAML Request, describing the SAML assertion it requires for a successful login.

   At this time, the client COULD inspect the SAML Request and compare it to a user-defined policy (e.g. about the nature of attributes that may or may not be shared with the Service Provider).

3. The client forwards the SAML Request to the IdP

4. The IdP creates a SAML Artifact reference and passes this reference back to the client.

5. The client calls GSS_Init_sec_context() with the handle to the first context as a reference and includes the Artifact.

6. The Service Provider (i.e. the GSS-API Acceptor) uses the SAML Artifact resolution protocol (which should be secured by TLS, using client and server certificates) to obtain the SAML Assertion from the IdP

7. Upon successful authentication and authorization - based on the information found in the assertion - the GSS-API Acceptor calls GSS_Accept_sec_context() and establishes the secured context.

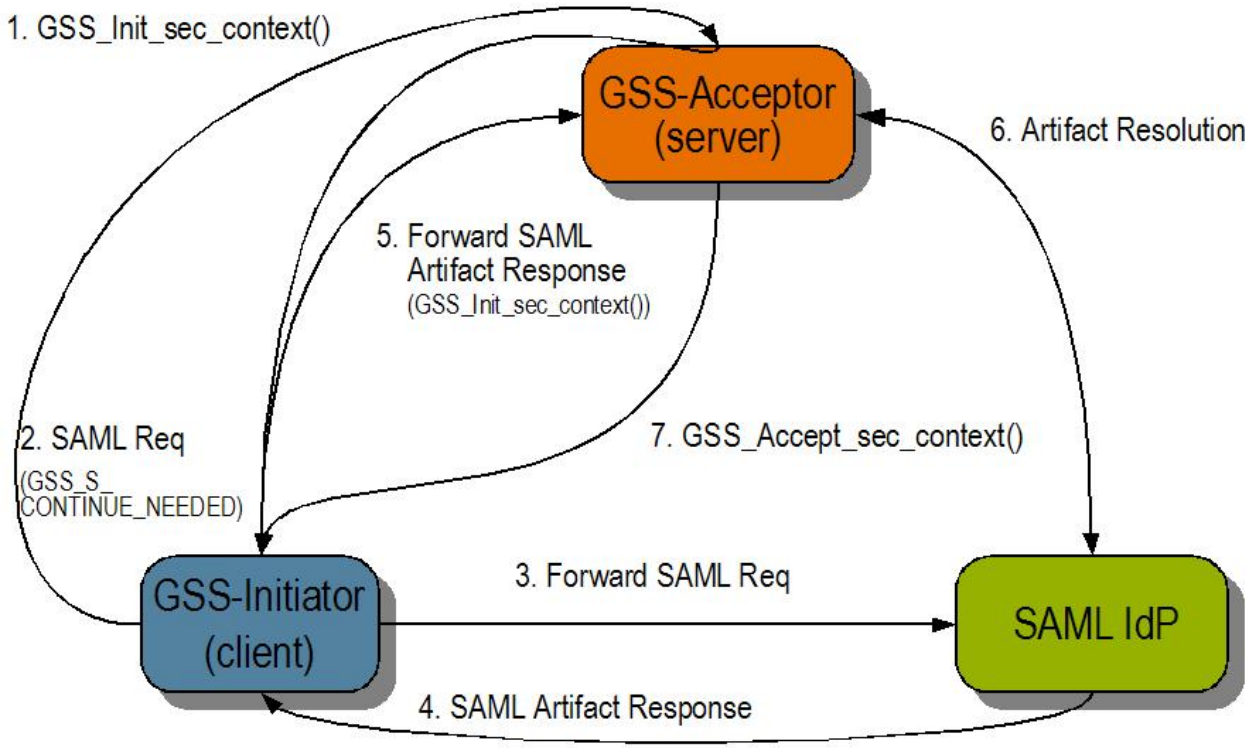

Figure 2. Native GSS-SAML Message Flow

Unfortunately, the XML Encryption and Signature protocols are not quite ideal for the required transactions: There is no key exchange or negotiation protocol, making it necessary to create a ephemeral-ephemeral Diffie-Hellman style mechanism (for asymmetric keys) and a Needham-Schroeder style mechanism (for symmetric keys).

Another issue around the XML security protocol is that they do not provide for Authenticated Encryption, i.e. an ordered way to encrypt and sign a particular message. However, there are a variety of technologies today, that already use Authenticated Encryption (such as e.g. Kerberos in IETF RFC 3962).

5. Discussion

While all approaches have their respective advantages and benefits there are some fundamental differences.

1. Profiling and implementing the decoration approaches would likely be easier than going through the process profiling a completely new protocol and implementing it. Another important benefit of the decoration approaches is that it takes most of the cryptographic considerations out of the mechanism design, re-uses code and, even more importantly, re-uses credentials. This is particularly important, since migrating from one system to another is always painful.

2. The fresh-start options would define a fundamentally new GSS mechanism that would need to be implemented at least on Solaris and Windows. This is a new approach that would promise the unification of WebSSO and Platform SSO technologies.

Acknowledgements

I would like to thank Nicolas Williams, Eve Maler and Hubert Le Van Gong for their insightful comments.

Bibliography

[GSS-API] *Generic Security Specification Application Programming Interface, Version 2 Update 1:* *ftp://ftp.rfc-editor.org/in-notes/rfc2743.txt*

[SAML] *SAML Specification at OASIS:* *http://www.oasis-open.org/committees/documents.php?wg_abbrev=security*

[SASL] *Simple Authentication and Security Layer:* *ftp://ftp.rfc-editor.org/in-notes/rfc2222.txt*

[X.680] *ASN.1* ITU-T Recommendation X.680

[X.891] *FastInfoset* ITU-T Recommendation X.891 ITU-T Study Group 17 05/2005

Biography

Gerald **Beuchelt**
Web Services Architect
Sun Microsystems, Inc. [http://www.sun.com]
Chief Technologist's Office
1 Network Dr.
Burlington
Massachusetts
01803
United States of America

Gerald Beuchelt is a Web Services Architect in the Chief Technologist's Offices Business Alliance group. He is focusing on advanced web services and security technology and their application, with an in-depth focus on standards and Microsoft interoperability.

Before, Gerald was an Infrastructure Services Specialist in CTO's Competitive Strategy Group. He joined the CSG in 2000. Since then he has worked on operating systems, middleware software including Java and .NET, and security, specifically in Microsoft product related areas. In this function, he worked closely with Sun Legal.

Prior to that, he worked for Sun Microsystems Deutschland (Germany) as a Senior Systems Engineer, most recently taking the lead in orchestrating technical support for large sales projects.

Before joining Sun, Gerald worked as a lead network architect and system administrator in ITP and IGM, Cologne, Germany, and as a freelance writer for two major German computer periodicals.

Gerald recently submitted two invention disclosures to the Sun Patent Office and contributed to numerous field and engineering education summits. He also authored and co-authored various competitive white papers on Active Directory.

Education:

* BS Physics 1994, Ludwig-Maximillian-Universität, Munich

* MS Mathematical Physics 1997, Albertus Magnus Universität zu Köln, Cologne