Development of a Flexible PERMIS Authorisation Module for Shibboleth and Apache Server

Wensheng Xu, David Chadwick, Sassa Otenko

Computing Laboratory, University of Kent, Canterbury, England, CT2 7NZ
`{w.xu, d.w.chadwick, o.otenko}@kent.ac.uk`

Abstract. This paper describes the development of a flexible Role Based Access Control (RBAC) authorisation module – the Shibboleth and Apache Authorisation Module (SAAM) which is based on the PERMIS privilege management infrastructure. It explains how the module can work with the Apache web server, with or without Shibboleth. We argue that this can effectively improve the level of trust and flexibility of access control for the Shibboleth architecture and the Apache web server, as well as provide a finer grained level of control over web resources.

1 Introduction

Shibboleth [1] is a cross-institutional authentication and authorisation architecture for single sign on and access control over web resources. It is specified by the Internet2 middleware architecture committee and many universities in the USA and Europe have started to build experimental services based on it. Shibboleth can allow distributed users belonging to different institutions to share web resources conveniently and safely while respecting the users' privacy. What makes the Shibboleth architecture especially attractive is that authentication of a user is carried out by the home site (i.e. where the user originates from) whilst authorisation for a user to access specific web resources is carried out by the resource website. Such separation of authentication and authorisation functions eases the creation and management of federations of resource providers and users.

Shibboleth defines a protocol for carrying authentication information and user attributes from the user's home site to the resource site. The resource site can then use the user attributes to make the access control decision about the user's request. A user only needs to be authenticated once by the home site in order to visit other Shibboleth protected resource sites in the federation, as the resulting authentication token is recognised by any member of the federation. In addition to this, protection of the user's privacy can be achieved, since the user is able to restrict what information about him will be released to the resource providers from the user's home site.

Shibboleth's functionality is achieved by a simple trust relationship between the resource site and the user's home site. To put it simply, the resource site trusts the origin site to authenticate the user and to provide the correct set of attributes for the user, and the home site trusts the resource site to give access to users with the correct set of attributes. If a finer grained trust relationship is required to allow for distributed management of user attributes and dynamic delegation of authority, then a more sophisticated authorisation infrastructure than that provided by Shibboleth is required. For example, if the resource site trusts specific managers/authorities to allocate specific attributes to different groups of users, this cannot be conveyed via Shibboleth since there is a single attribute authority (AA) at each home site. Furthermore, the security of the source of the user attributes at the home site might be of concern to the resource site, both because of how the attributes are stored, and because of the user's dynamic pseudonymity[1]. Finally, the access control decision making based on these attributes is simplistic in its functionality, and the management of the access controls is mixed together with web server administration at the resource site. Therefore the flexibility of setting the access control policy is adversely affected.

These limitations in Shibboleth can be alleviated by integrating a policy controlled Privilege Management Infrastructure (PMI) into it. PMIs are described in the 2001 edition of X.509 [3]. PERMIS [2] is an implementation of an X.509 PMI, and uses the Role Based Access Control (RBAC) [10] paradigm. PERMIS is built in accordance with the ISO 10181-3 standard [15] and incorporates a sophisticated policy controlled application independent RBAC decision engine, or policy decision point (PDP), in its software suite. Roles are stored in X.509 attribute certificates (ACs), and since these are digitally signed for integrity protection, it can support the distributed management of roles between multiple AAs. Other experimental RBAC implementations have been developed, for example by Ferraiolo et al [12] and Sandhu et al [13, 14], but PERMIS is the first one to use X.509 ACs to store roles. PERMIS has already been successfully applied in several applications, and more recently has been integrated with the Globus Toolkit [4]. By developing and integrating a RBAC authorisation module into Shibboleth – the PERMIS SAAM (Shibboleth-Apache Authorisation Module) – a highly improved authorisation capability can be achieved for distributed web resource access control.

The rest of this paper is organised as follows. Section 2 describes Shibboleth, and lists its main features and limitations. Section 3 analyses how Apache [7] authentication and authorisation works, how Shibboleth interacts with this, and the approach that needs to be taken to replace Shibboleth authorisation by PERMIS authorisation. Section 4 presents the PERMIS SAAM system structure and the Apache directives that control it. Section 5 describes the interactions between the PERMIS SAAM and Shibboleth. Section 6 describes how the PERMIS SAAM can be integrated with the Apache web server without Shibboleth. Finally Section 7 gives the conclusions.

[1] Dynamic pseudonymity, provided by Shibboleth, allows the user to have a different pseudonym each time she contacts the resource site. Whilst this provides better user privacy (the user cannot be profiled by the resource site), it reduces the strength of the association between the user and her attributes. Furthermore, if multiple attribute authorities (AAs) issue attributes to the user, it will be difficult to facilitate that all of them dynamically re-issue the attributes each time the user's identity changes.

2 Main Features and Limitations of Shibboleth

As a middleware layer Shibboleth uses SAMLv1.1 [5] for encoding some of its messages. When a user contacts a Shibboleth-protected resource site (target site) with the browser, requesting access to a particular URL, the user is required by the Shibboleth Indexical Reference Establisher[2] (SHIRE) to go to a WAYF (Where Are You From) site to pick his/her home site (origin site), and their browser is redirected to their home site's authentication server where the user is invited to log in. After the user is authenticated by the origin site, the browser is redirected back to the target site along with a handle package which includes an assertion that this user has been successfully authenticated by a particular means (e.g. username/password, Kerberos or digital signature), a unique handle generated by the Handle Service for the user (the user's pseudonym), and the Attribute Authority (AA) location at the origin site where the user's attributes may be obtained from. Then the Shibboleth Attribute Requester (SHAR) at the resource site returns the handle to the AA of the origin site and gets a set of attributes of the user from the AA. The messages between the target site and origin site are encoded in SAML and are embedded as a browser cookie, so the user observes only redirections between the sites.

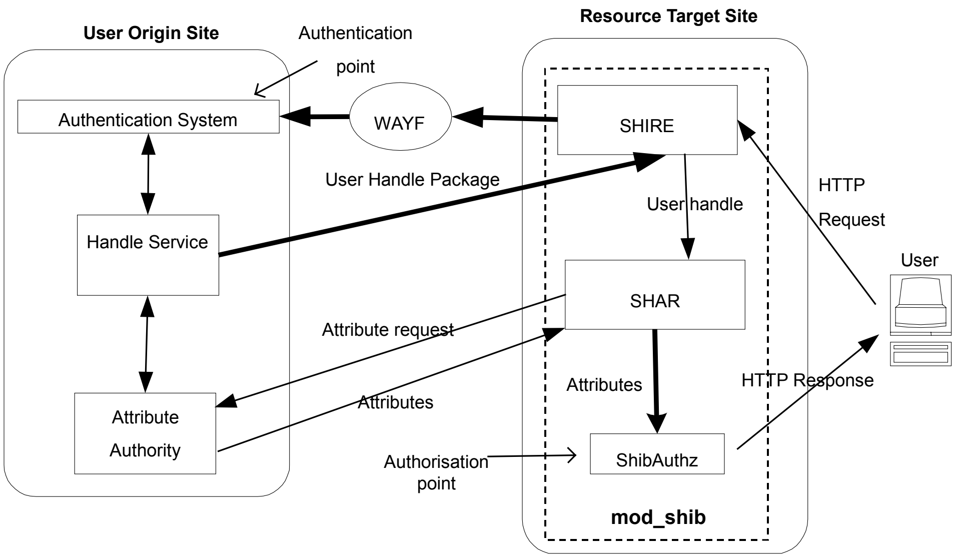

Fig. 1. The Shibboleth Authentication and Authorisation Process

The user attributes are then passed to the Shibboleth authorisation function - ShibAuthz, which will make an access control decision based on these attributes. The SHIRE, SHAR and ShibAuthz are all included in the Shibboleth Apache module

[2] In this article we refer to Shibboleth version 1.2 and its related documentation. At the time of writing Shibboleth architecture undergoes significant changes, whereby some of the components of the system will be regrouped and renamed.

called *mod_shib*[3]. The web server will then give a response back to the user browser based on the decision result. The whole Shibboleth authentication and authorisation process is shown in Fig. 1.

Because user authentication and authorisation are taking place at different locations, namely at the origin site and the target site respectively, Shibboleth allows for a different pseudonym (the handle) for the user's identity to be invented by the origin site every time. Both the origin site and the user can have control over the release of the user's attributes, so the user's privacy can be well protected. On the other hand, because authentication and authorisation are performed by different sites and the user's name is not provided to the target site for privacy reasons, the target site's access control is only based on the user's attributes without the need to know who issued them, whether they are still valid, or whether they are even the correct attributes for the particular user, so the safety of the target site heavily relies on trusting the origin site to return the correct attributes.

The messages carrying these attributes are digitally signed by the SAML authority at the origin site, so the security of these messages is ensured, but the security of the source of the attributes is not guaranteed. In many sites a back end LDAP [11] server is the authoritative source for both authentication and attribute information. These attributes in the LDAP server are not digitally signed, so it is relatively easy for these attributes to be tampered with compared to for example digitally signed X.509 attribute certificates (c.f tampering with passwords compared to tampering with X.509 public key certificates). Furthermore Shibboleth doesn't cater for multiple attribute authorities at the home site. There is only one AA that creates the cryptographically protected SAML tokens. The AA must query the attribute repository (e.g. LDAP server) to collect the user's attributes that are typically stored there as plain text. Even though the repository can be managed by multiple administrators, we would like to argue that this may not be secure enough, as it is difficult to ensure that an administrator does not exceed their authority. Because the security of the attributes in the repository is essential to the whole Shibboleth system, origin sites typically have a single administrator centrally managing the attributes of the users. This reduces the flexibility of the attribute assignments and inhibits the distributed/devolved management of them.

Another limitation of the Shibboleth infrastructure is that it provides only a basic access control decision making capability. The authorisation decision made by the target site is based on the attributes received from the origin site, and the access rules that are defined by Apache directives in the Apache configuration file. The directives can only express basic simple access control rules based on regular expressions, for example "users with attribute 'staff' can have access to location A" or "users with attribute 'senior member' can have access to location B", but it can not express conditional rules (e.g. access if time is between 9am and 5pm), complicated rules (e.g. ones with multiple conditions based on the parameters of the user's request) or RBAC features such as separation of duties or role hierarchies. This is acceptable for simple

[3] This is correct from the functional perspective. In the Shibboleth implementation however, the SHAR's actual function is implemented by an independent module which communicates with mod_shib by internal socket communications.

applications, but for advanced applications this is a weakness for resource site administrators.

Because the basic access control rules are defined in the Apache configuration file and the authorisation function is carried out by the Shibboleth Apache module, then every time the target site needs to change its access rules, it needs to redefine the directives in the Apache configuration file and restart the Apache server. This means that the administrator of the Apache web-server has to manage the access control rules, and there is no way for the owner of the resources to directly specify the access control rules without going via the Apache administrator. This limits the resource owner's flexibility for management of access control over his resources.

From the above discussion we can see that the Shibboleth target site makes access authorisation decisions after it receives user attributes from the Shibboleth origin site. We want to improve Shibboleth so that:

(1) If the attributes at the origin site are digitally signed by a relevant AA, then the trust between the origin and target sites no longer solely relies on the security of the origin attribute repository, e.g. LDAP server, as the attribute certificates (ACs) are tamper-proof themselves. Consequently the security level for the whole system can be effectively improved; for example, X.509 ACs can be adopted as user attributes and they can be stored in the origin site's LDAP repository and be released to the target site for access control decision making;

(2) When the target site receives the attributes or ACs from the origin site, at this point a sophisticated RBAC decision engine (e.g. PERMIS) can be used to make access control decisions instead of Shibboleth's own simple authorisation function. This will help to implement sophisticated access control features such as separation of duties and role hierarchies, so a finer grained and more refined access control mechanism can be deployed at the target site.

Once Shibboleth and PERMIS are integrated, both the security and flexibility of the Shibboleth infrastructure can be effectively improved. An in-depth discussion of the trust models and different approaches to the integration of Shibboleth and PERMIS can be seen in [9]. But for any of these models, two common problems need to be solved for integrating Shibboleth and PERMIS. Firstly, how can PERMIS replace Shibboleth's original authorisation functionality and make decisions based on attributes without the need to modify Shibboleth at the source code level. Secondly, how can ACs replace attributes and be stored and transferred by Shibboleth. The first question is addressed below and the second question is discussed in Section 5.

3 Analysis of Apache and Shibboleth authentication and authorisation functions

3.1 How Apache performs Authentication and Authorisation

Apache [7] is a popular open-source HTTP server that is widely used in universities and institutions to provide a web resource sharing service. How does the Apache server handle HTTP requests? The Apache server breaks down HTTP request handling into a series of processing phases, including:

– URI to Filename translation;
– Authentication identity check: to check who the user is;
– Authorisation access check: to check if the user is authorized here;
– Module-specific access checks: to check if there is any restriction from this module upon the requested resource;
– Sending a response back to the user;
– Other phases, unrelated to this article.

The basic functionality of the Apache web-server can be extended by adding so-called *modules* to it, and each module can handle one or several processing phases. (The authorisation phase at a Shibboleth target site is handled by one such Apache module - *mod_shib*.) When an Apache module is written, it must contain code to register specific handlers (functions) that are to be called for specific phases. Each of the above phases is processed sequentially by the Apache server and each registered handler (handling function) is called once for each phase, unless a preceding module completes the phase processing (see later). In Apache 1.3, the order in which the modules are called is fixed for all the phases, and depends upon the order in which the module is loaded. In Apache 2.0, a certain amount of flexibility has been introduced into the calling order, as each module can indicate its priority (FIRST, MIDDLE, LAST) for each phase. If two or more Apache modules have handling functions for the same processing phase, then these handling functions will be executed one after another according the order in which they were loaded. A simplified HTTP request handling process in the Apache server 1.3 is illustrated in Fig. 2.

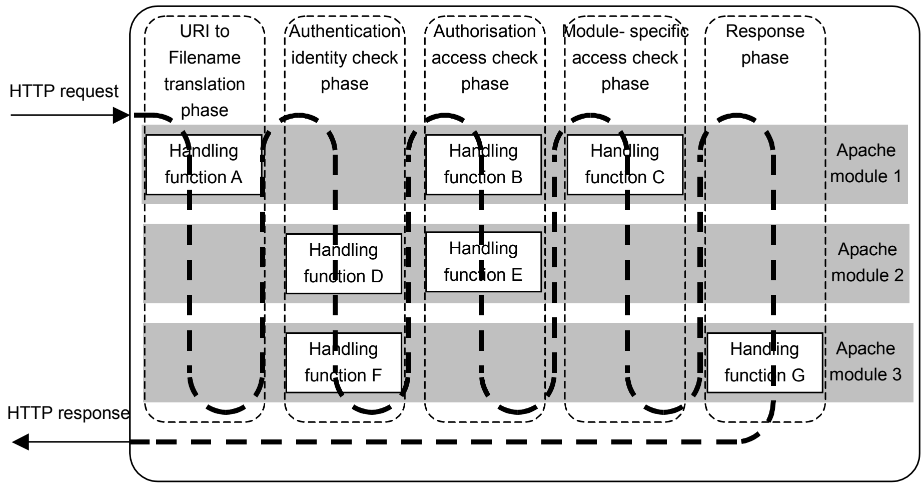

Fig. 2. HTTP Request Handling Process in the Apache Server 1.3

For the authentication and authorisation access check phases in the HTTP request handling process, the Apache server looks at a succession of Apache modules in sequence to match and invoke the corresponding handling functions. If a relevant module handling function is invoked by the Apache server for a phase, there may be three possible results:

– If the request is handled successfully, a magic integer constant OK will be returned to the Apache server and the subsequent Apache modules will not be invoked in this phase;
– If the module handling function finds an error in the user's request for whatever reason, one of the HTTP error codes will be returned, such as FORBIDDEN or HTTP_UNAUTHORIZED. This will also terminate the handling of the request, only this time the subsequent Apache modules will not be invoked for this phase or subsequent phases, and the user will be informed of the error by the browser according to the HTTP error code;
– If the module handling function declines to handle this phase, then the magic integer constant DECLINED will be returned to the Apache server. In this case the Apache server will continue to look at the rest of the modules in order to find a handling function to serve this phase. Usually DECLINED is returned by modules when the request is not applicable, like in cases when the requested location is not protected by the module (e.g. the AuthType directive is missing or specifies a type of authorisation that is not supported by this module).

The first two results above are "definite", i.e. there can be no other opinion about the request. The third result is "indefinite" and means that the handling function of a module cannot make a decision. So only when the preceding Apache module handling function returns the constant DECLINED, can the subsequent Apache modules be invoked for this phase, otherwise the rest of the Apache modules are skipped as if they didn't exist.

3.2 Shibboleth Integration with Apache

The Shibboleth Apache module is called *mod_shib,* and it provides the access control service and single sign-on capabilities. Mod_shib is invoked at the target site during two phases of the Apache HTTP request handling process: the Authentication phase and the Authorisation phase. The SHIRE and SHAR are invoked during the Authentication phase and ShibAuthz is invoked during the Authorisation phase. During the Authentication phase the SHIRE redirects the user's browser to the user's home site for authentication if it is the first time for the user to access a federated target site in this session. Both the Shibboleth origin and target sites are issued with X.509 public key certificates and these certificates are configured into mod_shib. After a user is authenticated at the origin site, a digitally signed handle package is sent back to the SHIRE at the target site. The SHIRE checks the signature on the User Handle Package to validate that the handle package is really coming from a trusted origin site. In this way, the Shibboleth target site trusts that the user has been reliably authenticated. The SHAR then collects attributes of the user from the AA at the origin site via the attribute query communication. On subsequent access requests in the same session, the SHIRE and SHAR simply check the user's cookies and retrieve the attributes from there. In the Authorisation phase ShibAuthz is invoked to make the access control decisions based on the attributes of the user.

Our aim is to allow mod_shib at the target site to perform normal Shibboleth authentication and attribute collection in the Authentication phase, but to override its authorisation mechanism in the Authorisation phase with our PERMIS RBAC policy-controlled PDP instead. So the design of the PERMIS authorisation module is straightforward. It should be invoked before the mod_shib authorisation code in the Authorisation phase, and obtain the attributes that Shibboleth has already retrieved from the AA in order to make a decision in accordance with the PERMIS authorisation policy.

4 System Structure of the PERMIS SAAM

Based on the above analysis, we developed the PERMIS SAAM authorisation module to work in conjunction with Shibboleth to provide a generic authorisation function based on RBAC and the PERMIS Privilege Management Infrastructure. The SAAM works as an Apache module and provides an authorisation handling function called during the Apache authorisation phase. By proper construction of the Apache configuration file, SAAM can be loaded and registered before the Shibboleth module, and can take the responsibility for making authorisation decisions thereby bypassing the Shibboleth authorisation function, without disturbing the rest of the functionalities of Shibboleth.

4.1 Functions of PERMIS

The PERMIS infrastructure comprises a privilege allocation (PA) component, a privilege verification (PV) component, a policy decision point (PDP) and a policy management GUI. The PERMIS PA component is responsible for allocating privileges to users in the shape of roles stored in X.509 attribute certificates (ACs). The PA component may be distributed and used by many managers to give roles to their subordinates. The role ACs are then stored in one or more LDAP directories for subsequent use by the PV component. After a user is authenticated, the PERMIS PV component can access these LDAP directories to retrieve the role ACs for the user (the *pull* mode of operation). Alternatively, the ACs can be given to the PV component by the caller for instant validation (the *push* mode of operation).

The PERMIS infrastructure is driven by a PERMIS policy that comprises a Role Allocation Policy (RAP) and a Target Access Policy (TAP) (see later). This may be created using the policy management GUI.

The role ACs are verified against the RAP by the PV component and all valid roles/attributes are passed to the PDP. The PDP then makes its access control decision for the user's request based on the TAP and the valid attributes. The PDP returns a granted or denied response to the caller according to the policy in force at that time.

In the integration of Shibboleth and PERMIS, authentication is carried out by the Shibboleth system. Shibboleth is responsible for providing PERMIS with the user name as it appears in the X.509 Attribute Certificates. Shibboleth may push the X.509 ACs into PERMIS, otherwise PERMIS may pull them from LDAP directories.

Note that Shibboleth has to provide the name of the user (as held in the X.509 ACs). Whilst this may decrease the user's privacy somewhat, it does not have to seriously undermine it, as the name used by the system does not have to be the user's real name. To maintain user privacy, which is a core consideration in Shibboleth, pseudonyms can be adopted as holder names in X.509 ACs, just like pseudonyms are adopted as user names in Shibboleth. The X.509 pseudonym can be a distinguished name string, or it can be the hash of the user's public key (although this requires the user to be PKI enabled, which many are not today). The main difference between the Shibboleth and X.509 pseudonyms is that the former ones are dynamic whilst the latter ones are static, which means that the target site can still build up a profile of the static pseudonymous user. If even this is too sensitive, then the PERMIS SAAM can adopt the simple Shibboleth trust model and transfer (unprotected) attributes attached to an anonymous handle, in which case X.509 ACs are not needed. In this scenario we would use the PERMIS PDP as a substitute for the original Shibboleth access control decision-making functionality, in order to benefit from its superior decision making functionality without sacrificing any of Shibboleth's privacy protection features, but conversely, we do not take advantage of the distributed role management functionality that X.509 ACs provide. Ultimately, the quality of user privacy can be determined by the origin site/application, but it is a trade off with the (loss of) trustworthiness and flexibility in the binding between a user and his/her attributes.

4.2 PERMIS RBAC policy

The PERMIS RBAC policy is the basis for access control of resources. It is written in XML and is kept in an X.509 Attribute Certificate, digitally signed by the Source of Authority (SoA), who is typically the resource owner. This serves the dual purpose of separating the policy specification from system administration of the Apache web-server, and makes the policy tamperproof. This policy AC is the root of trust for the access control decision making. A hierarchical RBAC model is adopted by PERMIS to specify the authorisation policy for the whole domain of resources controlled by one SoA. One PERMIS RBAC policy is able to control access to all resources in a domain by the same set of rules.

In the PERMIS RBAC policy there are two main sub-policies: the RAP and the TAP. The RAP is responsible for defining a list of trusted AAs, the set of attributes they are trusted to assign, and the groups of users they can be assigned to[4]. When the PERMIS PV component is passed a set of attribute certificates, it can retrieve the valid and trusted attributes from them according to the RAP and discard the invalid and untrusted attributes[5].

[4] This is where the user name is used by PERMIS.

[5] Note that since the RAP is defined at the Target site, the validity and trustworthiness of the user attributes is controlled by the resource owner.

The TAP is responsible for defining the set of targets that are protected by this policy, the associated actions that can be performed on them, the attributes that a user needs in order to be granted the actions, and the restraints/conditions that apply to granting access. After the PERMIS PDP gets the attributes of the user from the PV component, then it can make access decisions for the user based on the TAP.

Beside the RAP and TAP, the PERMIS RBAC policy also includes the following sub-policy components:

- The subject sub-policy specifies the subject domains, i.e. only users from these subject domains may be authorised to access resources covered by the policy;
- The role hierarchy sub-policy specifies the different roles and their hierarchical relationships to each other;
- The Source of Authority sub-policy specifies which SoAs are trusted to allocate roles, and permits the distributed management of role allocation to take place; these are, in effect, the multiple AAs at the Origin sites who are trusted by the Target;
- The target sub-policy specifies the target domains covered by this policy;
- The action sub-policy specifies the actions (or methods) supported by the targets, along with the parameters that should be passed along with each action, e.g. action *GET* with parameter *Filename*; in the Shibboleth-PERMIS integration scenario the actions should be the HTTP methods defined by RFC2616: *GET, PUT, POST, DELETE,* etc [8].

A full description of the PERMIS RBAC policy can be found in [6]. By adopting and enforcing the PERMIS RBAC policy, flexible fine grained access controls can be achieved.

4.3 Structure of the PERMIS SAAM

Based on the PERMIS infrastructure and the Shibboleth system architecture, the system structure of the PERMIS SAAM is shown in Fig. 3. All the components of SAAM are enclosed by dashed round-cornered rectangles and the rest of the components in the figure are Shibboleth. As in PERMIS, there are three sub systems in the PERMIS SAAM: the PERMIS PA sub system which is distributed to the various origin sites, and the PERMIS PV/PDP sub system and Policy Management sub system which are entirely located within the target site. The PV/PDP sub system is responsible for validating the ACs and making access control decisions, while the PA sub system at the origin site is responsible for assigning privileges to users. The Policy Management sub system at the target site is responsible for defining the RBAC policy and digitally signing it and storing it to the policy LDAP repository (denoted as “Policy LDAP” in Fig.3). If PERMIS is working in pull mode, the PERMIS PV fetches the user attribute certificates directly. Note that this requires the PV at the target site to be able to access the LDAP directories at the AC storage sites directly. Multiple AC LDAP directories are supported by SAAM in pull mode. If PERMIS is working in push mode, then Shibboleth is responsible for fetching the user attributes or ACs from the origin site and passing them to the PERMIS PV at the target site.

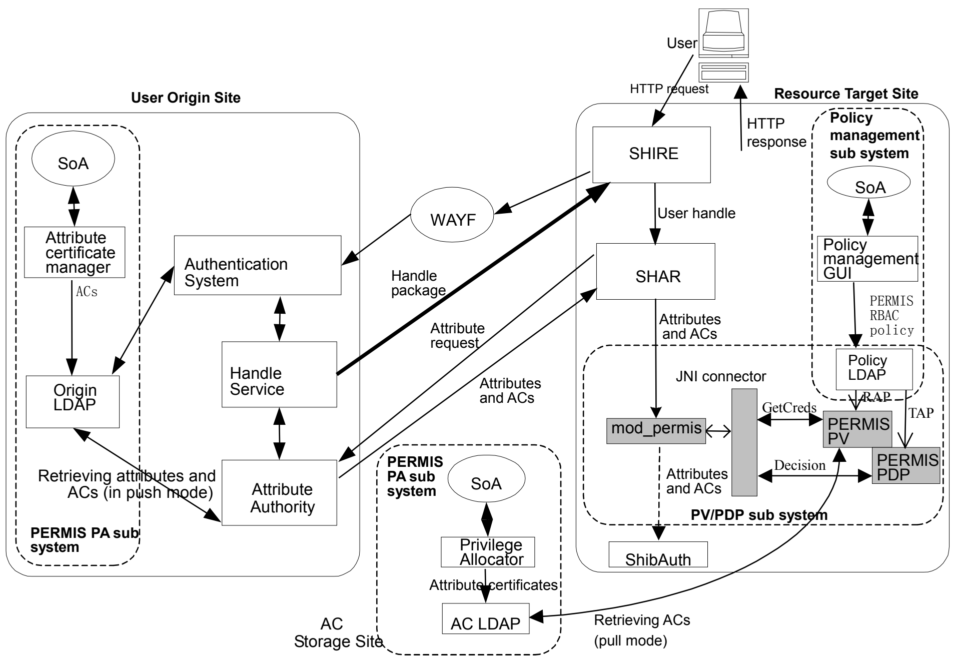

Fig. 3. Structure of Shibboleth-PERMIS SAAM Integration

The PERMIS PV/PDP sub system consists of four parts: an Apache module - mod_permis which is written in C++, the PERMIS PV and PDP which are written in Java, and a Java JNI (Java Native Interface) connector which is written in C. The PERMIS PV and PDP can be called by mod_permis via the JNI connector. Mod_permis interfaces with Apache and Shibboleth to collect all the information necessary for making a decision and passing this information to the PERMIS PV and PDP. The PERMIS PDP which is based on RBAC makes a decision and passes it back to mod_permis, which translates it into "OK" or "HTTP_UNAUTHORIZED" error codes. Apache will either send the requested resource or an error information page back to the user browser depending on the result. Note that since PERMIS returns a "definite" result when the PERMIS SAAM is active, Shibboleth authorisation is not invoked. To ensure that PERMIS is called before Shibboleth authorisation, mod_permis should appear before the Shibboleth Apache module (mod_shib) in the Apache 2.0[6] configuration file. (Since Apache 1.3 loads its modules in reverse order, mod_permis should appear *after* mod_shib in Apache 1.3.) Each location[7] in the Apache configuration file may use a different form of authorisation. The PERMIS SAAM is active only if the PermisAuthorisaton directive is present for the location (see below). If it is not present, mod_permis always returns

[6] Because mod_shib is already set as FIRST in Apache 2.0, we have no way to give mod_permis a higher precedence other than the order in which it is loaded.

[7] As indicated by the Apache <location> directive.

"DECLINED", so that Shibboleth or any other configured authorisation module will be invoked in this case.

In the implementation of the integration of Shibboleth and the PERMIS SAAM, several global configuration directives are needed in the Apache configuration file (see below). Two local directives are also used for each protected location to indicate that the PERMIS SAAM is being used for this target resource, thereby bypassing the authorisation function of Shibboleth.

4.4 The PERMIS SAAM Apache Directives

The PERMIS SAAM is configured using the following directives in the Apache http.conf file:

PermisPolicyIdentifier – this holds the unique number for the PERMIS policy to be used

PermisPolicyIssuer – this holds the LDAP distinguished name of the SoA who signed the policy

PermisPolicyLocation – this contains the URL of the LDAP directory holding the policy

PermisAuthorisation – this is inserted into every <Location> that is to be controlled using PERMIS authorisation. (Note, in the absence of the PermisPullMode directive the AuthType for this <Location> must be set to Shibboleth.)

PermisPullMode (optional) – this is inserted into every <Location> that is to pull ACs from LDAP repositories pointed to by the PermisACLocation directives. When this directive is not present, the default mode of operation is the push mode (Shibboleth gets the Attribute Certificates from the Origin, then mod_permis pushes them to the PDP).

PermisACLocation (optional) – this contains a URL of an LDAP directory from where user ACs may be pulled (this directive may be repeated as often as required)

5 Interactions between Shibboleth and the PERMIS SAAM

According to the different trust models adopted by the Shibboleth target and origin sites [9], the PERMIS SAAM can work in different modes with X.509 ACs - either push mode or pull mode. Furthermore, either plain attributes or X.509 ACs can be pushed to the PERMIS SAAM by Shibboleth. These are described in the following subsections.

5.1 PERMIS SAAM in push mode with X.509 ACs

If the origin site wishes to distribute attribute assignments to different managers, and perhaps implement dynamic delegation of authority, and the target site is willing to trust different attribute authorities at the origin site, then the origin site should store digitally signed attribute certificates in its LDAP repository (denoted as "origin

LDAP” in Fig. 3). Alternatively, if either the target and/or the origin do not trust the origin site’s attribute repository to securely store unsigned attributes, then the origin should assign ACs to users and store these ACs in its LDAP repository. In these cases, SAAM should work in push mode and accept ACs from Shibboleth.

In this mode of operation, one user attribute (the user’s distinguished name) and all user ACs (*attributeCertificateAttribute;binary)* should be configured for release to the target by the origin AA server. The user’s DN and the role ACs should be retrieved by Shibboleth and passed to the PERMIS PV/PDP for validation and making access control decisions for the user’s requests. For the PERMIS PV to validate that these are the correct ACs, the user’s DN should be available to match with the holder name in the ACs. The PV uses the RAP in the PERMIS policy to decide who is trusted to assign which attributes to whom. As discussed in [9], supplying ACs and user DNs in the integration of Shibboleth and PERMIS doesn’t necessarily compromise a user’s privacy since pseudonyms can be used as the DN and the AC holder name. On the other hand, some applications actually require the user’s DN to be present in order to perform correct access controls, and so passing the user’s DN in these cases is actually beneficial.

The interactions between Shibboleth and the PERMIS SAAM are as follows.

(1) When a user contacts a Shibboleth-protected resource site with the browser, requesting access to a Shibboleth-PERMIS protected URL, the user is redirected by the SHIRE to the WAYF site.

(2) After the user selects his/her home (origin) site at the WAYF site, the browser is redirected to the origin site’s authentication server and the user is authenticated there.

(3) After successful authentication, the browser is redirected back to the SHIRE along with a handle package.

(4) The SHAR at the target site gets the handle and sends the handle to the AA of the origin site for attributes query.

(5) The AA retrieves the user’s DN and the role ACs of the user from the origin LDAP directory, base-64 encodes the attribute certificates, and sends them back to the SHAR.

(6) The SHAR puts the attributes in the Apache HTTP headers whose names can be defined and configured in the Shibboleth Attribute Acceptance Policy (AAP). This is the last step of the authentication phase.

(7) In the authorisation phase in the HTTP request handling process, mod_permis is first invoked by the Apache server.

(8) If the location being requested by the user is not being protected by PERMIS, then mod_permis returns DECLINED and the Shibboleth authorisation function ShibAuthz will subsequently be invoked, otherwise the user’s DN and role ACs are acquired by mod_permis from the HTTP headers.

(9) Mod_permis calls the PERMIS PV and PDP to make an authorisation decision, which is based on the user’s DN, the role ACs, the target resource that the user is requiring, the action to the target resource (i.e. the HTTP method) and the current RBAC policy incorporating both the RAP and TAP.

(10) After the PERMIS PDP makes the granted/denied decision, the decision is returned back to mod_permis;

(11) Mod_permis returns the decision result to the Apache server, and the user can be granted or denied access to the target resource according to the decision result.

From the above interactions between Shibboleth and the PERMIS SAAM we can see that in this mode the only difference between normal Shibboleth and this integrated Shibboleth is that ACs are retrieved and passed by Shibboleth instead of plain text attributes. Since ACs are stored in the LDAP as digitally signed binary attributes and normal Shibboleth cannot retrieve binary attributes[8], Shibboleth needed to be slightly modified to handle them. On the origin side one Java class for retrieving attributes from LDAP - JNDIDirectoryDataConnector.class - was modified by us and another new Java class - Base64ValueHandler.class - was developed by the Shibboleth developers. The latter encodes the ACs into Base64 plain text (in Step 5 above). Now the encoded ACs can be transferred as plain text attributes from the origin to the target site, where they are decoded into normal binary ACs before being passed to the PERMIS PV/PDP (in Step 9 above), for use by the RAP and TAP in decision making.

If a user possesses multiple roles, then multiple ACs can be assigned to the user and stored in the LDAP directory at the origin site. Shibboleth will retrieve all the role ACs at the origin site, encode them and then join them with semicolons, before passing them to the target site as a multi-valued attribute (in Step 5 above). After mod_shib puts the combined text encoded AC into the HTTP header (in Step 6 above), mod_permis will retrieve it and restore it into separate encoded ACs which can then be passed to the PERMIS PV/PDP for access control decision making (in Step 7 above). In this way multiple ACs can be handled and utilised in access control decision making in Shibboleth.

5.2 PERMIS SAAM in push mode with plain attributes

If the target site trusts the origin's attribute repository and the origin as a single AA, then the origin will store plain attributes in its repository, and pass them in digitally signed SAML messages to the target. This is the standard Shibboleth mode of operation. In this mode, the interactions between Shibboleth and the PERMIS SAAM are nearly the same as in Section 5.1 except that it is the user's attributes, not the user's DN and role ACs, that are passed by Shibboleth and used by the PERMIS PV/PDP to make decisions. In this case SAAM works in push mode, by pushing the attributes which were retrieved by Shibboleth, to the PERMIS PV/PDP.

The Shibboleth origin site can be configured to append a scope domain to each released attribute. Scope domains are used to distinguish between different attribute issuers at the origin site, for example some attributes could have a scope domain of "salford.ac.uk", while others could have a scope domain of "computing.salford.ac.uk" (note that the same attribute cannot have multiple scope domains, which effectively precludes dynamic delegation of authority). When the PERMIS PV is being passed "scoped" attributes instead of digitally signed ACs, the scope domains take the place of the AC signers (i.e. the SoAs). In order to validate

[8] This is true as of version 1.2 of Shibboleth. However the Shibboleth developers have said that binary attributes will be supported in a future release.

“scoped” attributes, the PERMIS RAP should specify the scope domains as SoAs in place of AC issuer DNs. We have reserved a special URL “shib:<scope domain name>” for this. In the above example there would be two corresponding SoAs identified in the RAP by the special URLs: “shib:salford.ac.uk”, and “shib:computing.salford.ac.uk”[9]. If scope domains are not being used by an origin site, then SAAM inserts the name of the origin site as the scope domain for all the attributes. The scoped attributes can now be validated against the RAP by the PERMIS PV in the same way as X.509 AC issuers, except that cryptographic validation cannot be performed. Thus there is no proof who actually issued the attributes as “scoped” attributes don’t have digital signatures.

The other difference from the scenario in Section 5.1 is that the user’s LDAP DN is not provided (since they have a pseudonym dynamically generated by the origin site). Therefore the PERMIS Subject Domain sub-policy should include the null DN (meaning any DN is allowed) and the RAP should refer to this subject domain when specifying whom the attributes can be assigned to. Otherwise the attributes of the pseudonymous users (users with a null DN) will not be valid and all access will be denied to them.

5.3 PERMIS SAAM in pull mode

If the target trusts different attribute authorities based at the origin site and elsewhere, and wishes to authorise users based on these, then the origin site may not always be able to push all the attributes to the target site. In this case the PERMIS SAAM should work in pull mode to fetch the ACs itself. An example might be: a graduate is issued with a degree certificate by a university, a doctor is issued with a “clinician” certificate by the General Medical Council, and an engineer is issued with a “certified MS engineer” by a Microsoft accredited agency. In this case various distributed LDAP repositories (denoted as “AC LDAP” in Fig. 3) may sit in various places other than the origin site, and should be accessible by the PERMIS PV. The PERMIS PV can operate in pull mode and fetch all the needed ACs from the LDAP repositories. In this working mode, only one attribute of the user - the user’s DN, should be configured and stored in the origin LDAP repository. The user’s DN denotes the holder identity of the ACs in the various LDAP repositories and this DN will be retrieved and passed by Shibboleth to the PERMIS PV, so that the PV can know which ACs to retrieve from the various LDAP repositories. Once the ACs have been retrieved by the PERMIS PV, the PV will use the RAP to determine which ACs are trusted, and the PDP will use the TAP to determine if the user has the necessary attributes to access the resource.

The interactions between Shibboleth and the PERMIS SAAM are as follows.

(1) A user contacts a Shibboleth-protected resource site with the browser, is redirected by the SHIRE to the WAYF site, is authenticated at the origin site; then the browser is redirected back to the target site along with a handle package. The SHAR

[9] The PERMIS policy syntax has been extended to allow URLs as SoA identifiers instead of LDAP DNs. Thus ‘<SOA ID=”Salford” URL=”shib:salford.ac.uk”/>’ defines an SoA that is identified by the Shibboleth scope domain “salford.ac.uk” in the plain-text attributes.

at the target site gets the handle and sends the handle to the AA of the origin site with an attributes query. (the same as Step 1 to Step 4 in Section 5.1)

(2) The AA retrieves the user's DN from the origin LDAP repository and sends this back to the SHAR.

(3) The SHAR passes the user's DN to the Apache HTTP header.

(4) In the authorisation phase in the HTTP request handling process, mod_permis is first invoked by the Apache server, and the user's DN is acquired by mod_permis through the HTTP header.

(5) Mod_permis calls the PV and passes the user's DN to the PV. The PV retrieves the user's ACs from the various LDAP repositories according to the user's DN, then validates them against the RAP. Finally the PDP makes an authorisation decision based on the user's validated attributes, the target resource, the action being requested and the current RBAC policy.

(6) After the PDP makes the decision, the decision is returned back to mod_permis.

(7) Mod_permis returns the decision result to the Apache server, then the user can be granted or denied access to the target resource according to the decision result.

6 PERMIS SAAM with Apache and without Shibboleth

Since the PERMIS SAAM can work in pull mode and the PV is able to directly fetch ACs from LDAP repositories elsewhere, we can integrate the PERMIS SAAM with other Apache authentication systems without requiring Shibboleth to provide authentication or the user's attributes. This will provide us with a PERMIS authorisation service for web based resources, provided the user's DN can be passed from the authentication system to SAAM. The PV can then use the user's DN to fetch the user's ACs from the various LDAP repositories and make access control decisions based upon them.

6.1 System structure

In this section we describe how the PERMIS SAAM is configured to work with the Apache server where the Apache module mod_auth_ldap is used as the authentication module. The system structure is shown in Fig. 4. There is only one major difference between Fig. 4 and Fig. 3: in Fig. 3, the authentication service is performed by the Shibboleth system which is distributed between two computer systems (the origin and the target), while in Fig.4 the authentication service is performed by mod_auth_ldap which is an Apache module located in the same (target) computer system as the PERMIS SAAM. Note that the LDAP AC repository in Fig.4 doesn't necessarily need to sit in the same computer system as the PERMIS SAAM - it may sit somewhere else as it does in Fig. 3.

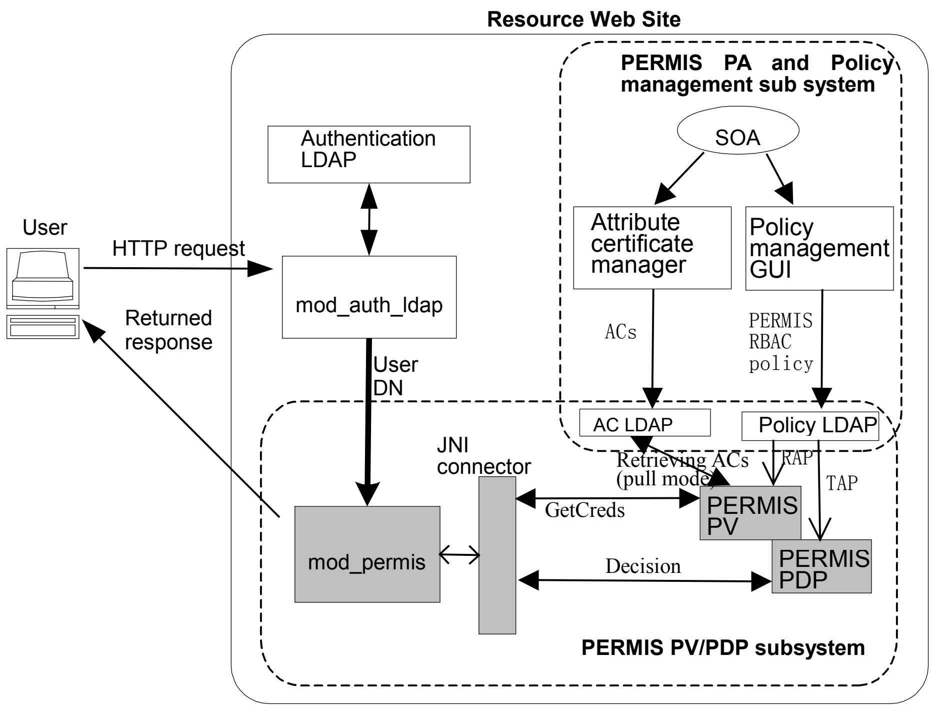

Fig. 4. Structure of Apache-PERMIS SAAM Integration

6.2 Interactions between SAAM, the authentication module and the Apache server

The interactions between SAAM, the authentication module (mod_auth_ldap) and the Apache server are as follows.

(1) When a user contacts an Apache web server, requesting access to a URL which is protected by mod_auth_ldap and mod_permis, the user is prompted by mod_auth_ldap to enter their username and password in order to be authenticated.

(2) Mod_auth_ldap authenticates the user by searching in the authentication LDAP server and locating the correct entry which matches the username and password, then retrieves and puts the user's DN in the Apache HTTP header.

(3) During the authorisation phase in the HTTP request handling process, mod_permis is invoked by the Apache server, and the user's DN is acquired by mod_permis through the HTTP header.

(4) Mod_permis calls the PV and passes the user's DN to it, the PV retrieves the user's ACs from the configured LDAP servers and validates them. The PDP then makes an authorisation decision based on the valid attributes and this is returned back to mod_permis.

(5) Mod_permis returns the decision result to the Apache server, and the user is granted or denied access to the target resource according to the decision result.

In our implementation, the Apache module mod_auth_ldap has been slightly modified so as to output the user's DN to the HTTP header during authentication[10].

7 Conclusions

The PERMIS SAAM module has been successfully developed and integrated with Shibboleth to replace the authorisation function in Shibboleth without modifying the Shibboleth source code. The flexibility, functionality and granularity of Shibboleth's authorisation decision making capabilities have been improved by adding PERMIS's policy controlled hierarchical RBAC implementation. When additional RBAC functionality, such as dynamic delegation of authority and separation of duties are added to future PERMIS releases, these will be automatically inherited by Shibboleth. Although the PERMIS SAAM was originally targeted at Shibboleth and was integrated with Shibboleth and the Apache server, an unexpected benefit is that it can work perfectly well with other Apache authentication modules without requiring Shibboleth to be present. By deploying the SAAM module, flexible, distributed, fine grained and more functional access control can be achieved by Apache web sites as well.

8 Acknowledgements

The authors would like to thank UK JISC for funding this work under the SIPS project.

References

1. S. Cantor. Shibboleth Architecture, Protocols and Profiles, Working Draft 02. 22 September 2004, see http://shibboleth.internet2.edu/
2. D. W. Chadwick, A. Otenko, E. Ball. Role-based access control with X.509 attribute certificates. *IEEE Internet Computing*, March-April 2003, pp.62-69.
3. ISO 9594-8/ITU-T Rec. X.509 (2001). *The Directory: Public-key and attribute certificate frameworks*
4. D. W. Chadwick, A. Otenko, V. Welch. Using SAML to link the GLOBUS toolkit to the PERMIS authorisation infrastructure. In *Proceedings of Eighth Annual IFIP TC-6 TC-11 Conference on Communications and Multimedia Security*, Windermere, UK, September 15-18, 2004, pp.251-261.
5. OASIS. *Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1*, 2 September 2003.

[10] Note that if other authentication modules are to be used with PERMIS, they also need to be modified to publish the authenticated DN in the HTTP headers of the request.

6. D.W.Chadwick, A. Otenko. RBAC Policies in XML for X.509 Based Privilege Management. In M. A. Ghonaimy, M. T. El-Hadidi, H.K. Aslan, editors, *Security in the Information Society: Visions and Perspectives: IFIP TC11 17th Int. Conf. On Information Security (SEC2002)*, May 7-9, 2002, Cairo, Egypt. Kluwer Academic Publishers, pp.39-53.
7. The Apache Software Foundation. http://httpd.apache.org/
8. http://www.w3.org/Protocols/rfc2616/rfc2616.html
9. D. W. Chadwick, A. Otenko, W. Xu. Adding Distributed Trust Management to Shibboleth. In *Proceedings of 4th Annual PKI R&D Workshop: Multiple Paths to Trust*, NIST, Gaithersburg, MD, April 19-21, 2005.
10. R. Sandhu, D. Ferraiolo, R. Kuhn. The NIST Model for Role Based Access Control: Towards a Unified Standard. In *Proceedings of 5th ACM Workshop on Role-Based Access Control*, Berlin, Germany, July 2000, pp.47-63.
11. M. Wahl, T. Howes, S. Kille. *Lightweight Directory Access Protocol (v3), RFC 2251*, Dec. 1997.
12. D. Ferraiolo, J. Barkley, and R. Kuhn. A role-based access control model and reference implementation within a corporate internet. *ACM Transactions on Information and System Security*, vol.2, no.1, February 1999, pp.34-64.
13. S. P. Joon, R. Sandhu, and G. Ahn. Role-based access control on the web. *ACM Transactions on Information and System Security*, vol.4, no.1, February 2001, pp.37-71.
14. J. S. Park, R. Sandhu. RBAC on the Web by smart certificates. In *Proceedings of 4th ACM workshop on role-based access control* (RBAC '99, Fairfax, VA, Oct. 28-29, 1999). ACM, New York, NY.
15. ITU-T Rec X.812 (1995) | ISO/IEC 10181-3:1996. *Security Frameworks for open systems: Access control framework*.