Abstract:
Web Servers and Applications Servers generally maintain security state information for currently active users, particularly once some type of authentication has occurred. This specification defines a format for communicating such security session state based on the OASIS SAML Assertion. It also specifies two different mechanisms for communicating this information between servers via a standard Web browser.

Status:
This document is a Working Draft and as such as no official standing with regard to the OASIS Technical Committee Process.
# Table of Contents

1 Introduction (non-normative).................................................................................................................... 3  
1.1 Terminology....................................................................................................................................... 3  
1.2 Normative References...................................................................................................................... 3  
1.3 Non-normative References................................................................................................................ 3  
2 Session Management Architectures (non-normative)........................................................................ 4  
3 Session Management Algorithm (non-normative)........................................................................... 6  
3.1 Stateful Token Algorithm.................................................................................................................. 6  
3.2 Session Reference Algorithm........................................................................................................... 7  
4 Token Format (normative).................................................................................................................. 9  
4.1 Assertion Header............................................................................................................................... 9  
4.2 Authentication Statements............................................................................................................... 10  
4.3 Attribute Statement........................................................................................................................ 11  
5 Token Carried in Cookie (normative)................................................................................................. 12  
6 Session Reference Carried in Cookie (normative)........................................................................... 13  
7 Cookie Naming (normative).............................................................................................................. 14  
8 Security Considerations (non-normative)........................................................................................... 15  
9 Conformance....................................................................................................................................... 16
1 Introduction (non-normative)

Although the http protocol is deliberately stateless, efficient implementation of security requirements such as attribute-based authorization and inactivity timeout require maintaining state associated with each active connection. This state may consist of historical information (authentication occurred), relatively static information (user’s attributes) and dynamic information (time of last interaction).

Web applications are commonly implemented by passing requests from browsers to any of a number of servers. These servers may be heterogeneous or homogeneous in function, geographically centralized of distributed. Typically users are unaware that multiple servers are involved. It is therefore desirable to simulate a single system with uniform knowledge and behavior.

This means that a server receiving a request from a browser that last interacted with a different server must have a means to obtain the most recent session state. The only practical method of doing this is to pass the information via the browser using an http cookie. The cookie may be used either to pass the encoded session token itself, or if it is too large, to pass a reference to the token.

1.1 Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in IETF RFC 2119 Error: Reference source not found.

1.2 Normative References

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
</table>

1.3 Non-normative References

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Reference]</td>
<td>[reference citation]</td>
</tr>
<tr>
<td>[Reference]</td>
<td>[reference citation]</td>
</tr>
</tbody>
</table>

[Ed: Should we cite the technical overview?]
2 Session Management Architectures (non-normative)

In this document the server providing session information is called the Session Authority (SA) and the server using the information is called the Session Consumer (SC). These roles operate only in the context of a single interaction. Usually servers will take on each role in turn. The token is created by the SA and read by the SC.

Session management can be implemented using a variety of architectures. For example, each Web or Application server can implement a session management capability internally as shown in Figure 1. In this case each server acts as both SA and SC.

![Figure 1 – Every Server a Session Manager](image)

Session management can also be implemented by one or more dedicated session management servers as shown in Figure 2. These are accessed as needed by web and application servers. Depending on the specific design the session manager may act as SA and SC or the roles may be divided between the session manager and web servers.
Figure 2 – Dedicated Session Management Servers
3 Session Management Algorithm (non-normative)

This section describes the type of algorithm which MAY be used to by a server which is acting as both an SA and SC. There are two variants, depending on whether the cookie contains the Token or is a reference to the Token.

3.1 Stateful Token Algorithm

When the session state is encoded into the cookie, interactions are entirely between web browsers and session managers. There is no direct communications between the SA and SC as shown in Figure 3.

1. When an application request is received, the server first checks to see if a session cookie is present. If it is not, the server proceeds as it would with any request from a user who has not authenticated. Depending on the request this may mean permitting it, prompting for authentication or taking some other action.
2. Next the server determines if the cookie is stateful or a session reference. Processing of a session reference is described in the next section.
3. The server verifies the signature of the Token. The ability to determine the correct key to use for this purpose implies some type of key management function. If the signature is not valid, the request is discarded with no action, so as to reduce the effect of denial of service attacks by unauthorized users. (Administrative reporting of potential attacks may occur.)
4. The Token validity interval in the <Conditions> element is checked to make sure the token is currently valid. The current time must be equal to or later than the value of the NotBefore attribute and earlier than the value of the NotOnOrAfter attribute. Other checks may be performed to ensure the Token contains the required information.
5. If IP address checking is enabled, the address in the Token is compared to the IP address from which the request originated. If they are different, the request is discarded.
6. Idle time out may be implemented by configuring each server with a maximum idle time value. Typically, the value will be the same for all servers hosting the same application type, but this algorithm does not depend on this being the case. It is simply assumed that each server is configured with a maximum idle time value by some means unspecified in this document. In practice, maximum idle time values might range from 5 minutes to 30 minutes. If idle timeout is enabled, the server subtracts the user's time of last access from the current time and compares the result to the maximum idle time value. If the time since the last request exceeds the maximum value, the Token is discarded, any existing session information for that user is cleared and the user is informed that the session has timed out because of inactivity. Typically the user is prompted to re-authenticate.

7. Maximum login time (sometimes called session time limit) may be implemented by configuring each server with a maximum login time value. This may be a single value or depend on the type of login performed most recently. Maximum login time limits typically range from 1 hour to 24 hours. If maximum login time is enabled, the server subtracts the time of the most recent authentication from the current time and compares the result to the maximum login time. If the time since the last authentication exceeds the maximum value, the user is prompted to login, before processing the request.

8. Now the server is able to use the contents of the Token. The information may be stored in some session management data structure or simply used directly. At a minimum the information must be retained until the response to the request has been sent. For example, the user name may be used for authorization, personalization or as a key to look up other user attributes.

9. When the response is sent, the server constructs a Token containing the received values for user attributes. The value of `TimeLastActive` is set to the current time. Any new authentications performed during the request are described in Authentication Statements. The Token is then signed and inserted in the cookie of the response.

10. As an optimization, the server may maintain a Token Freshness value, which allows Tokens to be reused if they were created recently. The value might be something like 30 seconds. If the `IssueInstant` Attribute subtracted from the current time is less the Token Freshness value, the received Token is put in the cookie instead of creating and signing a new Token. This reduces the overhead of a series of closely spaced requests at the cost of reducing the precision of the idle timeout and maximum login time algorithms.

### 3.2 Session Reference Algorithm

Instead of the cookie containing the Token, it may instead merely contain a reference to the session. The actual session Token is obtained by making a query to the SA which generated the reference. In this case the cookie contains two parts: a server endpoint in the form of a URI and a large random number. In this case, the SA and SC communicate directly as shown in Figure 4.
The SC calls the indicated endpoint, providing the reference as an input value. The SA checks to see if
the reference corresponds to a valid session. If not, it returns an error. If so, it creates a session Token
and returns it. If this back channel connection is integrity protected, e.g. using TLS, then it is not
necessary to sign the Token. The SC then processing described in section 3.1 beginning with step 4.
4 Token Format (normative)

The format of the Session Token is based on the SAML Assertion Element defined by [SAMLv2Core] The Assertion MUST contain exactly one Authentication Statement and at exactly one Attribute Statement. The contents of the Assertion and the Statements are specified in the following sections.

4.1 Assertion Header

The assertion header MUST contain the following items.

Version
This MUST contain “2.0” AS required by [SAMLv2Core].

ID
This MUST contain a unique identifier as required by [SAMLv2Core].

IssueInstant
This MUST contain the time the Token was created as required by [SAMLv2Core]. When the cookie contains a session reference, it will usually differ from the user's TimeLastActive.

<Issuer>
This MUST contain the name of the Session Authority.

<Signature>
When the Assertion is carried in a cookie, it MUST be signed. See Section 5. If the Assertion is signed, the SC MUST verify the signature before processing it.

<Subject>
The Subject MUST contain the following Elements and Attributes.

<NameID>
The NameID MUST contain the name of the user.

Any implementation of this specification MUST profile the use of the NameID element and its associated Attributes: NameQualifier, SPNameQualifier, Format and SPProviderID or else it MUST state that they are not used.

<SubjectConfirmation>
The SubjectConfirmation MUST contain a Subject Conformation Method attribute.

Method
The Subject Confirmation Method MUST have a value of
<SubjectConfirmationData>
SubjectConfirmationData must have the following attribute.

Address
The address MUST contain the address of the browser in IPv4 dotted decimal format, e.g. "198.51.100.1" or in IPv6 address format as described in Section 2.2 of [RFC 3513], e.g., "2001:db8::1"). The SC MAY compare the value to the known address of the browser.

<Conditions>
The Conditions Element MUST contain the following attributes.
NotBefore
NotOnOrAfter
These define the validity interval of the Token. The SC MUST ignore the Token if the current date time does not fall in this range.

<Advice>
The Assertion MUST NOT contain an Advice Element.

4.2 Authentication Statements
The Assertion MUST contain exactly one Authentication Statement. It MUST contain the following attribute.

AuthnInstant
This time value MUST be the time authentication occurred, as defined in [SAMLv2Core].

<AuthnContext>
The contents of the Authentication Context MUST conform to [SAMLv2AuthnCtx].
The AuthenticationStrength Attribute in the Attribute Statement, (see section 4.3), MUST correspond to the value assigned to the authentication method present in the Authentication Statement.
The level of assurance (LOA) associated with this Authentication MAY be expressed as specified in [SAMLv2IdAssure].

4.3 Attribute Statement
The Assertion MUST contain exactly one Attribute Statement.

<Attribute>
The following SAML Attributes MUST be present.

**SessionId**
This MUST be of type string and MUST contain the unique identifier of the session. (This is not the same as the session reference described in section 6.)

**SessionStart**
This MUST be a time value as defined by [SAMLv2Core]. The value MUST be the date time of the start of the session. Every time the user authenticates, a new session is deemed to start. Therefore the SessionStart will always be the same as the AuthnInstant in the authentication statement.

**AuthenticationStrength**
This MUST contain a value of type integer in the range of 0-99. It is a deployment-specific value associated with every type of Authentication supported by the deployment, where a higher number represents a more secure method. The value of the Attribute MUST correspond to the value assigned to the authentication method represented in the Authentication Statement present in the Assertion. Authentication method is defined as a specific Authentication Context Class with specific instance values or ranges of values.

The means by which the mapping of Authentication methods to AuthenticationStrength is communicated to SAs and SCs is outside the scope of this Profile.

**TimeLastActive**
This MUST contain the date time of last request from the browser.

**TokenFormatVersion**
This string value MUST contain the major and minor version numbers of the Token format being used, e.g. “2.3”

The Attribute Statement MAY contain other Attributes as specified in [SAMLv2Core].
5 Token Carried in Cookie (normative)

If size allows, the session token MAY be carried in the cookie. The cookie name should be chosen so that this case and the use of a reference (see section 6) can be easily distinguished.

When the token is carried in the cookie, it MUST be signed as specified in [SAMLv2Core]. The Token MAY also be encrypted as specified in [SAMLv2Core].
6  Session Reference Carried in Cookie (normative)

Instead of transmitting the Assertion in the cookie, the SA MAY instead put a reference to the Assertion in the cookie. The reference then MAY be used to retrieve the Assertion.

When this approach is used, the cookie value MUST consist of an http scheme URL followed by the “&” character, followed by an unguessable number of at least 256 bits.

The URL represents a server endpoint which supports the SAML Artifact Resolution Protocol as specified in [SAMLv2Core]. In this case, the unguessable number from the cookie is used at the Artifact.

[Ed: Does this need to be spelled out in more detail?]

[Ed: Is it ok to reuse the Artifact Resolution Protocol while deviating from the SSO Artifact Profile?]

The SA using this scheme MUST respond to protocol requests by returning the indicated Assertion with the session information.

The Token MUST be carried over secure transport and/or signed as specified in [SAMLv2Core]. The Token MAY also be encrypted as specified in [SAMLv2Core].
7  Cookie Naming (normative)

The cookie used to contain a session state reference or stateful cookie will have a three-part name.
- The first part of the name will indicate that this is a Session Token using this scheme.
- The second part will allow an SC to distinguish between a stateful and a reference token.
- The third part will indicate the version of the cookie format.

This will allow an SC to process the cookie without having to inspect the token to determine its type.

[Ed: Should this be specified in more detail or omitted and left as a deployment option?]
8 Security Considerations (non-normative)

The short summary is that this proposal has essentially the same security properties as existing deployed products.

The primary threats are: 1) Token forgery, 2) Token capture and unauthorized use and 3) unauthorized disclosure of Token contents.

When the Assertion is carried in the cookie, the signature will prevent forgery.

Capture of the Token as it traverses the network use can easily be prevented by protecting the browser session with TLS. This has been rare in past because of performance concerns. However, recently Google has publicized work showing that Running TLS has a minimal effect on capacity and throughput. They are also working on reducing latency, particularly in the initial handshake.

Depending on the application, it may be possible to capture a cookie via a cross-site scripting exploit. Cookies can also be subject to interception if presented to some web sites without using TLS. Setting the “Secure” property on the cookie as specified in [RFC 2965]. Cookies may also be captured if any server in the domain is controlled by an attacker, whether or not TLS is used.

IP address checking will generally be effective in preventing this type of impersonation, but the widespread use of Network Address Translation (NAT) makes this questionable. It would seem that an attacker who could intercept messages from a point along the network path from browser to server and could also transmit from that point, could spoof the IP address. Encrypting the Assertion would hide the IP Address there, but it would still appear in the IP header.

Another threat is that one sever could take the token from a user and use it to impersonate that user to another server. This scheme assumes that servers can be trusted not to do this, just as they are trusted not to misuse the passwords users type in.

If unauthorized disclosure is a concern, the Assertion can be encrypted as specified in [SAMLv2Core]. However, if an unauthorized party can obtain a copy of the token, whether encrypted or not, it can be presented to impersonate the user. Therefore the utility of encrypting the Assertion is unclear. Generally, exposure of a user’s session state information to that user will not be considered a threat.

When the cookie carries only a reference, no integrity check is required. If the value is invalid, the SAML request will fail. (Technically SAML will return an empty response.) Again, interception of the cookie will permit impersonation, but this seems to be a threat to any cookie-based scheme.
9 Conformance

A Session Authority conforms to this specification if it

• generates Assertions conforming to Section 4,
• uses the cookie naming scheme specified in Section 7, and
• transmits the Assertion using the method defined in Section 5 or Section 6.

A Session Consumer conforms to this specification if it

• can process an Assertion as specified in Section 4,
• can process a cookie named as specified in Section 7, and
• access an Assertion using the method defined in Section 5 or Section 6.

[Ed: These are specified in terms of deployment. Should they be specified in terms of implementation?]
Appendix A. Acknowledgments

The following individuals have participated in the creation of this specification and are gratefully acknowledged

Participants:

[Participant name, affiliation | Individual member]
[Participant name, affiliation | Individual member]
[Participant name, affiliation | Individual member]
Appendix C. Revision History