AN AUTHENTICATION BROKER FOR VIRTUAL LABORATORIES

Ricardo Malta[1], José Faísca[1], José Quintino Rogado[1,2]

[1] ECATI - Computer Engineering, Universidade Lusófona
[2] CICANT Research Center, Universidade Lusófona

376, Av. Campo Grande, 1749 - 024 Lisboa, Portugal

ricmalta@gmail.com, jose.faisca@gmail.com, jose.rogado@ulusofona.pt

Abstract

This document describes a task of the ongoing Virtlab project taking place at the "*Universidade Lusófona de Humanidades e Tecnologia*" (ULHT), which aims at providing a framework for federated access to modular and configurable virtual network laboratories, based solely on open source technologies. The paper reports on the first implementation of an essential Virtlab concept, the Secure Authentication Broker, which enforces secure authentication and authorization to the virtual resources. This module allows students to access remotely, by means of a seamless e-Learning web interface, various virtual environments (Virtual Machines, Networks and Storage) which are configured according to their academic enrolment profiles. The paper provides a detailed description of the current Authentication Broker architecture and implementation choices, and explains how the mediation mechanism introduced with this module can be further used to enable the adoption of more sophisticated authenticated schema.

Keywords - Virtual laboratories, e-Learning, authentication, virtualization tools.

1 INTRODUCTION

One of the primary goals of the Virtlab [1] project is to allow students to access virtualized laboratory resources through the same e-Learning paradigm they use to retrieve the course contents and perform their general assignments. To achieve this goal, a certain number of existing technologies have to be combined, in order to provide a seamless infrastructure based on which the different platforms appear as a fully integrated environment. Among others, these include Web access (HTML, HTTP, etc. since most e-Learning tools are web based), remote access to virtualization platforms where the virtual labs are executed, and authentication / authorization functionality to provide secure and selective access to the resources.

The Secure Access Broker (SAB) acts as a mediator (or broker) between these three environments: it replaces the traditional authentication flow established between a virtual desktop and a virtual environment by an alternative schema based on HTTP, accesses an identity repository for credential validation, and consequently grants or denies access to the required resources.

This approach introduces a very flexible and extensible mechanism, which allows the adoption of various authentication models. In a first phase, a basic authentication scheme was introduced, performed against credentials stored in a local repository. The final objective, however is to provide an authentication scheme based on the Shibboleth [2] federated environment, and will be addressed in the final section of this document.

The next intermediation step performed by the Broker takes place once students are granted access to the resources, which is to provide full control of the virtual environment using the same web interface. This is achieved by embedding in an HTML page the virtual machine administration controls and a specific remote machine console applet, to which the requested virtual machine controlling endpoint is provided. The applet implements the VNC [3] protocol, which allows the establishment of a mediated connection with the virtual machine console.

Therefore, the rights acquired by the student by successfully performing a Web based authentication and authorization step, are transformed in a capability to communicate with the virtualized resource,

using a different access protocol, thus performing a delegation of rights between these two different environments.

To summarize, the Secure Authentication Broker implements the following functionality:

- Intercept the communications between users and virtual environments
- Interpret and validating the user credential
- Retrieve the user profile and validating their access to the resources requested;
- Create and manage the user authentication session;
- Delegate the web access credentials to the virtual console manager;
- Manage multiple and heterogeneous virtual environments.

The paper starts with a brief introduction to the various technologies incorporated in the Broker. Virtualization and remote access techniques are covered in section 2, and then a more detailed description of the Authentication Broker architecture, implementation choices and user interface is presented in section 3. Section 4 analyses the security conditions introduced by the mediation mechanism, and the paper concludes with a critical perspective of the current achievement and the presentation of the next milestones.

2 VIRTUALIZATION TECHNOLOGY

Virtualization is the fundamental keystone of the Virtlab concept and the reason that led to the design of the Authentication Broker. Virtualization has changed the way computing infrastructures are deployed, and the current virtualization technologies provide almost infinite possibilities for expanding systems resources. In the last decade virtualization technology has suffered an enormous evolution, and is nowadays more robust, faster and richer providing features and special purpose virtualization systems are widely available, supported by various architectures and hardware optimizations [4]

The most important virtualization techniques are the following:

- *Full Virtualization*: All the hardware and firmware are emulated, and it is therefore possible to run most kinds of operating system as guests with no modifications, although this technique may provide poor performance, since it is essentially based in software emulation.
- *Paravirtualization:* The guest system uses a virtualized version of the host hardware, but the guest operating system must be modified, in order to intercept all the guest application system calls and redirect them to the host operating system or hypervisor.
- *Fully virtualization with hardware-assistance:* This approach relies on specific hardware assisted virtualization optimizations, enabling the fully virtualized systems to take advantage of the host hardware with no need of modifications to the guest operating system. Almost all recent mid-range processors support such features, and in certain conditions this approach provides near native performance to guest environments.

Depending on the goals and requirements of the virtual environment, virtualization can be implemented on top of a fully fledged operating system like Windows or Linux, or achieved by means of a specific virtualization operating system, usually referred to as a *hypervisor*. The former is well suited for end user virtualized environments, while the latter is more suited for consolidating data center resources or creating high performance virtualized infrastructures. The Virtlab environment referred in this paper uses hardware assisted virtualization based on hypervisor technology.

2.1 The Hypervisor and Virtualization Server

The hypervisor and server virtualization environments are the most critical areas of this technology, which have been the object of more recent developments for the direct impact they have on the deployment of modern virtual data-centers.

A Hypervisor is a software platform that allows the virtualization of one or more operating systems in parallel, all sharing the same hardware, network and storage. A hypervisor is usually a "bare-metal" operating system that runs directly on the computer hardware and provides an abstraction model of all the hardware resources to the host operating systems. On top of the hypervisor, a certain number of

higher level host system services are implemented, generally referred to as the *virtualization server*, which provide an environment for configuring, controlling and managing the operation of the virtualization platform, including its CPUs, networking, storage, and providing full control of guests operating systems by means of dedicated management consoles.

Among the functionality provided by the virtualization server, unabridged access to the guest operating system console is one of the most important, since it provides the user with the ability to manage the virtualized operating system as if it were running in a real computer. In the case of the Virtlab environment, accessing the system console is crucial for computer and software engineer candidates, providing them with ability of entirely controlling their own computer(s) and operating system(s), which they can configure, optimize and tune according to the needs of their courses and work assignments.

Therefore, the use of an hypervisor architecture in this project it is a clear consequence of the necessity for creating multiple and diverse virtual environment (including pools of virtual machines connected by virtual networks), that can be suited to specific classroom needs, possibly on demand. This requirement clearly excludes another possible approach which is to create virtualized environments directly on the classroom generic computers, which in general do not have the needed hardware capabilities, and whose administration quickly becomes a terrible nightmare.

The hypervisor used in the Virtlab project is the Kernel-based Virtual Machine [5] and the QEMU/KVM [6] virtualization server, because they are based on a standard Linux modules, which are developed in open source projects and are currently a reliable path to the hardware-assisted virtualization. With the hardware extensions enabled and the QEMU/KVM Virtualization Server, it is possible to run an enterprise class full virtualization environment supporting every type of x86 and i64 operating systems at practically no costs.

However, in what concerns the implementation of the functionality envisioned for the Virtlab environment, the virtualization server must be accessed through a higher abstraction layer, allowing an ubiquitous access across the campus and also from external locations by means of a Web based e-Learning environment.

2.2 Remote Access to Virtual Environments

Generally, remote access to commercial virtualization platforms is performed by means of vendor specific protocols and methods, like in VMWare [7], which are well optimized and secure protocols, built for an architecture and line of products, but that lack the flexibility needed to suit the objectives of an academic project. On the other hand, in some open source platforms, which achieve a degree of performance and robustness comparable with the ones of commercial products, remote access is easier to control and intercept, first because all the interfaces are exposed, and secondly because they adopt standard and well documented interfaces, such as Libvirt [8].

Libvirt is a library that allows local and remote communication with virtual machine environment and completely abstracts the virtualization platform details, by exposing an API which is decoupled from any particular virtualization technology. Libvirt provides methods for all the generic operations necessary to manage virtual machines, network and storage. Internally, libvirt use multiple implementations, each targeting a different virtualization technology, each implementation being referred to as a driver.

All communications with a specific VM using libvirt occur after a binding is established with its hosting hypervisor. Internally, a libvirt driver uses whatever management channels are available for the specific virtualization technology. For some drivers, this may require that the libvirt management server runs on the managed host, directly interacting with the local hypervisor, while others may be able to communicate remotely [9].

To remotely control a virtual machine, two instances of libvirt need to be in contact: one on the remote console application, and another on the virtualization server that runs on top of the hypervisor and controls the execution of all the virtual machines on a given host. In our case this functionality is implemented by the QEMU/KVM Virtualization Server.

To connect to a remote resource, a hostname must be supplied in a specific URI format:

driver[+transport]://[username@][hostname][:port]/[path][?extraparameters]

For example, to access a QEMU server on from a remote machine we will use *qemu:///server/system/*.

From an API point of view, apart from the change in the URI format, libvirt behaves the same way for all the supported hypervisors. Normal calls are routed over the remote connection in a transparent way, and error values from a remote server are returned as if they happened locally.

The libvirt management server allows access to hypervisors running on local or remote machines through encrypted and authenticated connections. A network interface using TCP/IP socket as well as a UNIX domain socket is provided, and both interfaces can be enabled or disabled using the libvirt configuration templates.

TCP/IP socket connections can be protected and authenticated using Transport Layer Security [10]: essentially the libvirt management connection will open a TCP port for incoming connections, which are protected and authenticated using client/server x509 certificates.

The libvirt library and server can also use the Simple Authentication and Security Layer (SASL) libraries and the Generic Security Services Application Program Interface [11] plug-in to enable Kerberos [12] strong security and provide both authentication and encryption of all the remote management protocol messages.

Unencrypted connections using TCP/IP socket are also possible and can be combine with SASL username/password based scheme, known as "digest-md5" to provide session authentication and encryption. The UNIX domain socket is only accessible on the local machine, is not encrypted and uses Security-Enhanced Linux (SELinux) or UNIX permissions for authentication.

The libvirt management connection can also be protected over a Secure Shell (SSH) connection. When using SSH the entire authentication is done using public-key cryptography. The default transport, if no other is specified, is TLS.

Libvirt has bindings for several programming languages, such as C, Java or Python.

2.3 The Amazon EC2 Management Console

The Amazon Web Services Elastic Compute Cloud [13] is a general purpose virtualized infrastructure that implements the concept of Cloud Computing. Using this platform, anyone can access an unlimited pool of resources, that can grow or shrink (thus the term *elastic*) according to the variation of the users' computing and storage requirements. The service is not free, but a specific low cost pricing schema bundled with a micro charging and payment system makes it attractive even to individual users. Using this platform, it is possible to instance a single machine to host a web site or a sophisticated pool of clustered nodes.

An essential functionality to make this platform usable is the management console, which allows the users to remotely access their environments and perform all the usual management and configurations operations. The AWS offers this service based on a Web interface, on which the user has full control over the instances over the virtual environment, and in some cases, login into the virtual environment through a remote desktop.

The Virtlab remote management console uses concepts that are very similar to those used in the AWS console, mainly when considering the following points:

- Remote access using an integrated Web interface;
- Implement remote access using Web Services technology;
- Generate access capabilities from web based authentication sessions;
- Provide remote login possibilities to the virtualized systems.

The scale and scope of the two approaches are obviously different, since AWS is a globally accessed cloud computing platform developed by one of the most powerful and well known internet enterprise, and Virtlab is an academic initiative aimed at improving the e-Learning environment in the context of an university.

However, when the full range of the Virtlab specifications are implemented, mainly when an authentication schema based on federated technology is used to access the virtualized environment, we believe that the Virtlab environment can considered as an academic, freeware and lightweight version of the AWS platform, all scale and programming resources considerations obviously taken apart.

3 THE AUTHENTICATION BROKER

The Authentication Broker enables Web style remote access to virtual environments consisting of machines, networks and storage infrastructures. This section provides a detailed description of its architecture and implementation.

3.1 Architecture

The present version of the Secure Authentication Broker architecture is based on two modules which implement two different functions, as depicted in Fig. 1:

1. The Authentication and Authorization Mediator (AAM), which implements authentication and access control to the virtualized environments;
2. The Web Client (WC) which provides a Web Interface for controlling the virtual machines.

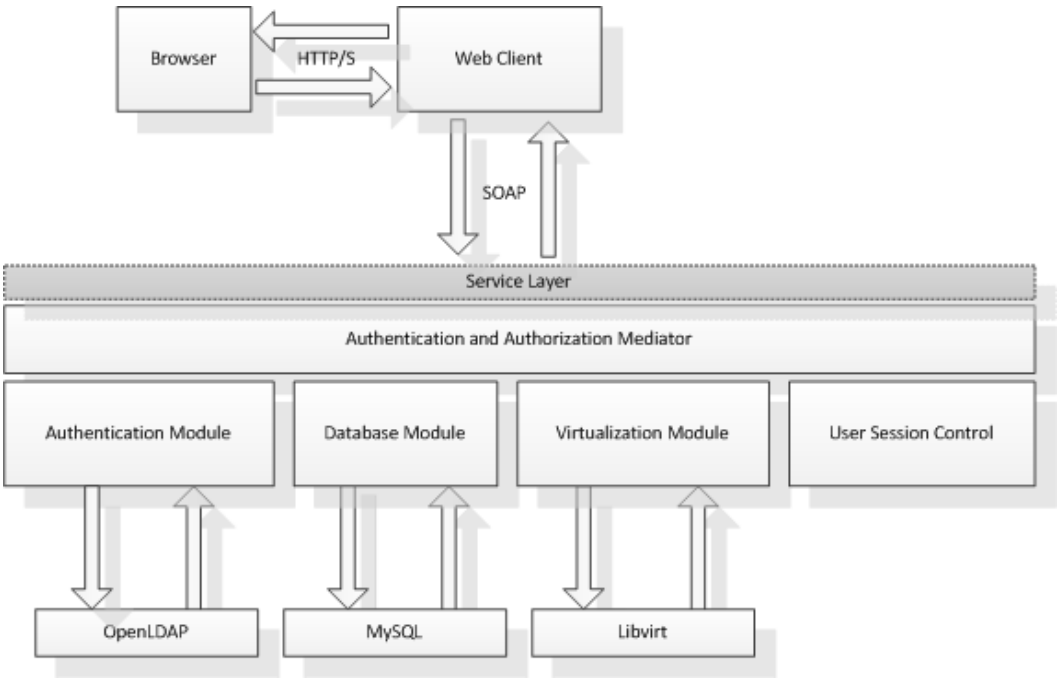

Fig. 1: Security Access Broker general architecture

A. The Authorization and Authorization Mediator

This component acts as a proxy between the students' authentication and profile information, and the Virtual Machines infrastructure, performing the following functions:

- Abstract the diversity of virtual environments;
- Abstract the diversity of authentication and profiles repositories;
- Acquire login credentials for authentication;
- Parse the student credentials and verify them against the repository;
- Index the repository data base with the user identifier to the corresponding profile
- Create and maintain the user sessions during the access period
- Delegate the authentication rights in capabilities to access to virtual resources

The Authentication and Authorization Mediator (AAM) manages four connectors, through which is interacts with the following environments:

1. The Web Client, implemented as a Web Service provider layer, through which all user requests are conveyed to the AAM
2. The user profiles repository, implemented in an LDAP server, for username and password verification, as well as for the organization of the users in groups (classes).
3. The configuration database, implemented in MySQL, which stores the users' profiles and the virtual machines configuration to which they entitled to access,
4. The virtualization environment, using the libvirt API, through which all the management operations necessary for the environment setup or originated by user requests are performed

The AAM also contains a specific module for user session management: when a user authenticates successfully, a session identifier is created that will tag all this user's subsequent requests to the SAB.

B. The Web Client

The Web Client is responsible for the creation of the interface through which the users interact with the virtual environment, providing the following functions:

- Creating the graphical environment on which all the control icons and forms are embedded. This interface can be embedded in an e-Learning tool like Moodle;
- Providing the authentication page;
- Transforming user VM management requests in interactions with the libvirt component of the AAM;
- Providing the remote console applet to be executed in the user browser when the machine starts or resumes its execution as a consequence of a user request.

The web client communicates with the AAM my means of a Web Service component using the SOAP protocol, in order to execute the functions described above.

This Web Client is developed in PHP and JavaScript and hosted in a HTTP server.

3.2 Implementation

The implementation description follows the order by which the different modules are used in a typical sequence for the establishment of an authentication session, as described in Fig. 2.

The SAB implementation was performed using free license or open source components, and the main programming languages used were Python [14], PHP and JavaScript.

Python enables the use of built-in and third-party libraries and functionalities which were used for the implementation of the mediator authentication engine.

The implementation of the LDAP connector, providing access to the user profile repository, was based on the Python LDAP library [15], which offers a high level interface for executing LDAP queries to an OpenLDAP [16] repository. These queries are used during the user authentication step (see Fig. 2) to compare the provided credentials with those stored on the repository, and to retrieve attributes that allow the creation of a full user profile, such as first name, last name, class, current graduation year, etc., and to lookup for existing students - users.

The Virtlab configuration repository is based on a MySQL database, and the corresponding connector uses the Python library for that technology.

After the authentication is performed, session keys need to be generated for each user, in order to manage their context in the platform, and avoid subsequent accesses to be intercepted by the login redirection. These keys were based on the UUID [17] format and the Python's standard library uuid module was used to generate and manage them. During a valid user session, all the messages and interactions from a given user carry the same temporary key.

The communication with the virtual machines was implemented using the libvirt-python package. This allows the execution of basic operations on the virtual machines (such as start, stop, restart and pause) and the communication between the user and the virtual machine, either through a command

line console or a graphical user interface. The virtual machine management functionality was based on concepts derived from ongoing open source projects, such as the RedHat Virt-Manager project [18]. Regarding the remote access management functionality, the same assumptions were used as in the virtual shell (Virsh) and Virt-viewer application which are tools also available from the RedHat Virt-Manager project.

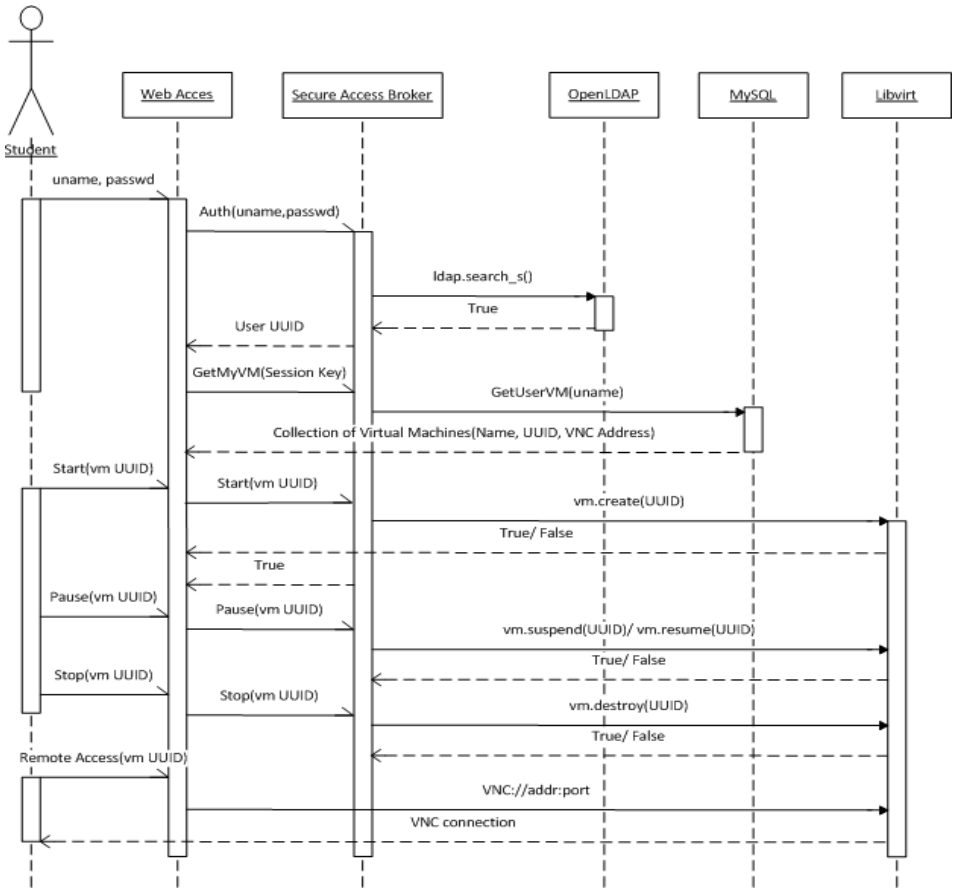

Fig. 2: Activity Diagram of the SAB functionality

The web service modules were developed using the Python WSGI [19] and SoapLib [20]. Based on these libraries, the following Web Service methods were developed, for:

- authenticating and login in to the platform;
- generating a session key;
- verifying the existence of a session;
- logout from the platform;
- identifying which virtual machines the user has permission to access;
- gaining control of a virtual machine state.

The process used for the implementation of the SAB is an excellent example of how available open source code can be easily combined, using a Rapid Application Development methodology, to build complex modules in a relatively short period of time.

3.3 User Interface

Fig. 3 presents a screenshot of the Web Interface provided by the Secure Authentication Broker. This figure shows how the environment can be accessed from a generic Browser (in this case Safari), and how, after following a generic Web authentication process, the Virtual Machine of a remote console is displayed and accessed, allowing full control of the virtualized system.

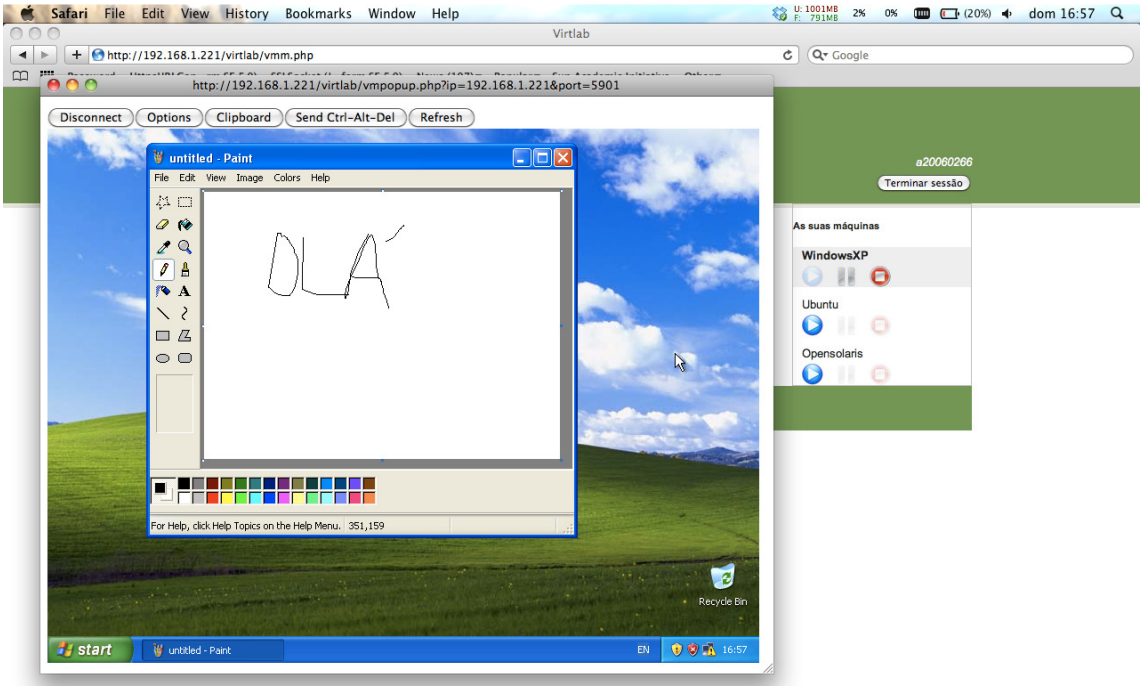

Fig. 3: SAB User Interface with the Remote Console

In the example shown the active guest system is Windows, but we can see on the right part of the image that two other systems (Ubuntu and Solaris) are available, that can be also activated if they make part of the student's academic profile.

As this picture clearly demonstrates, the access paradigm is completely compatible with most web based e-Leaning platform, and integrating it with this environment is perfectly possible.

4 SECURITY CONSIDERATIONS

As its name implies, one of the most important aspects of the Secure Access Broker is providing secure access to all the components of the architecture. Since the SAB is a distributed application which exchanges multiple messages between its main components, the security conditions of the various message paths have to be carefully analyzed in order to avoid possible holes.

If we consider the SAG architecture depicted in Fig. 1, there are two distinct domains with different characteristics with respect to the exposure to possible attacks.

The first is composed by the path established between the user Browser and the Web Client Module, which uses HTTP connections, and can be considered totally unsecure, since it traverses networks like the campus infrastructure or the Internet. The security conditions of this domain are exactly similar to those that any of us encounters when we access a net banking portal from home. The solutions for securing these accesses are well known and proven, consisting in using the SSL/TLS protocol to create secure connections, through which all HTTP messages (HTTP/S) are conveyed. We can therefore easily establish secure conditions between the user browser and the Web Client Module.

Besides, the end user never invokes the libvirt API or libvirt management server directly; it uses the SAB through its authenticated Web access. The SAB authentication mechanism will handle the end user access to the resources.

The second domain is composed by the rest of the modules (see Fig. 1), which in normal operation mode will be deployed in a security precinct behind a firewall. In the scope of this project, internal connections between the SAB, the repositories and the libvirt management server, are unencrypted and unauthenticated. As these connections are tightly controlled either by using the SAB access control mechanisms or by making the environment accessible only from a dedicated administrative LAN, we assume that this domain is well protected. In fact cannot be accessed by untrusted entities, either by using the guest OS or by using network connections to the guest OS, since these are mediated by the hypervisor. This implies that outside entities can be considered to have no access to

the libvirt management server, which results in the conclusion that the security aspects and hypothetical security flaws of the management server are not relevant. We can therefore consider that all internal traffic established behind the Web Client is secure and not exposed to external attacks.

The other fundamental aspect that remains is to secure the process of entering the internal precinct, i.e: to guarantee that the authentication mechanism and consequent decision on allowing access to the internal resources does not permit intruders to gain access to the virtualized environment. As described in the previous section and illustrated in Fig. 2, once users establish a secure HTTP/S connection with the SAG, they are asked to provide their credentials through this connection. The Web Client Module securely receives these credentials, validates them, and if they match, queries the profile data base to retrieve the Virtual Machine configurations the user is entitled to access. The user selects the virtual machine to activate and consequently, the rights obtained by a successful authentication are transformed in a capability to establish a connection with the libvirt server that controls the Virtual Machine, through a process called delegation.

The delegation takes the form of a specific identifier (or endpoint), generated by the libvirt server for each Virtual Machine created, which is sent to the user after a successful authentication. This identifier uniquely designates the specific Virtual Machine that the user has selected. Only the user receiving this identifier is entitled to establish a connection with its correspondent virtual machine, and the libvirt server only accepts one simultaneous connection with the same identifier. In the first implementation of the SAB the identifier is associated with a Virtual Machine for the whole duration of its existence, but it is relatively easy to generate a new identifier each time a virtual machine is activated, therefore preventing an intruder to capture and later reuse the identifier of an idle VM not currently in associated with any user.

These considerations taken in account, we believe the Secure Access Broker can establish sufficiently strong security conditions for a safe operation in the context of an academic campus. In the next section, an evolution path for the Secure Authentication Broker functionality is provided.

5 FUTURE WORK AND CONCLUSIONS

The Secure Access Broker described in this paper was a first important proof-of-concept for pursuing with the next steps of the Virtlab project. Basically, it validated the possibility to provide a simple and secure web access to virtual resources that can be seamlessly integrated with an e-Learning model.

Some simplifications, however, were assumed in this implementation, mainly in what concerns the authentication and access models, which need further refinement in a real and effective deployment. These were mainly: the way authentication is performed, how profile attributes are disseminated and interpreted, and how unique virtual machine endpoints are generated and managed.

As stated in Section 1, the final goal of this project is to use the university authentication and profile repositories to provide on demand access to virtual laboratory resources. This means that the authentication method needs to be fully integrated with the one used by the e-Learning tool, so that users do not have to authenticate multiple times to access both environments. The other important aspect has to do with the fact that student profiles have to be extracted from the university real repository, interpreted and mapped into specific virtual machine pools configurations.

The first aspect will be solved by replacing the simple authentication model implemented in the current version of the SAB by a more sophisticated one, which is based in a Federated Authentication Platform such as Shibboleth. The second needs an important work to be performed in the way student profiles are represented and stored, that involves both technical and administrative initiatives. Anyway, the use of a flexible platform like Shibboleth, that allows the establishment of Attribute Release Policies based on advanced and standard languages like XML, will certainly allow a clean solution for this problem.

To conclude, we believe that this achievement can be considered as an important step for future Federated access to Web applications that contributed to promote Identity Management initiatives inside our home institution. The administrators of academic infrastructures will also extract an enormous benefit from the fact that most of the class resources can be virtualized, and that secure access is generically provided only to entitled users, all this with a very low overhead. On the other hand, the cost benefit and the capacity to avoid vendor dependency by using open source solutions in what concerns virtualization software, is also worth mentioning, since it allows us to provide a high level of system maintenance and integration in the long run. We think that institutions that approach

virtualization carefully as a strategy, and not just as a simple project, will be better positioned to benefit from this technology in the future.

References

[1] Quintino Rogado, J., 2009, “VirtLab: Virtual Laboratories in Federated Environments”, V International Conference on Multimedia, Information and Communication Technologies in Education, Lisboa, Portugal, April 2009, http://www.formatex.org/micte2009/book/593-597.pdf.

[2] Carmody, S., “Introduction to Shibboleth and Phases for Deployment“, EDUCAUSE Seminar on Flexible Web-Based Authentication and Authorization, June 2007 Portland, Oregon, http://net.educause.edu/Proceedings/12960.

[3] Tristan Richardson; Quentin Stafford-Fraser; Kenneth R. Wood; & Andy Hopper (January/February 1998). "Virtual Network Computing". *IEEE Internet Computing* ***2*** *(1): 33–38. http://www.cl.cam.ac.uk/Research/DTG/attarchive/pub/docs/att/tr.98.1.pdf.*

[4] Leung, F. et al, "Intel® Virtualization Technology: Hardware Support for Efficient Processor Virtualization" Intel Technology Journal, August 2006, www.intel.com/technology/itj/2006/v10i3.

[5] Kernel Based Virtual Machine, December 2009, http://www.linux-kvm.org/page/Main_Page

[6] QEMU: a Generic Open Source Machine Emulator and Virtualizer, http://wiki.qemu.org

[7] VMWare Documentation: “Remotely Managing Virtual Machines with VMWare GSX Server”, http://www.vmware.com/support/gsx3/doc/manage_remote_gsx.html

[8] Libvirt Documentation “The Libvirt API Concepts”, December 2009, http://libvirt.org/api.html

[9] Libvirt 0.7.3 Application Development Guide - A guide to application development with libvirt Edition 1, Red Hat, 2009, http://berrange.fedorapeople.org/libvirt/appdev-guide/html

[10] The TLS protocol, RFC 2246, IETF Specification, http://www.ietf.org/rfc/rfc2246.txt

[11] The Generic Security Service API, IETF Specification, http://www.ietf.org/rfc/rfc2078.txt

[12] Kerberos: The MIT Network Authentication Protocol, http://web.mit.edu/Kerberos

[13] The Amazon Elastic Compute Cloud, http://aws.amazon.com/ec2

[14] The Python programming language, http://www.python.org

[15] LDAP client API for Python, September 2009, http://www.python-ldap.org

[16] An Open Source implementation of the Lightweight Directory Access Protocol, http://www.openldap.org

[17] A Universally Unique Identifier Uniform Resource Name Namespace, IETF RFC 4122, July 2005, http://www.ietf.org/rfc/rfc4122.txt

[18] Managing Virtual Machines, Red Hat Documentation, http://virt-manager.et.redhat.com

[19] Web Server Gateway Interface, http://www.wsgi.org/wsgi

[20] Python SOAP Library, September 2009, http://trac.optio.webfactional.com