Integral Federated Identity Management for Cloud Computing

Maicon Stihler, Altair Olivo Santin, Arlindo L. Marcon Jr.
Graduate Program in Computer Science
Pontifical Catholic University of Paraná
Curitiba, Brazil
{stihler,santin,almjr}@ppgia.pucpr.br

Joni da Silva Fraga
Department of Systems Automation
Federal University of Santa Catarina
Florianópolis, Brazil
fraga@das.ufsc.br

Abstract—**Cloud computing environments may offer different levels of abstraction to its users. Federated identity management, though, does not leverage these abstractions; each user must set up her identity management solution. This situation is further aggravated by the fact that no identity federation solution is able to integrate all abstraction layers (i.e. IaaS, PaaS, and SaaS). On this paper we describe a new architecture offering integral federated identity management, to support multi-domain clients in a multi-provider environment. We also present some implementation details. The proposed architecture offers significant advantages over current offerings: it eases identity management without losing flexibility, offers better user tracking through the whole cloud computing layers, and enables the implementation of multi-provider environments through account data replication.**

Keywords: cloud computing; federated identity management; single sign-on.

I. INTRODUCTION

Federated identity management deals with the establishment of trust relationships between various security domains, to share authentication data to reduce management complexity and security risks. It also helps to simplify authentication procedures for end users (e.g. by employing single sign-on, SSO) [1]. This subject has been studied and applied to many environments, such as web resources [2], web services [3], and grid computing [4], an evidence of the high relevance of federated identity management.

The emergence of cloud computing created a new environment that is not completely addressed by previous works. Cloud computing can be categorized as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), or Software as a Service (SaaS) [5], respectively by increasing level of abstraction. It is a common approach for higher levels of abstraction to leverage functionalities provided by the lower levels. However, current federated identity solutions are isolated to a single level (e.g. identity is federated only for IaaS or SaaS). Thus, if a SaaS provider wants to employ user identification at a lower level (e.g. to track user actions for auditing) she will have to come up with her own *ad hoc* solution, as the lower levels (i.e. PaaS and IaaS) are completely unaware of such user. The matters are further complicated if the environment spans multiple IaaS providers, as no available solution can offer a federation that is both horizontally (i.e. between multiple IaaS providers) and vertically (i.e. through all abstraction levels – SaaS, PaaS, and IaaS) integrated.

We designed a new architecture for federated identity management aimed at IaaS users, who wish to provide services and resources to other subjects. A defining characteristic is the transparent translation of high level identities (i.e. from SaaS level), authenticated by a third party identity provider (IdP), to lower level identities (i.e. for PaaS/IaaS usage). This allows SaaS users to perform authentication on their IdP and interact with the SaaS with SSO. Furthermore, the SaaS provider is able to track the user actions on the lower levels of abstraction, as the architecture provides the means to attach a unique credential that is valid on the IaaS. This also enables the provider to create applications tailored to each user, running under their own identity (i.e. no shared application), with individual accounting.

An interceptor installed in front of the SaaS application captures the user identity, received from the user's IdP, and exchanges it for an internal token on a security token service (STS). This token contains a unique identification that is digitally signed and registered on a central repository. The interceptor attaches this token to the user's request; another component operating on the IaaS level can then, for example, start a user processes under this identification. The central repository provisions this account data to the low level components, and is able to replicate data to other IaaS providers, effectively allowing the SaaS provider to track user activity over the entire environment.

This work makes significant contributions to the field of identity management. Previous works generally deal with relatively homogeneous scenarios (e.g. every resource is a web site) or makes some assumptions (e.g. authentication is interactive and password-based). We present a proposal to tackle a much more complex scenario, allowing sharing of information through all cloud abstraction layers, as well as on environments spanning multiple IaaS providers. The end users (i.e. from SaaS) are free to use their own IdPs, while the SaaS provider translates their identities transparently. The proposal brings various security advantages, like better auditing, accounting, and facilities for access controls.

The paper is organized as follows: Section 2 discusses some related works; Section 3 describes the proposed

This work was partially sponsored by the Program Center for the Research and Development on Digital Technologies and Communication (CTIC/MCTI), grant 1313 and National Council for Scientific and Technological Development (CNPq), grants 310319/2009-9 and 478285/2011-6.

978-1-4673-0229-6/12/$31.00 ©2012 IEEE

architecture. On Section 4 we present some implementation details. Section 5 presents our conclusions.

II. Related Works

Shibboleth [2] offers federated identity management based on two main components: the service provider (SP) and the identity provider (IdP). SPs protect web resources (i.e. web sites) and establish trust relationships with IdPs that act as authoritative authenticators. Once a user is authenticated on the IdP, she can access resources protected by SPs that rely on this IdP, without need for further authentications; she just needs to inform which one is her IdP. Under the covers, IdPs and SPs exchange authentication data according to the SAML specification. Identity providers have autonomy to decide, according to their privacy policies, which SPs may access the user's authentication data. OpenID [6] resembles Shibboleth on some aspects. The building blocks are called Relying Party (RP) and OpenID providers. OpenID providers store authentication data, and RPs play a role similar to what SPs do on Shibboleth. However, the user is the one who decides who may have access to her authentication data. This approach empowers the user discretion to choose who is to be trusted and, therefore, makes the trust relationships more flexible.

Kerberos [7] is a distributed authentication service very popular for operating systems. It employs encrypted messages, following the authentication protocol by Needham and Schroeder [8], with the addition of timestamps. Kerberos provides single sign-on on systems that trust its authentication mechanisms (i.e. the authentication service and the ticket granting service). It is also possible to perform federated authentication between different security domains, if the authoritative authentication servers have trust relationships established with each other [9].

A model for distributed identity management for digital ecosystems is described on [10]. Authentication data is abstracted through the usage of user profiles that are encrypted and replicated to trusted peers. It describes a model to support dynamic digital ecosystems, with single services and service compositions.

The above mentioned works represent a good approach when dealing with a relatively homogeneous environment (e.g. only operating systems with Kerberos, or only web sites with Shibboleth). Although the proposal on [10] works with credential translation, it is heavily based on SAML. This is not easily supported on popular operating systems, thus restricting its usage mostly to web based resources and systems that are SAML compatible.

III. Federated Identity Management Architecture

First, we need to define the entities belonging to the scenario considered in this work:

- *SaaS User*: any entity that wants to access the resources exposed as a SaaS application. Her identity is unknown outside of the SaaS context.
- *IaaS Contractor*: someone that contracts resources from IaaS providers, to deploy the SaaS Application to be provided to *SaaS Users.*
- *IaaS User*: the entities recognized on the IaaS resources (e.g. operating systems) as valid accounts.
- *Contracted Resources*: infrastructure resources like processing time, disk space, and network bandwidth.
- *SaaS Application*: a system that employs *contracted resources* to provide some specific functionality to the *SaaS users.*
- *Identity Providers*: any entity that is responsible for managing the *SaaS users'* authentication data.

IaaS contractors employ *contracted resources* to offer *SaaS applications* to the *SaaS users*. It is a popular approach to sell such services in a *pay-as-you-go* manner. However, as the *SaaS user* is only known inside the *SaaS application* itself, it is hard to measure with precision how much of the contracted resources a given *SaaS user* has consumed. Without the unification of *SaaS user* identities and *IaaS user* identities, it is only possible to obtain estimated (average) individual resource consumption.

In this proposal it is important to know which *SaaS user* is executing which actions on the infrastructure level. This not only enables a fair accounting system, it also permits an accurate identification of users for fine grained auditing and access control in IaaS.

With these requirements in mind, we propose a new architecture for federated identity management that integrates the IaaS and SaaS layers. Some components are operated on the PaaS layer to conceal the implementation details of identity translation from the *IaaS contractor*. We also designed an approach to permit the *SaaS user* to use the *SaaS application* without any interference.

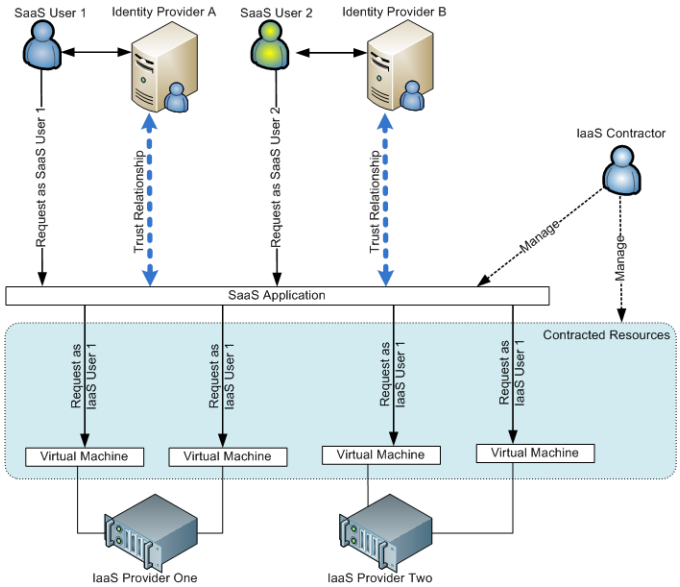

Figure 1: Example Scenario

A. Example Scenario

On Figure 1 we show a scenario involving all the entities described above. The *IaaS contractor* acquires the *contracted resources*, in the form of virtual machine instances, from two different providers (IaaS Providers One and Two). The *IaaS contractor* then deploys her *SaaS application* over the *contracted resources*; she already has the burden of managing both the infrastructure and the application. It is also important to set up trust relationships with the many *identity providers* responsible for the *SaaS users*' authentication.

Figure 1 exposes the limitation of current approaches: even though the *IaaS contractor* can delegate *SaaS users*' authentication to third party *identity providers*, she cannot be sure of which user is consuming what resources on the infrastructure. An outside view, perceives the SaaS application consumption as referring to a single entity (e.g. *IaaS User 1*). The *IaaS contractor* must design her *SaaS application* specifically to track *SaaS users*' actions, for accounting and auditing purposes, as well as for authorization and access control enforcement.

B. The Architecture

The proposed architecture is shown on Figure 2. It is composed of four identity domains: *User domain* corresponds to the security domain in which a given external user exists; *IaaS domain* is where the *contracted resources* exist, and may be spread over different *IaaS providers*; *SaaS domain* refers to the security domain of the provided *SaaS application*; The *PaaS domain* contains the components necessary to perform identity translation and data replication over different *IaaS providers*. On real world scenarios there may be more occurrences of the same kind of domain (e.g. many *IaaS and User domains*), we presented only one of them for the sake of simplicity.

A trust relationship between the given *user's domain* and the *SaaS domain* is prerequisite for the *SaaS user* to access the provided application and resources. The configuration of this relationship depends on the type of *Identity Provider* employed. The proposed architecture can support different *identity providers* (e.g. Shibboleth, OpenID) by using different application access points (i.e. *interceptors*).

The basic steps needed for a given *SaaS user* to access the provided *SaaS application* are described bellow (see Fig. 2):

1) The *SaaS user* tries to access the *SaaS application* interface with her browser.

2) The *interceptor* cannot find a valid session for the access request, thus it requests the *SaaS user* to authenticate on her choosen *identity provider*.

3) The *SaaS user* is forwarded to the *Identity Provider* authentication interface, and performs her login if needed.

4) The *Identity Provider* redirects the *SaaS user* to the *interceptor*, embedding the required *proof of authentication* (e.g. a SAML token).

5) The *SaaS user's* browser tries to access the provided *SaaS application*, though now the request has an embedded *proof of authentication*.

6) The *interceptor* validates the *proof of authentication* and requests a new *security token* from the *Security Token Service (STS)*. The trust relationship between *interceptors* and the *STS* is established by the PaaS administrator, who owns both components.

7) The *STS* verifies if the *SaaS user* is allowed on the IaaS security domain. A new token containing the *IaaS user*' *ID* is issued for valid users and is signed with the *STS* private key:

a) First time *SaaS users* get their account registered on the *IaaS Identity Provider*, this is needed only once. A unique identification is created, derived from the *SaaS user*'s identification. Thus the *STS* can always issue new security tokens whitout need for storing the IaaS identities locally.

b) The *IaaS Identity Provider* centralizes the accounts for all *IaaS users*, and shares this data with the authorized mechanisms (e.g. operating systems) eliminating the need for account creation on each virtual machine's operating system.

8) The *interceptor* forwards the *access request* to the application endpoint (e.g. an internal URL), embedding the *security token* obtained from the *STS*.

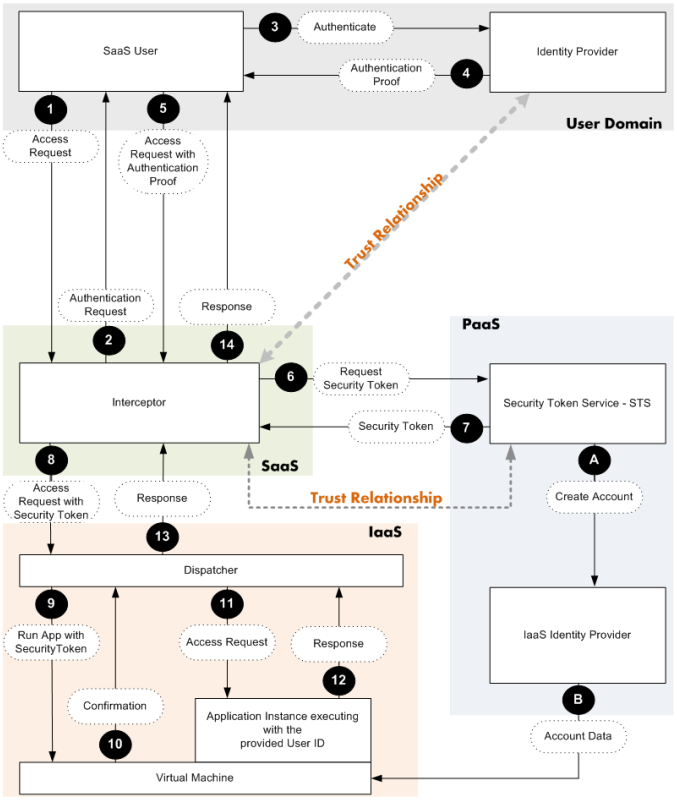

Figure 2: Architectural Overview

9) A local component (i.e. the *dispatcher*), running with administrator rights, captures the request and extracts the security token. If it is signed by the *STS*, it is considered authenticated. The *dispacher* uses the IaaS identification contained on the request to execute actions on behalf of the requesting user (e.g. to start an application instance owned by

the provided user identity). The trust relationship with the *STS* is configured by the *PaaS administrator*.

10) As the underlying mechanisms recognize the IaaS identification, the action is performed with success.

11) The *access request* is delivered to the *SaaS application instance* owned by the *IaaS user* provided on the *security token.*

Steps from 12 to 14 refer to the forwarding of SaaS application responses to the SaaS user's browser *or client application.*

The actual *SaaS user* authentication procedure depends on the solution used by the *user domain*. For approaches that offer *single-sign on*, we consider that the *SaaS client application* will be running on top of a web browser, thus inside the authenticated *SaaS user's* session. The application requests are tunneled inside this authenticated session and the *interceptor* enriches the *access request* with the *security token* from the *STS*.

For *SaaS users* who want to use custom developed applications (e.g. web service clients) it is possible to provide a separate gateway (i.e. *interceptor*). The only difference from the mentioned scenario is that the *SaaS user's* application will have to actively authenticate itself on the *identity provider* and cache the *authentication proof* for future use. Therefore, the application requests will not be tunneled inside browser authenticated HTTP sessions.

Account data stored on the *IaaS identity provider* can be replicated to other instances running on different *IaaS providers*. This allows for the unique identification of any user, no matter where the *SaaS application* instance is created, and provides what we called horizontal federation of identification. Data replication is desirable for reducing communication costs (e.g. network latency), though it is possible to set up an environment in which each *IaaS IdP* would be responsible for some subset of the identities, delegating the rest to the remaining *IaaS IdPs*.

Vertical federation is achieved by supporting different user authentication mechanisms in the front-end (i.e. *interceptor*), enabling different *SaaS users* to use their chosen identity management approaches, though maintaining identification unity through the abstraction levels by using the *STS* in cooperation with the *IaaS identity provider.*

The *dispatcher*, running on each virtual machine, acts as a proxy, performing the actions the *SaaS user* requested. It is needed to start the *SaaS application* with the user's credentials or to import (*mount*) the user's network storage devices on the target system, and so on. This is fundamental for the individual accounting, for auditing purposes, and to employ access controls tailored to specific users.

IV. Implementation Details

Bellow we present brief descriptions of the software components for the prototype implementation:

- OpenID authentication system [6]: offers a flexible solution for web authentication. It is used in the role of the *Identity Provider* from Figure 2. We selected this implementation for its ease of use and because it is a well-known framework for SSO purposes.
- JAX-WS RI [11]: an implementation of the Java API for XML Web Services. It used to create Java Applets running inside a web browser's HTTP session, making SOAP requests to the *SaaS application*.
- Apache Tomcat [12]: a popular servlet container, used for hosting web based components, like the *interceptor* and the *STS*.
- Apache Axis 2 for Java [13]: a web services toolkit used in conjunction with the Apache Rampart [14] security module to implement the *STS* component, compliant with the WS-Trust specification.
- OpenLDAP [15]: an LDAP implementation, supporting data replication to slave servers. This component is used on the role of *IaaS Identity Provider*.
- OpenSAML [16]: a popular SAML toolkit, used by the *STS* to create *security tokens* according to the SAML specification.

We are using Eucalyptus [17], an IaaS platform capable of deploying a group of virtual machines running the Debian GNU/Linux [18] operating system.

The OpenLDAP directory provides account information to the Linux operating systems through a component called NSS_LDAP. This enabled us to configure the operating systems with account information without the need for locally creating user accounts. It also enabled the data replication to other *IaaS providers* with native protocols. We also considered using a Kerberos implementation, but it seemed very inflexible if compared to an LDAP directory, because it depends on clock synchronization and user accounts must be manually created on each machine, what is not reasonable in a dynamic environment as cloud computing.

The *dispatcher* module runs as the root user, it spawns user processes which drop the administrative rights, using only the credentials provided on the *security token*. It evaluates *security tokens* in the SAML format, and considers tokens signed by the trusted *STS* to be authentic.

The *STS* implements a scheme for deriving unique user identification from the *SaaS user's* name (e.g. applying a hash function). This identification is then registered on the LDAP directory, and the operating systems are notified to synchronize their data with the directory server. It then creates SAML assertions and signs them with its private key.

The *interceptor* is a servlet that acts as a proxy. It receives requests from *SaaS users* that want to access the *SaaS application* interface (i.e. an URL pointing to a provided applet). The URL is protected by a module implementing the OpenID authentication, so the user must go through the authentication procedure, which ends up creating an authenticated session in the servlet. The *interceptor* interacts with the *STS*, and requests the *security token* using the WS-Trust specification. After successfully receiving a *security token*, it stores the token on a session attribute for further

reference; sessions that already contain a *security token* does not need to repeat this step. The token is attached to the application request before it is forwarded to the *dispatcher*.

The *SaaS user's* application is composed of an applet that executes inside an OpenID authenticated session. The application uses SOAP as its protocol, using the JAX-WS implementation. When a request is made, the *interceptor* considers it just an XML document. However, we are able to embed SOAP headers on it before it arrives on the *dispatcher*. Thus, the *SaaS user* herself is unaware of the credential translation taking place. The proposal could, as well, support Shibboleth users by providing a different *interceptor* able to interpret this protocol.

Even though we are working mainly with Linux-based operating systems, most modern operating systems have mechanisms to enable impersonating other users when given the appropriate credentials. Besides, LDAP is a very well supported technology, so the proposed architecture can adequately support different implementation scenarios.

We also devised the architecture to be application agnostic. That is, it can support SOAP based web services, HTTP based applications like CGI, REST, and so on. The use of web browsers was selected to show that it is possible to offer integral federated identity management transparently, though without losing the single sign-on functionality.

V. CONCLUSIONS

In this work we presented a new architecture and platform for federated identity management, mapping high level identities (i.e. from SaaS) to low level identities (i.e. for IaaS). By eliminating this integration gap we enable the SaaS application developer to track resource usage on a user based fashion, that is, it is possible to do fine grained accounting.

Besides, the application developer also gains advantages as: possibility of applying security policies for individual users on the infrastructure level; obtain accurate auditing records, as the users are known on the whole cloud computing layer; get the ability to deploy applications on multiple IaaS providers without losing identity unity.

The end user is not disturbed by the presence of the proposed architecture. We showed that it is possible to use interceptors to offer credential translation transparently. This also permits the proposal to support many authentication approaches in the front-end by just changing the interceptor.

The implementation aspects of prototype showed that the proposal is reasonable. There are plenty of freely available technologies and components, based on open standards, which supports the implementation of the proposed architecture.

References

[1] Shim, S.S.Y.; Geetanjali Bhalla; Vishnu Pendyala; "Federated identity management," Computer , vol.38, no.12, pp. 120- 122, 2005.

[2] Morgan, R. L.; Cantor, S.; Carmody, S.; Hoehn, W.; Klingenstein, K.; "Federated Security: The Shibboleth Approach," EDUCAUSE Quarterly, vol. 27, no. 4, pp. 12-17, 2004.

[3] OASIS; "Web Services Federation Language (WS-Federation) Version 1.2," Available at: http://docs.oasis-open.org/wsfed/federation/v1.2/ws-federation.html, Retrieved on January, 2012.

[4] Mikkonen, H.; Silander, M.; "Federated Identity Management for Grids," International conference on Networking and Services, pp. 69, 2006.

[5] Badger, L.; Grance, T.; Patt-Corner, R.; Voas, J.; "Cloud Computing Synopsis and Recommendations," Available at: http://csrc.nist.gov/publications/drafts/800-146/Draft-NIST-SP800-146.pdf, Retrieved on January, 2012.

[6] OpenID Foundation; "OpenID Authentication 2.0," Available at: http://openid.net/specs/openid-authentication-2_0.html, Retrieved on January, 2012.

[7] Steiner, J.G.; Neuman, B.C.; Schiller, J.I.; "Kerberos: An Authentication Service for Open Network Systems," Proceedings of the Usenix Conference, 1988.

[8] Needham, R. M.; Schroeder, M. D.; "Using encryption for authentication in large networks of computers," Commun. of the ACM. vol. 21, no. 12, pp 993-999, 1978.

[9] Neuman, B.C.; Ts'o, T.; "Kerberos: An Authentication Service for Computer Networks," IEEE Communications, vol. 32 no. 9, pp 33-38, 1994.

[10] Koshutanski, H.; Ion, M.; Telesca, L.; , "Distributed Identity Management Model for Digital Ecosystems," The International Conference on Emerging Security Information, Systems, and Technologies, pp.132-138, 2007.

[11] GlassFish Community; "The JAX-WS Reference Implementation," Available at: http://jax-ws.java.net/, Retrieved on January, 2012.

[12] The Apache Software Foundation; "Apache Tomcat," Available at: http://tomcat.apache.org/, Retrieved on: January, 2012.

[13] The Apache Software Foundation; "Apache Axis2/Java," Available at: http://axis.apache.org/axis2/java/core/, Retrieved on: January, 2012.

[14] The Apache Software Foundation; "Apache Axis2 Security Module," Available at: http://axis.apache.org/axis2/java/rampart/, Retrieved on January, 2012.

[15] OpenLDAP Foundation, "OpenLDAP Software," Available at: http://www.openldap.org/, Retrieved on Jan 2012.

[16] OpenSAML; "Open Source SAML Implementation," Available at: https://wiki.shibboleth.net/confluence/display/OpenSAML/Home, Retrieved on: January, 2012.

[17] Eucalyptus Systems, Inc.; "Eucalyptus Open Cloud Platform," Available at: http://open.eucalyptus.com/, Retrieved on: January, 2012.

[18] Debian Foundation; "Debian GNU/Linux," Available at: http://www.debian.org, Retrieved on: January, 2012.