A Single Sign-On Framework for Web-Services-based Distributed Applications

Markus Hillenbrand*, Joachim Götze*, Jochen Müller*, Paul Müller*
*University of Kaiserslautern, PO Box 3049, D-67653 Kaiserslautern, Germany
{hillenbr, j.goetze, jmueller, pmueller}@informatik.uni-kl.de

Abstract—The basic specifications of Web Services completely ignored the need for secure services that are based on a solid authentication and authorization infrastructure. An enhancement of the basic specifications is in progress, but takes time to complete. A large-scale application built today is still in need of an authentication and authorization infrastructure to offer its services to the customers of the service provider and only to the customers. In the following such an infrastructure is described that is able to empower Web Services to properly handle authentication and authorization. This infrastructure is designed as services that can be used by any demanding Web Service.

I. MOTIVATION

Web Services are capable of providing all kinds of services to their clients. Although allowing every client to use a Web Service without even knowing anything about the implementation or the location of the Web Service was part of the primary targets of the basic specification, it was soon recognized that this objective leads to several problems. The obvious field of operation is the use of Web Services inside large companies, but often the services are meant to be used also by clients of that company. If a company does not provide internal Web Services only, security cannot be ensured solely by using a standard corporate firewall controlling the network traffic, because the firewall would not be capable of distinguishing between allowed and prohibited data packets as they are both sent to the same destination: the company’s Web Service server. Although there are solutions under development for this (e.g. a Web Service firewall which is able to validate XML data against the WSDL specification of a Web Service developed by the Communication Systems Research Group of the Christian-Albrechts-University in Kiel [1]), these solutions can only distinguish between proper and improper access, but not between allowed or disallowed clients, especially if the clients are mobile and access the company’s network through frequently changing Internet access points.

These problems are typical for the daily business of companies and illustrate the need for an authentication and authorization infrastructure based on Web Services. Several standardization groups started to define a solution that solves these problems by enhancing the current Web Service standards, e.g. [2], [3] and [4]. The target of these standards is not only the security of a single Web Service, but the dynamic interaction of Web Services belonging to different corporations or domains in a secure manner.

In the meantime, while the standardization process is still in progress, an advanced authentication and author-
showing that now functionality for Web Services has been added.

Part of the Liberty Alliance Project is the specification of the Liberty Identity Web Services Framework [12]. This framework defines standardized messages to handle authentication, message protection, service discovery and policies.

III. BASIC CONCEPTS

A. Single Sign-On

A single sign-on (SSO) environment is an authentication infrastructure that provides a single authentication authority for all authentication processes by outsourcing authentication processes to a specialized infrastructure that handles authentication for all applications. Jan de Clercq defines SSO in [13] as follows:

“Single sign-on is the ability for a user to authenticate once to a single authentication authority and then access other protected resources without re-authentication.”

In a service-oriented architecture the management of authentication information has to be provided as a service. This is done by a standardized scheme to handle authentication that is located on a centralized authentication infrastructure. From the users point of view one authentication gives transparent access to all services that use this single sign-on service.

Fig. 1 shows the authentication process in a SSO environment using tokens. After the primary sign-on the user receives a temporary token that he can use to proof his identity while signing on to any authentication authority trusting the primary authentication authority.

B. Federation of Trust

Because of the diversity of SSO solutions, it is difficult to combine several infrastructures to enhance the range of a SSO domain. Especially if two companies are working together closely, the ability of SSO is highly desirable, but none of the companies would be willing to change their complete authentication infrastructure to match with the infrastructure of the other company. Additionally it is not the same kind of trust relationship as it is between two authentication authorities inside a single company. Both companies would only be willing to give access to information needed for their joint venture, but not to all business secrets, and they would want complete control of the access rights they are granting.

Another problem arises from short-term co-operations. The trust relationship has to be installed quickly and it may not exist for a long time. A Federation of Trust between two authentication authorities can solve these problems by defining mechanisms to enable different security realms to federate using different or alike mechanisms by allowing and brokering trust of identities, attributes, authentication between participating authentication authorities.

Fig. 1: Service-based SSO

IV. MODEL FOR AN AUTHENTICATION AND AUTHORIZATION INFRASTRUCTURE

The process of authentication in this model is closely related to the authentication protocol Kerberos [14], but it is extended to fulfill the needs of a federated SSO environment on the basis of Web Services. The Kerberos Protocol has the advantage of providing an easy to use basis for a SSO environment utilizing the security token to prove a successful sign-on. Although having advantages, the protocol has certain drawbacks to overcome. Kerberos V4 and V5 both provide the possibility for federated authentication (interrealm authentication), but both versions are not suitable for a highly dynamic network.

Kerberos V4 uses only direct trust between certain authentication domains (in Kerberos terminology: realms). This leads to a maximum of $n^2$ interrealm security keys for $n$ domains. Not only the number of security keys is huge, but the time to set up a domain needing many communication paths to different domains (Fig. 2).

Kerberos V5 was designed to improve the interrealm communication and introduced a tree-like hierarchy of domains. A domain only needs to store the interrealm security keys from the parent node and the child nodes. If a user needs to access a federated domain, the user has to request a security token from the authentication service in it’s home domain that is marked valid for
remote authentication. This token could then be used to authenticate locally. Although this concept offers several advantages to the interrealm authentication of the V4 protocol, the V5 interrealm authentication is still focused on a static hierarchy of authentication domains.

Additionally both types of Kerberos protocols do not support the ability to provide authentication information for access control. Therefore this extension does provide authorization information stored by a centralized authority. Furthermore it is possible to have a mapping of authorization information regarding to different environments and operating systems utilizing user roles. This abstraction of authorization information based on a mapping allows every party belonging to a Federation of Trust to adjust the authorization of federated users as they seem fit.

A. Components of the authentication and authorization infrastructure

An authentication and authorization infrastructure consists of four basic components. The first component is the user or client. This component has to be integrated in the system of the client and handles any operation needed for authentication and authorization. The authentication service - providing everything needed to allow a user to claim an identity and proof that this claim is justified - is the second component. If the user has successfully claimed an identity, he receives a token to prove his identity. This token has to be used with the third component: the authorization service. This service provides the user with tokens which allow him access to services, if he has the privileges to use these services. Finally there is the service component. The task of this component is to ensure that the user of the service can provide a proper service token, i.e. he is successfully authenticated and received authorization to access the service, and that the access rights are sufficient to utilize the requested service functionality.

B. Authentication in the local authentication domain

The authentication service has to handle two different types of service requests. At first there is a user belonging to the local authentication domain signing on and secondly there is a user from a federated domain signing on. The focus of this section is on the local sign-on process, while the subsequent section explains the sign-on process in a federated domain.

During the authentication process the claimed identity of the requesting user is verified. During the following analysis of local and federated authentication processes, the transfer of encrypted data between communicating parties is needed many times and in order to provide a general notation, the notation defined in [14] is used.

Referring to Table I, in step 1 the client C transmits his username U, his domain D_U, the timestamp \( T \) and the lifetime \( L \) to the authentication service \( AS \) in order to claim his identity. The authentication service \( AS \) responds in step 2 by sending back the key \( K \) and the security token \( T_{C,STS} \) which is encrypted with the password \( K_D \) of the user \( U \). The term \( T_{C,STS} = \{ U, D_U, STS, T, L, K \} \) \( K_{STS} \) is used to refer to a security token that is issued by the authentication service \( AS \) to be used by the client \( C \) to authenticate to a service token server \( STS \) to request a service token for a service the client is going to use. The key \( K_{STS} \) is a shared secret between the authentication service \( AS \) and the service token server \( STS \). Although these two services are regarded as separate services, it is possible to combine them to a single service. The key \( K \) is used for encryption when the client \( C \) communicates with the service token server \( STS \).

During this authentication process the client never presents his credentials to the authentication service, because of that the authentication service is not able to compare the credentials with the stored credentials of the user. Nonetheless the authentication service replies to the user with a security token that enables the user to authenticate to the service token server, but this reply is encrypted with a key \( K_D \), the password of the identity the user claimed to be. Whether the correct user has claimed that identity or not, only the correct user is able to provide the correct password, i.e. the key for decryption, to take possession of the security token contained in the reply message. Because of that a user’s password is never transmitted across the network.

The process of using a service is separated into two consecutive operations. First the client has to request a service token from the service token service (step 3 + 4) and then present that token to the corresponding service to authenticate himself (step 5 + 6). After that the client is allowed to utilize the service.

In step 3 the client \( C \) sends his request for a service token to the service token service \( STS \). For this request the client \( C \) presents the already acquired security token \( T_{C,STS} \), the name \( S \) and domain \( D_S \) of the service the client wants to use and an authenticator \( A_{C,STS} = \{ U, D_U, T, L, N \} K \). An authenticator is a data record containing information that allows the receiver to check if the transferred message has been recently generated. The authenticator is encrypted using the session key \( K \) known only to the client and the server. The nonce

![Fig. 2: Full Interrealm connectivity with Kerberos V4](image-url)
$N$ is a serial number used to prevent the replay of the message. The service token service $STS$ has to store the last serial numbers used to validate the freshness of an authenticator. This communication is implicitly encrypted, because $T_{C,STS}$ and $A_{C,STS}$ are already encrypted.

### TABLE I

**COMMUNICATION IN THE AUTHENTICATION PROCESS**

<table>
<thead>
<tr>
<th>Step</th>
<th>Sender</th>
<th>Recipient</th>
<th>Transmitted data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$C$</td>
<td>$AS$</td>
<td>$: U, D_U, T, L$</td>
</tr>
<tr>
<td>2</td>
<td>$AS$</td>
<td>$C$</td>
<td>$:{K, T_{C,STS}} K_U$</td>
</tr>
<tr>
<td>3</td>
<td>$C$</td>
<td>$STS$</td>
<td>$: S, D_S, T_{C,STS}, A_{C,STS}$</td>
</tr>
<tr>
<td>4</td>
<td>$STS$</td>
<td>$C$</td>
<td>$:{T_C, S, K'} K$</td>
</tr>
<tr>
<td>5</td>
<td>$C$</td>
<td>$S$</td>
<td>$: T_C, S, A_C, S$</td>
</tr>
<tr>
<td>6</td>
<td>$S$</td>
<td>$C$</td>
<td>$: A_{SC}$</td>
</tr>
</tbody>
</table>

In step 4 the service token service $STS$ responds to the requesting client $C$ with a service token $T_{C,S} = \{U, D_U, S, D_S, A, T, L, K'\} K_S$, that can only be used by the client $C$ for the service $S$ (encrypted with the secret key $K_S$, that is shared between the service token service $STS$ and the service $S$), and a session key $K'$ for the communication between the client $C$ and the service $S$. The reply is encrypted with the secret key $K$, which the client received during authentication (step 2). The service token service $STS$ gets access to the secret key $K$, because the secret key $K$ is encrypted in the security token $T_{C,STS}$ the client presents to authenticate.

After step 4 the first part of the process to use a service is complete. The client has acquired everything needed to access the service. In step 5 the client $C$ sends the service token $T_{C,S}$ and the authenticator $A_{C,S} = \{U, D_U, T, L, N\} K'$ to the service $S$.

In step 6 the service $S$ responds to the client $C$ with the authenticator $A_{S,C} = \{S, D_S, T, L, N - 1\} K'$. The nonce $N$ received by the service $S$ is decreased by 1 and returned in step 6. This allows the service to ensure the freshness of the message by comparing the nonce with the already received nonces and it allows the client to check the correctness of the reply by the service, who had to successfully decrypt the authenticator $A_{C,S}$ received from the client to present the correctly decreased nonce in the authenticator $A_{S,C}$. This decreased nonce proves that the service is the expected service and not a malicious service impersonating the expected service. After that the client and the service are mutually authenticated and the client can start to use the service.

### C. Authentication in a federated authentication domain

In this section the authentication process of a user belonging to a federated authentication domain is explained in detail. This process only differs from the explained authentication process in the previous section in its first two steps. After that the user has proven his identity and can provide the security token received ($T_{C,STS}$).

The first two steps change transparently for the user. From the client’s point of view in step 1 the client $C$ transmits his username $U$, his domain $D_U$, the timestamp $T$ and the lifetime $L$ to the authentication service $AS$ in order to claim his identity. The authentication service $AS$ responds in step 2 by sending back the key $K$ and the security token $T_{C,STS}$ which is encrypted with the password $K_U$ of the user $U$.

The difference in the authentication process has its origin in the use of the user’s password to encrypt the response of the authentication server. This password cannot be found in the local credential database, because the user does not belong to the local authentication domain. Therefore the data has to be requested from the federated authentication service as is explained in the following paragraph while the communication is shown in Table II.

### TABLE II

**AUTHENTICATION UTILIZING DIRECT TRUST**

<table>
<thead>
<tr>
<th>Step</th>
<th>Sender</th>
<th>Recipient</th>
<th>Transmitted data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$C$</td>
<td>$AS$</td>
<td>$: U, D_U, T, L$</td>
</tr>
<tr>
<td>1'</td>
<td>$AS$</td>
<td>$ASR$</td>
<td>$:{U, D_U, T, L} K_{fed}$</td>
</tr>
<tr>
<td>2</td>
<td>$ASR$</td>
<td>$AS$</td>
<td>$:{K, T_{C,STS}, {K, T_{C,STS}} K_U} K_{fed}$</td>
</tr>
<tr>
<td>2'</td>
<td>$AS$</td>
<td>$C$</td>
<td>$:{K, T_{C,STS}} K_U$</td>
</tr>
</tbody>
</table>

The local authentication service $AS$ transmits the sign-on request encrypted with the secret key $K_{fed}$ to the remote authentication service $ASR$ handling the authentication for the domain the client $C$ belongs to (step 1'). The secret key $K_{fed}$ is specified during the establishment of the Federation of Trust. In step 2’ the authentication service $ASR$ supplies the authentication service $AS$ with the requested security token $T_{C,STS}$. Because the local authentication service $AS$ needs to access the security token, $T_{C,STS} = \{U, D_U, STS, T, L, K\} K_{fed}$ is encrypted with $K_{fed}$ instead of $KSTS$. This security token $T_{C,STS}$ is returned to the client, who has to provide the correct key $K_U$ to decrypt the received data and extract the secret key $K$ that has to be used for the communication with the local service token service $STS$. After that the local service token service $STS$ can handle any further requests of the client $C$ by himself.

If the authentication request is from a user that does not belong to a domain with which the local domain has a direct trust relationship, the protocol must be extended in order to bridge the intermediary domains (refer to Table III).

The extension of the protocol can be seen in the steps 1', ..., 1'($n$) and 2', ..., 2'($n$). In order to indicate that there can be more than one intermediary authentication service $ASI_i$ with $i = 1, .., n$, these steps are separated by dots. In order to bridge these intermediary communication nodes, the data transmitted is encrypted with the corresponding secret key $K_{fed}$, that these intermediary communication nodes share with their predecessors respectively successors. All intermediary communication nodes have a direct trust relationship with their predecessors respectively successors which is the basis of an indirect trust relationship, i.e. trust is federated across the network of authentication domains.

The model does not define how a local authentication service can find the federated authentication service that
can process the sign-on request, but if an indirect trust relationship exists and because a successful communication does only have a finite number of intermediaries, it is possible to find a path in the network graph. Algorithms to find a routing table for efficient communication can be found in [15].

After the security token has been sent to the requesting user, the authentication process is finished. The further requests of this user to receive any service tokens needed can then be sent directly to the local service token service STS, that handles these requests without further interaction with the remote authentication service ASR, but may need to contact the remote service token service to request authorization information of the user.

D. Authorization

During the authentication process the service S receives the token $T_{C,S} = \{U, D_U, S, D_S, A, T, L, K'\} K_S$ including the authorization information $A$. This information $A$ includes everything the service needs to decide whether or not a client has access to the requested functionality of the service. Additionally the authorization information cannot be changed by the user to obtain extra privileges, because the whole token $T_{C,S}$ is encrypted with a shared secret $K_S$ of the service token service and the service.

As there are many different kinds of services, e.g. print service or ftp service, with different types of restrictions for their functionalities, e.g. a user is only allowed to print in black and white or a user may only download one file simultaneously per session from the ftp service, the appearance of the authorization information cannot have universal definition. Therefore the appearance of the authorization information is not defined at all. Only the service itself can generate and understand the authorization information of a user. This is sufficient, because the service token service has only the task to provide the authorization information in a centralized way for its security domain, but does not need to understand the provided information.

As the authorization information does not have one single shape, it is possible to introduce more sophisticated access rights. The access rights can now not only consist of a simple “yes” or “no”, it is now possible to define access rights in any way the service may support restrictions. Especially services that use a concept of access rights that is not very common can be supported.

Beyond the access rights resulting from a single user role, a user can impersonate multiple user roles, therefore the service token service has to provide a service token, that contains all authorization information for every related user role, because the service token service is not able to decide which user role can provide enough authorization to fulfill the desired request of the user. Additionally it is possible that user roles provide disjoint authorization for the service, so that the request of the user can only be fulfilled by using more than one user role. Because of that the appearance of the authorization information is: $A = \{A_{role_1}, ..., A_{role_N}\}$ where $N$ is the number of user roles the user can impersonate.

E. Delegation

An additional feature of authorization, that can be very useful, is delegation. Delegation means that a user can give some of his access rights (or all of them) to another user for some period of time. In this model delegation is realized by the ability of the service token service to store a delegation order of the user who wants to transfer some of his access rights to another user. If the other user is requesting a service token not only his own user roles are taken into consideration to generate the authorization information included in the service token, but also the transferred access rights of the delegating user. In order to delegate rights to another user, the originating user must have the right to delegate, because this option can have serious impact on domain security if it is not used reasonably.

F. Mapping of user roles

The authorization information of a local user is provided by the service token service by issuing an encrypted service token. If the user does not belong to the local domain, additional problems arise.

The service token service has to issue a service token without access to the authorization information of the user, because he belongs to a federated domain. Additionally there is the problem, that the service token service can not utilize this information from a federated domain, because the remote operating system might vary from the local operating system, i.e. the access control information is not comparable with the access control information in the local domain, and the user might impersonate a user role that does not exist in the local domain, i.e. no access rights can be identified with this user.

The solution proposed here uses a mapping of user roles. This mapping assigns a user role of a federated domain to a user role in the local domain. The mapping only works in one direction: from the federated domain to the local domain.

The advantages of this solution are as follows:
- A user impersonating a user role of a federated domain not being mapped, does only acquire access privileges on a guest level in the local domain.

<table>
<thead>
<tr>
<th>Step</th>
<th>Sender → Recipient</th>
<th>Transmitted data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C → AS</td>
<td>$U, D_U, T, L$</td>
</tr>
<tr>
<td>1’</td>
<td>AS → ASI_1</td>
<td>$(U, D_U, T, L) K_{fed}$</td>
</tr>
<tr>
<td>2(n)</td>
<td>ASI_n → ASR</td>
<td>$(U, D_U, T, L) K_{fed}$</td>
</tr>
<tr>
<td>2(n)</td>
<td>ASR → ASI_1</td>
<td>$(K, T_{C,STS}, (K, T_{C,STS}) K_{U}) K_{fed}$</td>
</tr>
<tr>
<td>2’</td>
<td>ASI_1 → AS</td>
<td>$(K, T_{C,STS}, (K, T_{C,STS}) K_{U}) K_{fed}$</td>
</tr>
<tr>
<td>2</td>
<td>AS → C</td>
<td>$(K, T_{C,STS}) K_{U}$</td>
</tr>
</tbody>
</table>
The access rights of federated users in the local domain are set and controlled by the local domain. Because of the unidirectional mapping, each domain of a federation can control the access rights autonomously.

**Fig. 3: Mapping of user roles**

Fig. 3 shows an example of an user role mapping. The domainA and domainB have a direct trust connection and a mapping of user roles has been defined from domainA to domainB. The user role roleA is mapped to the user role admin and user role roleB is mapped to the user role user. The userA can impersonate the user roles roleA and roleB, while userB can only impersonate the user role roleB. Because of this mapping the userA can impersonate the user role admin in domainB, while userB has only a user status in domainB. Obviously the users from domainA will get access to domainB without the need to transfer the native access control information from domainA to domainB. Instead the native access control information of domainB can be used.

### G. Rejection of trust

Although federated access to services is highly desirable, the model of an authentication and authorization infrastructure described so far may lead to some unwanted access. There are situations conceivable where only access of users from security domains having a direct trust relationship is desired. In order to prevent indirect trust it would be possible to define a new federation of trust including only the security domains having a direct trust relationship. This approach results in distributed responsibility among all participating security domains and cannot be controlled by every domain autonomously.

Therefore the rejection of trust has been included in the model of an authentication and authorization infrastructure proposed here. Rejection of trust is the possibility to define a list of security domains that must not have access to the local security domain. This prevents users of unwanted domains from gaining access to the local security domain by an indirect trust relationship via a security domain that maintains a direct trust relationship with the local domain.

### V. PERFORMANCE EVALUATION

The model proposed here has been implemented as a prototype using the Java programming language with the SOAP implementation Apache Axis 1.1. The server and service modules of the authentication and authorization infrastructure utilize Tomcat 5.0 as a servlet container. Authentication and authorization information are stored in a PostgreSQL database.

Two processes have been the focus of the performance evaluation. First the time of an initial sign-on process without any communication of the authentication service with a federated domain, i.e. requesting and receiving a security token from the client’s point of view. The other evaluation was oriented towards the time needed to receive a service token for an exemplary service by a user from the local authentication domain, i.e. this also excludes the need for communication with a federated domain.

**Fig. 4: Performance of security token requests**

Fig. 4 shows the time needed during 1071 successful sign-on processes. The arithmetic mean of a security token request is 23 ms. The standard deviation of the normally distributed value set is 4.79 ms. As a result the prototype can handle approximately two-thirds of all sign-on requests in less than 28 ms and only 2-3 per cent of all sign-on processes have been longer than 33 ms.

Fig. 5 shows the time needed during 3000 service token requests. It is important to note that the first 1000 service token requests have been made by a single client thread and the requests 1001-2000 have been made by two client threads, while the service token requests 2001-3000 have been made by 5 client threads. Each thread subsequently requests 1000, 500, and 200 service tokens respectively. The arithmetic mean of a service token request is 61 ms. The standard deviation of the normally distributed value set is 54.8 ms. These data show that the prototype can handle approximately two-thirds of all
Fig. 5: Performance of service token requests

An issue for further investigation has been identified in the accounting of services depending on the impersonated user role, which leads to an extension of the previously described model. The extension is based on the great importance originating in such an environment from the user role utilized to access a service.

VI. CONCLUSION

This paper described the model of an authentication and authorization infrastructure supporting single sign-on and federations of trust on the basis of Web Services. In order to reach this aim, the focus has been at first on the design of an authentication protocol closely related to the well understood Kerberos protocol. This protocol has been extended with methods to transport authorization information and to delegate access privileges.

In order to support federations of trust, the protocol allows users from remote domains access to the local services by federating authentication to trusted domains. The authorization information of the user’s domain is transferred to the local domain on the basis of an user role mapping. This mapping transforms remote user roles to local user roles and thereby circumvents the problems of incomparable access control information and not existing user roles by assigning local access control information to federated users.

Support of delegation allows a user to receive additional access rights of another user on a temporary basis. The additional access rights are originating from some or all of the user roles of the original user.

REFERENCES