Jan Meizner*, Maciej Malawski**, Marian Bubak***

FLEXIBLE AND SECURE ACCESS TO COMPUTING CLUSTERS

The investigation presented in this paper was prompted by the need to provide a manageable solution for secure access to computing clusters with a federated authentication framework. This requirement is especially important for scientists who need direct access to computing nodes in order to run their applications (e.g. chemical or medical simulations) with proprietary, open-source or custom-developed software packages. Our existing software, which enables non-Web clients to use Shibboleth-secured services, has been extended to provide direct SSH access to cluster nodes using the Linux Pluggable Authentication Modules mechanism. This allows Shibboleth users to run the required software on clusters. Validation and performance comparison with existing SSH authentication mechanisms confirm that the presented tools satisfy the stated requirements.

Keywords: *clusters, security, single sign-on, federations, PAM modules, SSH, Shibboleth, SAML*

WYGODNY I BEZPIECZNY DOSTĘP DO KLASTRÓW OBLICZENIOWYCH

Badania opisane w tej publikacji zostały przeprowadzone w celu zapewnienia wygodnego sposobu zabezpieczenia dostępu do klastrów obliczeniowych za pomocą federacyjnego mechanizmu uwierzytelniającego. Wymóg ten jest szczególnie istotny w odniesieniu do naukowców wykorzystujących zarówno otwarte oprogramowanie, jak i komercyjne oraz własne pakiety (np. chemiczne lub medyczne), uruchamiane bezpośrednio na węzłach obliczeniowych. Nasze poprzednie rozwiązanie, umożliwiające aplikacjom nie-webowym używanie usług zabezpieczonych mechanizmem Shibboleth zostało rozbudowane tak, aby zapewnić bezpośredni dostęp poprzez protokół SSH do węzłów klastrów, za pomocą mechanizmu „Pluggable Authentication Modules" Linuksa. Umożliwiło to użytkownikom shibbolethowym uruchamianie niezbędnego oprogramowania zainstalowanego na klastrach. Proces walidacji oraz porównanie wydajności z istniejącymi mechanizmami uwierzytelnienia dostępnymi dla protokołu SSH wykazał, że opisywane narzędzia spełniają postawione przed nimi wymagania.

Słowa kluczowe: *klastry, bezpieczeństwo, mechanizm jednokrotnego uwierzytelnienia, federacje, moduły PAM, SSH, Shibboleth, SAML*

* ACC CYFRONET AGH, ul. Nawojki 11, 30-950 Kraków, Poland, jan.meizner@cyfronet.pl

** ACC CYFRONET AGH, ul. Nawojki 11, 30-950 Kraków, Poland, and Department of Computer Science AGH University of Science and Technology, al. Mickiewicza 30, 30-059 Kraków, Poland, malawski@agh.edu.pl

*** Department of Computer Science AGH University of Science and Technology, al. Mickiewicza 30, 30-059 Kraków, Poland, and Informatics Institute, Universiteit van Amsterdam, 1081 HV Amsterdam, The Netherlands, bubak@agh.edu.pl

1. Introduction

One of the main challenges when dealing with distributed system security is the need for a solution which would not only sufficiently protect such systems [21, 3], but remain simple and flexible enough for all classes of scientific users. The most important issue which we adress in this paper is to enable users from multiple organizations to obtain direct access to computing resources to run their applications without the need to use any Grid middleware and certificates, and without wrapping the applications in specialized services. One example involves accessing a computing cluster at a remote center in order to run applications such as Gaussian or Fluent. Another scenario is acquiring direct shell access to a set of computing nodes of a cluster using some form of cloud computing mechanism [12]. Given direct access to computing resources, scientists would be able to use custom scripts to e.g. find HIV mutations or perform protein analysis [4] almost as easily as on local resources. Such scenarios are of great importance for computational e-infrastructures [24], including (among others) PL-Grid [20].

The main goals of the presented research can be summarized as follows:

- analyze existing security solutions for managing access to computing clusters,
- propose a new method for providing scientists from multiple organizations with manageable, direct access to computing clusters without the need to use any Grid middleware, as many scientists rely on traditional shell tools to run their software,
- elaborate a method which can be easily integrated with Web-based clients,
- implement and test the proposed tool, and measure its performance.

These objectives imply many lower-level technical goals, which can be expressed in the form of functional and non-functional requirements for the system. The functional requirements are:

- secure authentication mechanism for cluster nodes,
- direct shell access and execution of unmodified software on these nodes,
- secure credential delegation mechanism,
- flexible management of user credentials, including support for decentralized and federated authentication systems.

The non-functional requirements are:

- easy access for users, without complex management of credentials (e.g. certificates) and authentication,
- efficiency of the solution,
- rapid deployment of the system on cluster nodes,
- ability to deploy most system components on the server side,
- minimizing OS modifications on the cluster nodes.

In this paper we present a new security tool based on the analysis of the above requirements and existing solutions. The proposed authentication mechanism extends

the one [14] created for the Virolab Virtual Laboratory [29, 4, 13], which is based on the widely-used *Shibboleth* [11] framework. It applies the *Linux Pluggable Authentication Module (PAM)* [1] mechanism to enable interaction with various PAM-aware software standards (including the *OpenSSH* [16] server). By combining Shibboleth with PAM mechanisms we can also take advantage of Web-based federated authentication systems and the easy way in which they integrate with computing clusters.

Thus far, the only resources supported by the security framework [14] were those specifically adapted to use Shibboleth. For example, it was necessary to wrap user applications as secure Web services or components [7] which could be remotely invoked. In such a case the Web service or component container is responsible for handling security issues. The novelty of our new approach is that generic applications installed on cluster nodes do not require any modifications and can be accessed directly by using the *SSH protocol*.

The paper is organized as follows. The following section describes related work. In Section 3, an architecture of the system is presented, including both a general overview and a detailed description of the authentication process. Section 4 describes basic components used to create the system, allowing secure access to clusters. Implementation details are covered in Section 5, with focus on programming languages and software libraries used. Subsequently, in Sections 6 and 7, the methodology and results of functionality and performance tests are shown. The final section presents conclusions as well as an outline of future work.

2. Related work

In this section various security frameworks and tools are analyzed in order to determine the extent to which they meet our requirements stated in section 1. Starting from standard authentication mechanisms for Linux, i.e. the *pam_unix module* and other general-purpose systems such as *pam_ldap* or the *key pair* method used in *OpenSSH*, we also analyzed more specific solutions designed for distributed environments or computing grids, such as *GSI*, *ShibGrid*, *GridShib* or *OpenID*. *XtreemOS* [5] security mechanisms are used as an example of a complete Grid-oriented Operating System. Finally, more complex hardware-based methods are also presented. In order to check if the existing solutions meet our requirements (see section 1), we evaluated these systems with respect to the following criteria: the need to alter OS of computing nodes (which is not allowed), levels of security (key factor), scalability, and manageability for users and for service providers.

Flat-file authentication provided by the *pam_unix* module from the PAM library [1] is a basic authentication method in many UNIX-like systems including Linux. It uses user information and credentials stored in a single file `/etc/passwd` (for security reasons, hashed passwords are usually stored in a separate file `/etc/shadow`). This module should always enable critical users to access the system if more complex authentication methods fail. However, this authentication method is not scalable. In our opinion, this tool provides a moderate level of security, as it requires long-term

credentials (passwords) to be provided for each authentication attempt to any node. This potentially allows eavesdropping (e.g. by malware) to gain illegal access. It also provides moderate managability due to the lack of a Single Sign On mechanism, which, however, is counterbalanced (to some degree) by a simple authentication and credential management process. The manageability level for service providers is low, as any operation on user accounts requires updating flat files on all nodes.

Plain *LDAP* based authentication (e.g. *pam_ldap* [18]) is useful for distributed systems as it provides a mechanism to store and update user-related information in a central location, and, at the same time, use them for all nodes. This prompts high manageability for providers. However, it is only well suited for organizations which employ common user registration systems as there are no federation mechanisms, and, consequently, its scalability level is moderate. Grid5000 is an example of a system using *LDAP* for authentication purposes [28]. The central *LDAP* directory is also the basis for the security solution used in PL-Grid [20]. As this method exploits the same type of credentials as *pam_unix*, its end-user security and manageability remain moderate.

OpenSSH [16] offers the ability to authenticate users based on a pair of keys. A private key is stored on a user machine and a public key has to be added to the `authorized_keys` on the server. The private key may be encrypted or unencrypted. The unencrypted-key scenario is commonly used because it is highly convenient, allowing users to obtain access without the need to provide a password (high level of manageability). This feature allows fully automated authentication from one node to another, but due to the lack of private key encryption, its security level remains low. On the other hand, using encrypted keys is highly secure because a would-be attacker needs to obtain both the key and its associated password. However, the use of encrypted keys prevents automated authentication (moderate level of manageability for users). Both scenarios are highly manageable for system providers (no need to set up any specialized infrastructure). The scalability level, in our opinion, is average, as the method requires distribution of keys to machines.

Software tools using X.509 certificates and private keys are commonly employed in grid systems (e.g. Grid Security Infrastructure [8]). This method might provide a Single Sign On mechanism through the so-called proxy certificates (which are signed by the user's private key and then delegated). It is also quite straightforward for users when they already have certificates/keys installed on their machines. However, the certificate acquisition process, which includes generating a request, sending it to the RA/CA and then installing the returned certificate, is too complicated for non-technical users. As a result, we can rate its manageability level for users as moderate. It is also very complex for service providers (poor manageability) as it requires keeping PKI infrastructure elements (CAs, RAs), and following complex procedures to ensure adequate (high) levels of security. This method is highly scalable as it doesn't require local user databases. GSI-OpenSSH [26] allows users to authenticate with a special version of the OpenSSH server using GSI credentials.

The ShibGrid project [25] aims at integration of the traditional GSI model based on X.509 certificates with a Shibboleth infrastructure. Its goals are to support users holding standard grid certificates (issued by a national CA and stored in a Shibboleth-protected MyProxy [27]) as well as generating low-assurance certificates for users with simple Shibboleth accounts. This method could be paired with the one described above to provide interoperability between Shibboleth and GSI. Such integration would, in turn, enable high levels of manageability for users and high levels of security (inherited from both models). However, maintaining both PKI and Shibboleth infrastructures would be even harder than operating PKI, so service provider manageability levels could only be described as very poor. Both PKI (as shown above) and Shibboleth (owing to its federation mechanisms) are highly scalable.

GridShib [23] is another method aiming at integration of GSI and Shibboleth, maintained by the institutions responsible for development of both technologies. GridShib uses MyProxy online CA to issue short-lived certificates instead of standard proxy certificates for users with Shibboleth accounts and without real grid certificates. As in the case of ShibGrid, this tool was considered for pairing with GSI. However, much like ShibGrid, it would add an unnecessary layer complicating the solution. As it is conceptually higly similar to ShibGrid, we rated all its aspects on par with ShibGrid.

OpenID [17] is an identity management framework which allows users with OpenID credentials from any of the providers to access various websites. However, in contrast to Shibboleth, OpenID IdPs are not controlled by any kind of federation that would ensure validity of user information. Thus, no user can be trusted to access the production resources based on an OpenID credential. This prompts a very low level of security. For this reason such a method is not appropriate for use in the framework described in this paper. Nevertheless, a potential future enhancement of our work might be to use OpenID to create a PAM module similar to pam_shib to allow even more simplified, limited access to some demonstration parts of the infrastructure with high levels of user manageability. Provider manageability is likewise high, as there is no need to maintain custom Identity Providers. Scalability also remains high, similarly to Shibboleth.

XtreemOS [5] is a Grid-oriented, Linux-based operating system, eliminating the need to use any grid middleware. It uses various standard Linux security mechanisms such as PAM and NSS to provide support for VO-based authentication, authorization and session management (high levels of manageability for both users and providers, high levels of scalability). The main goals of this system are highly compatible with ours, as it enables users to run their unmodified application on the grid in a similar way to executing it on local machines. However, we cannot use this system as one of our requirements (see section 1) prohibits large-scale modifications (which, of course, include replacement) of the nodes' OS. The security of this method depends on the applied mechanisms but in a default scenario remains quite high.

Hardware-based security systems rely on special cryptographic devices. The most common are smartcards or security authenticators capable of periodically generating

and displaying new passwords (like RSA SecurID [22]). Smartcards hold cryptographic chips that are capable of signing and decrypting information. An advantage over software solutions is that the private key is never moved out of the chip which ensures a very high level of security. However, this type of system is more expensive then a software-based one. Additionally, the process of issuing hardware-based certificates is more complicated and cannot be performed electronically. In the case of SecurID-like authenticators in addition to the cost and need for physical delivery of the device, its nature limits single sign-on duration to the interval between password changes, which is too short. As a result, the level of managability is low for users and very low for providers. The scalability is high for smartcards (akin to software certificates) and low for SecurID-like devices (very brief authorization).

The summary of the evaluation is presented in Table 1.

Table 1
Comparison of existing security solutions with respect to the criteria relevant to the requirements

Technology name	OS change required	security level	scalability	manageability (user)	manageability (provider)
pam_unix	no	medium	v. low	medium	low
LDAP	no	medium	medium	medium	high
OpenSSH					
unenc. keys	no	low	medium	high	high
enc. keys	no	high	medium	medium	high
X.509	no	high	high	medium	low
ShibGrid	no	high	high	high	v. low
GridShib	no	high	high	high	v. low
OpenID	no	v. low	high	high	high
XtreemOS	yes	high	high	high	high
Hardware					
smart cards	no	v. high	high	low	v. low
SecurID	no	v. high	low	low	v. low

Based on Table 1 we see that XtreemOS cannot be applied in our case, since it requires exchanging the whole operating system. OpenID and OpenSSH using unencrypted keys provide, in our opinion, insufficient security for accessing computing clusters. We also discarded options characterized by below-average manageability and scaleability. This leaves LDAP and OpenSSH with encrypted keys. However, in addition to these factors, we require a tool which could be smoothly integrated with our existing solution (which is Shibboleth-based). As additional layers of compatibility might lower the already-lacking manageability levels of candidate solutions, we decided to create a new tool which would extend existing software and prove sufficient for our new requirements.

3. Architecture of secure access to clusters

The proposed system, providing secure access to clusters, is composed of elements that could be classified as server- and client-side modules. Server-side modules are related to the Shibboleth-based framework [11], as well as to the provided services. All server-side modules need to be set up by the organization. Client-side modules are used to enable access to the system. They might be provided by the organization which could place them on some kind of user interface node, or they might be installed by users themselves on their machines. The architecture is shown in Figure 1.

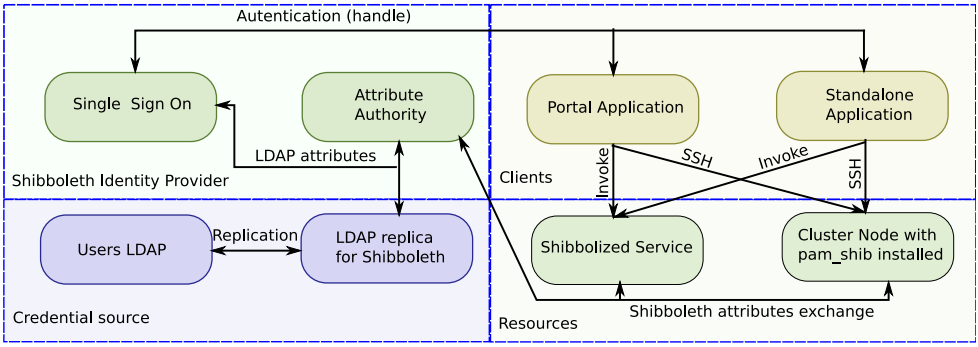

Fig. 1. General architecture of the described solution, showing its components: credential source, Shibboleth Identity Provider, resources and client tools

Most of the modules are placed on the server side. This ensures that users do not need to install complex software on their own computers. In Figure 1 the server-side modules are: *Credential source, Shibboleth Identity Provider* and *Resources*. The *Shibboleth Identity Provider*, which is a part of the Shibboleth framework, is composed of the *Single Sign On* and the *Attribute Authority*.

The *Single Sign-On* module of the IdP is responsible for issuing security tokens (called *handles*), based on provided credentials. Those tokens are used to authenticate their respective holders. Based on each handle, the Attribute Authority releases attributes that are used by the *Service Providers* for authorization. The credentials and attributes used by the *Shibboleth Identity Provider* are stored in the LDAP directory which acts as the *Credential source*. To ensure that compromising the infrastructure does not alter the original credentials or attributes, the LDAP server might be a local replica of a main server. The simplified interactions between components are shown in Figure 2. For clarity, both web and non-web clients are labeled as "Client", and both types of resources (Shibbolized service and cluster node) are labeled as "Resource".

Resources are divided into specifically Shibboleth-enabled (*Shibbolized*) services (like those provided in the ViroLab virtual laboratory [4, 14]) and computing nodes. *Shibbolized* services provide some application-specific functionality and their security mechanism is built into each service by its creators.

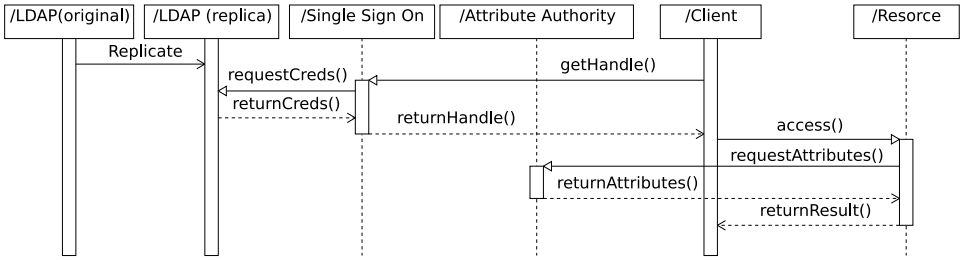

Fig. 2. Sequence diagram showing interactions between system components such as LDAPs, Identity Provider Elements (SSO and AA), clients (portal-based or standalone) as well as services (Shibbolized service or SSH accessible node protected by the pam_shib module)

Computing nodes are able to run any arbitrary software and their protection mechanism is based on the Linux PAM mechanism [1] (described in section 4). To achieve this functionality, we developed a *pam_shib* module, responsible for performing user authorization. Its implementation details are described later in this paper. The module allows system administrators to deploy standard OpenSSH [16] servers that do not need to be modified.

The architecture features two types of client applications – web-based ones, which follow the standard Shibboleth authentication protocol, and standalone ones (run on users' computers) which follow a specific authentication protocol [14]. These protocols are used to request a handle required to connect to the services. Both types of client applications can be used to access both types of resources (Shibbolized ones and standard applications on *pam_shib* protected nodes). The specific protocol of the standalone tools is based on the *ShibIdpClient* [14] command-line tool and performs automatic authentication to the IdP, followed by retrieval and extraction of the handle from the returned response. This is different from the HTTP POST and redirection schema used by the standard Shibboleth web clients. For our new mechanism, the IdP client has been augmented with the functionality needed for accessing the *pam_shib* protected nodes. Standalone client tools are also supplemented by a modified version of the OpenSSH client. The introduced modifications allow it to automatically use the Shibboleth handle obtained by the IdP client.

The *SSH* connection details using *pam_shib* are shown in Figure 3. Following a connection from the SSH client, the OpenSSH server passes the credentials to its module acting as the Policy Enforcement Point (PEP) (1). Subsequently, the PEP moves them to the module using PAM API (2) to contact its counterpart located at the *pam_shib* module (3) to request an authentication decision from the Policy Decision Point (PDP) (4). This part differs from the standard Shibboleth protocol designed for web-based tools as the credential is passed as a password, not by using HTTP redirection. The PDP uses the ShibAAClient library based on OpenSAML (5) to create a SAML assertion used to request attributes from the Shibboleth Attribute

Authority (AA) (6), which acts as the Policy Information Point. The AA also returns attributes as a SAML assertion (7). This assertion is then decoded by the ShibAA-Client and the extracted attributes are passed to the PDP (8). These steps (5-8) use the same protocol as the standard Shibboleth use case. Based on the attributes and its own configuration, the PDP reaches an authorization decision which is returned via the PAM API (9, 10 and 11) to the PEP. The PEP grants or denies access based on the authorization decision, returning proper data to the SSH Client (12). The final steps are also specific for our tool. The configuration of all modules is stored in a special configuration file (*Config*). The Shibboleth IdP configuration is stored by the IdP.

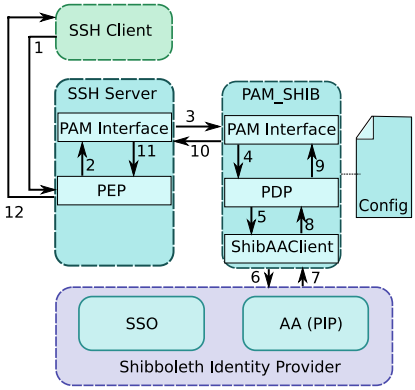

Fig. 3. The SSH connection process with the *pam_shib* module, together with the internal steps necessary to perform user authorization based on the supplied Shibboleth handle

In our opinion, the usage of the Shibboleth infrastructure makes the system highly manageable, as the federation feature allows administrators to keep credential and attribute databases limited to their own users, while allowing them to access resources provided by various partners. This architecture also provides the ability to arbitrarily select access control granularity. With an atribute-based authorization, the administrator can either provide fine-grained control (limiting access to specific individuals based on personal atributes such as uid or e-mail address), or more coarse-grained control (e.g. based on institution or role names). The configuration of the PAM module is simple, so, in our opinion, configurability remains quite good even though it has to be performed on all nodes. However, as part of future work, we plan to add better configuration mechanisms.

4. Basic security components

This section describes software elements that were used to construct our system.

The Linux Pluggable Authentication Modules (PAM) [1] mechanism enables development of authentication-related modules that can be used by various services. By

providing a unified API, PAM enables any module that supports custom authentication (or related) schema to be used by various, unmodified applications. A PAM module may support any combination of the following tasks: *authentication, account, session and password* [1]. The basic *authentication* task is the most important one from our perspective. The *account* task is responsible for restricting access to the system but not on the basis of authentication. The *session* task is used to perform some actions prior to granting access to the system, or immediately thereafter. The *password* task is executed if PAM-aware software attempts to change user credentials. The current version of our module uses the *Shibboleth* infrastructure only for the purpose of authentication, therefore only the *authentication* task is implemented by our *pam_shib module*. PAM also allows manageable configuration by system administrators. The administrator can decide (by editing the proper file) which services are to use which PAM modules and what happens if one of the modules fails. The administrator can also supply additional parameters for the module (e.g. the name of the configuration file). The configuration of actions taken upon module execution is very flexible. The administrator might apply predefined conditions or build custom ones. The ability to provide such an elastic configuration is crucial for our module as it allows administrators to apply standard authentication methods (like pam_unix) for local users, while also allowing Shibboleth access via the pam_shib module. PAM-based methods have been used to provide distributed authentication solutions in grid projects including the Grid5000 [28].

The OpenSAML [10] library allows developers to create or process *SAML* [15] assertions. These include authentication and attribute assertions used by the Shibboleth infrastructure for secure communication between its components. The library version used for the project supports both *SAML1.0/1.1* protocol used in Shibboleth 1.x and *SAML2.0* used in Shibboleth 2. C++ and Java implementations are available.

The elements of the Shibboleth [11] infrastructure might be located at various federated institutions (Home Organizations) and each institution can set up components called Identity Providers and Service Providers. The Shibboleth-based infrastructure may span any number of HOs, each of which can include IdP, SP or both. SP authenticates and authorizes users based on information (attributes) exchanged securely via the SAML [15] protocol. Those attributes, as well as user credentials required to protect the IdP, are stored in some kind of database or directory service.

5. Description of implementation

The most important elements of our software solution are the *pam_shib module* and the *shibaaclient library*. These components were implemented from scratch in the course of the work described in this paper. Additionally, the *ShibIdpCliClient* tool developed for the ViroLab project, had to be extended and the *OpenSSH client* slightly modified. Other parts of the system were built using standard software and required proper integration.

The *pam_shib* module is provided as a standard Linux shared library, implemented in ANSI C. It implements PAM *auth* functionality. To conform with the Linux PAM standard, it exposes the *pam_sm_authenticate* and *pam_sm_setcred* functions. The required authentication functionality is implemented by the *pam_sm_authenticate* function and other functions called from it. The shibaaclient library is used to request attributes from the Shibboleth AA. In addition, the module implements a simple configuration system by providing a specific C structure (PAMShibConf) and a couple of functions used to access it. Logging functionality is supported through the syslog, so all information can be logged to standard system logs with appropriate priorities. Module configuration remains highly manageable – all that is needed is to set up *pam_shib* like any other PAM module, and to create a configuration file for it. This file is then used to set the parameters related to the Shibboleth IdP configuration.

The Shibaaclient library provides easy access to Shibboleth attributes. It is implemented in C++ and its API is reduced to a few functions including the *shibGetAttributes* function used to retrieve the Shibboleth attributes for the provided handle. The library uses *OpenSAML* [10] to process *SAML* assertions and extract attributes. It retrieves assertions by connecting to the Shibboleth AA with the help of the cURL [6] library. Additionally, the Xerces-C++ [2] library is used as the XML parser.

The ShibIdpCliClient [14] is implemented in Java. It uses standard libraries provided by the Sun JDK as well as a Java version of the OpenSAML library to request and extract handles. The tool was extended to include the capability to save the handle for future use, and to return it in an appropriate format for our new system. Bash scripts responsible for further simplification of sign-in and sign-out procedures are also provided.

The *pam_shib* module requires passing system user names and Shibboleth handles as credentials. As the module is our custom creation, this scenario is not supported by the mainstream OpenSSH client. Such a client might be used without modification, but this would require passing the handle as a password. As this is not convenient, the client was altered so that it can read the handle from a file and then use it. The modification was made in such a way that it doesn't disable any standard authentication method. We also make sure that if handle-based authentication fails, the client can fall back to standard functionality (asking for a password). The additional code uses OpenSSH client logging functions so that users running the client in verbose mode can be informed about Shibboleth-related progress, warnings or errors. As the OpenSSH client is implemented in C, its modifications also need to use this language.

6. Validation

The most important task related to validation was to ensure that the system provides access to valid users only and that it does not interfere with standard (password-based) authentication mechanisms.

An overview of the validation environment is shown in Figure 4. It consists of three computer nodes connected via Ethernet. One node (A) acted both as an SSH

server and as a User Interface node. Server components (including the *pam_shib* module and the standard OpenSSH server), along with a patched OpenSSH client, were deployed an all nodes. The Shibboleth IdP client was installed on the UI node. All nodes accessed the Shibboleth infrastructure via the Internet.

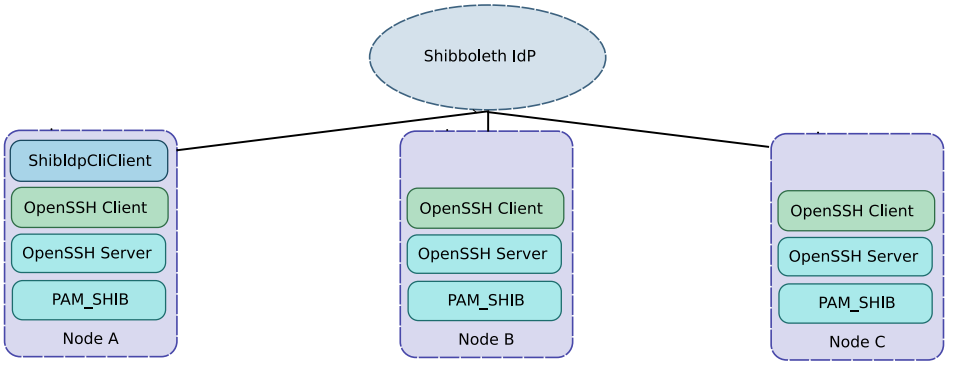

Fig. 4. The validation environment is composed of three nodes. All of them act as server nodes while one also performs the functions of a client node (A), with appropriate components deployed

During validation the following actions were performed:

1. Shibboleth authentication attempt using incorrect credentials,
2. Shibboleth authentication attempt using proper credentials,
3. series of SSH connection attempts – from node A to B, B to C, C to A and again from A to B,
4. a user was logged out (effectively, the Shibboleth handle was removed),
5. an SSH connection from A to B was attempted.

The actions listed above were chosen specifically to check the system's reaction to challenges specified at the beginning of this section, and the results were completely satisfactory. Authentication attempts with invalid credentials failed, and those with proper credentials succeeded. After being authenticated, the user was able to connect smoothly to subsequent nodes, which means that the delegation mechanism works properly. Finally, upon removing Shibboleth credentials (the handle), the user was asked for a password, and was able to obtain access to the node only after providing a proper password for the local account (checked against the `/etc/shadow` file stored on the nodes). This ensures that after removal of the handle, the *pam_shib* no longer allows access to the node, which is the expected behavior, and also that the default *pam_unix* authentication isn't affected by the described module.

7. Performance evaluation

The performance of the *pam_shib* was evaluated to ensure that it is feasable for use in a production environment. This module was chosen for performance testing as the most important part of the system. For the purpose of evaluation, a single node was used, running both server and client software. Connections were effected using the local loopback interface. This allowed simulating normal conditions and eliminated network-related delays in SSH connections. The Shibboleth infrastructure was placed in a remote location, so network delays had to be taken into account while interpreting the results as the *pam_shib module* needed to connect to it.

Performence evaluation of the module was conducted by running a sample application with a very short execution time via the SSH connection, and timing it 10 times. For this purpose the *hostname* application was chosen and its local execution time was recorded 10 times so it could be averaged and subtracted from previous measurements in order to derive the *SSH connection* time. For reference, an additional benchmark was performed using the *RSA key-pair* authentication method (without key encryption).

The average connection and remote *hostname* execution time was equal to 0.794 s. For reference, the *RSA key-pair* solution had an average response time of 0.385 s. After subtracting average *hostname* execution time (0.003 s), the *SSH connection* time for *mod_pam* was 0.791 s and for *key-pair* – 0.382 s. In our opinion, intervals below 1 s, being only 2 times higher than for the highly optimal *key-pair* solution, are acceptable. It is important to notice that the *pam_shib* module relies on a much more complex architecture then the key-pair solution. Specifically, the module must connect to the remote authentication service (the Shibboleth IdP) and process the retrieved SAML assertion. This renders the described method much more secure then the one based on unencrypted keys, but at the price of making the authentication process a bit slower.

8. Conclusions and future work

The validation process has shown that all our research goals (see section 1) were achieved. The system allows users to obtain Shibboleth-protected access to various generic software packages installed on clusters. This can include commercial software such as Gaussian or Fluent and custom programs or scripts developed by scientists.

We have also identified and successfully overcome certain obstacles related to the presented components. The most important issue here is that the OpenSSH server does not fully follow the Linux PAM standard. Despite the fact that the standard allows the PAM module to update the user's name and explicitly instruct application developers to check if this value hasn't changed, OpenSSH developers have decided to ignore it. As a result, we weren't able to use the Shibboleth user name for the *SSH connection* and map it to the system user name. Instead, we need to perform the

opposite mapping (from the system user name to the Shibboleth user name). Such mapping is required as system user names may not always match Shibboleth names.

In the future we plan to extend our tool in many ways. First, we will extend the module and client configuration options to allow more complex user name mappings as well as using a single configuration for multiple identity providers. It would definitely be helpful to introduce more dynamic authorization, e.g. by implementing or using existing NSS modules, similarly to the authorization solutions used by Grid5000 [28] or proposed recently in [19]. Another important challenge would be integration with a Web-based SSH client. A combined system could then be fully integrated with the GridSpace2 [9] platform. Another possible extension could make the described system act as an authentication mechanism for virtual machines used in cloud systems. The general concept of using such federated, secure and manageable methods for clouds seems to be very promising.

Acknowledgements

This work was partialy supported by the PL-Grid [20] project: POIG.02.03.00-00-007/08-00, www.plgrid.pl. Maciej Malawski acknowledges support from the UDA-POKL.04.01.01-00-367/08-00 grant from AGH. The authors are grateful to Piotr Nowakowski for his valuable suggestions.

References

[1] Morgan A.G.: *Linux-PAM.* `http://www.kernel.org/pub/linux/libs/pam/`, 2010.

[2] Apache Software Foundation.: *Xerces-C++.* `http://xerces.apache.org/xerces-c/`, 2010.

[3] Bonnefoi P., Sauveron D., Park J.H.: *MANETs: An exclusive choice between use and security?* Computing and Informatics, vol. 27, 2008, pp. 799–821.

[4] Bubak M., Malawski M., Gubala T., Kasztelnik M., Nowakowski P., Harezlak D., Bartynski T., Kocot J., Ciepiela E., Funika W., Krol D., Balis B., Assel M., Ramos A.: *Virtual laboratory for collaborative applications.* [in:] Handbook of Research on Computational GridTechnologies for Life Sciences, Biomedicine and Healthcare, IGI Global, 2009, pp. 531–551.

[5] Coppola M., Jegou Y., Matthews B., Morin C., Prieto L.P., Sanchez O.D., Yang E., Yu H.: *Virtual organization support within a grid-wide operating system.* IEEE Internet Computing, vol. 12, 2008, pp. 20–28.

[6] Stenberg D. et al.: *cURL.* `http://curl.haxx.se/`, 2010.

[7] Dyrda M., Malawski M., Bubak M., Naqvi S.: *Providing security for MOCCA component environment.* Proc. of 23rd IEEE International Symposium on Parallel and Distributed Processing, Rome, Italy, 2009, pp. 1–7.

[8] Foster I. T., Kesselman C., Tsudik G., Tuecke S.: *A Security Architecture for Computational Grids.* ACM Conference on Computer and Communications Security, 1998, pp. 83–92.

[9] *GridSpace2 Platform.* `https://gs2.cyfronet.pl/`, 2010.

[10] *Internet 2 Project OpenSAML.* `https://spaces.internet2.edu/display/OpenSAML/Home/`, 2010.

[11] *Internet 2 Project Shibboleth.* `http://shibboleth.internet2.edu/`, 2010.

[12] Keahey K., Tsugawa M., Matsunaga A., Fortes J.: *Sky computing.* Internet Computing, IEEE, vol. 13, 2009, pp. 43–51.

[13] Malawski M., Bartynski T., Bubak M.: *Invocation of operations from script-based grid applications.* Future Generation Computer Systems, vol. 26, 2010, pp. 138–146.

[14] Meizner J., Malawski M., Ciepiela E., Kasztelnik M., Harezlak D., Nowakowski P., Król D., Gubała T., Funika W., Bubak M., Mikołajczyk T., Płaszczak P., Wilk K., Assel M.: *ViroLab Security and Virtual Organization Infrastructure.* Proc. of Advanced Parallel Processing Technologies 8th International Symposium, APPT 2009, Rapperswil, Switzerland, 2009.

[15] *OASIS Security Assertion Markup Language.* `http://saml.xml.org/saml-specifications`, 2010.

[16] *OpenBSD Project OpenSSH.* `http://www.openssh.com/`, 2010.

[17] OpenID Foundation, *OpenID Specifications.* `http://openid.net/specs/`, 2010.

[18] PADL Software Pty Ltd. *pam_ldap module.* `http://www.padl.com/OSS/pam_ldap.html`, 2010.

[19] Marín Pérez J.M., Bernal Bernabé J., Alcaraz Calero J.M., Garcia Clemente F.J., Martínez Pérez G., Gómez Skarmeta A.F.: *Semantic-based authorization architecture for grid.* Future Generation Computer Systems, in press, Accepted Manuscript, 2010.

[20] PL-Grid project. *PL-Grid web site.* `http://www.plgrid.pl/en`, 2010.

[21] Perez M., Xiao B.: *Special section: Security on grids and distributed systems.* Future Generation Computer Systems, vol. 23, 2007, pp. 774–775.

[22] RSA Security. *SecurID.* `http://www.rsa.com/node.aspx?id=1156`, 2010.

[23] Scavo T., Welch V.: *A Grid Authorization Model for Science Gateways.*, Proc. of International Workshop on Grid Computing Environments, 2007.

[24] Schwiegelshohn U., Badia R.M., Bubak M., Danelutto M., Dustdar S., Gagliardi F., Geiger A., Hluchy L., Kranzlmuller D., Laure E., Priol T., Reinefeld A., Resch M., Reuter A., Rienhoff O., Ruter T., Sloot P., Talia D., Ullmann K., Yahyapour R., von Voigt G.: *Perspectives on grid computing.* Future Generation Computer Systems, vol. 26, 2010, pp. 1104–1115.

[25] Spence D. et al. *ShibGrid: Shibboleth Access for the UK National Grid Service.* Proc. of the Second IEEE International Conference on e-Science and Grid Computing, Washington, DC, USA, 2006.

[26] University of Illinois. *GSI-Enabled OpenSSH.* `http://grid.ncsa.illinois.edu/ssh/`, 2010.

[27] University of Illinois. *MyProxy*, `http://grid.ncsa.illinois.edu/myproxy/`, 2010.

[28] Varrette1 S., Georget S., Montagnat J., Roch J.-L., Leprevost F.: *Distributed Authentication in GRID5000.* Proc. of OTM Confederated Internationl Workshops and Posters, AWeSOMe, CAMS, GADA, MIOS+INTEROP, ORM, PhDS, SeBGIS, SWWS, and WOSE, Agia Napa, Cyprus, 2005.

[29] ViroLab Project Consortium. *ViroLab* `http://virolab.org`, 2010.