Security Infrastructure for A Web Service Based Resource Management System

Yong Yan, Michael Goss, Raj Kumar
Mobile and Media Systems Laboratory
HP Laboratories Palo Alto
HPL-2002-297
October 15th , 2002*

symmetric key cryptography, authentication, access control, integrity checking, Kerberos, global resource management, grid computing

A global resource management system intends to aggregate all kinds of heterogeneous resources that are geographically distributed so that a uniform resource programming interface can be provided to applications. The emerging web service model with single SOAP-based RPC interface provides a good way to uniformly abstract underlying resources and hide the heterogeneity of resources. In a web service mode based resource management system, the security infrastructure is a paramount component.

In this paper, we have designed and implemented a security infrastructure for a Web service model based global resource management system. In order to provide a secure shell around the global resource management system while keeping a simple Web access interface to the system, a Kerberos-like authentication system is built from the modification to the Kerberos model with a new set of authentication protocols while keeping the basic security mechanisms of the Kerberos. Our security infrastructure enforces Kerberos-like strong authentication and role-based access control. By encapsulating principals-to roles assignment into the service ticket, our system first seamlessly combines the Kerberos-like authentication with the role-based access control model. With the help of two security mechanisms: passport and service guard, transparent enforcement in the Web service messaging engine and the separation of security policy and security engine have been achieved so that our system can be easily extended and enhanced.

1 INTRODUCTION

Internet provides a pervasive interconnect infrastructure for aggregating all kinds of heterogeneous distributed computing resources together to support a wide range of information sharing, collaboration, and distributed computing. This makes it possible to build global information infrastructure to realize virtual organizations for the global sharing and management of resources and services as visioned by the Grid Computing [13], the Eos[30], and the AMSES[31].

In the global information infrastructure vision, a global intelligent resource management system is entailed to seamlessly harvest distributed heterogeneous resources to meet various kinds of dynamic resource demands of distributed computing or services. In order to provide a uniform resource programming interface over heterogeneous resources, a global resource management system usually virtualizes heterogeneous resources to a common resource representation. The emerging web service standard [16][29] provides a general and portable software architecture for universal application integration and messaging based on the XML-based SOAP RPC messaging mechanism and the WSDL-based extensible service representation. The generality of the web service architecture also provides a standard way to abstract heterogeneous resources. This motivates us to design and develop a global resource management system, named ARMS (Automatic Resouce Management Systems) (the old name is EWS), based on the web service standard.

In the ARMS, all resources and management functions are represented as web services so that the system can be easily deployed, extended, and managed. For the practical use of this system, an authentication and access control infrastructure is required to authenticate each request againist all kinds of potential security threats and control accesses to services. The authentication should accomplish two major functions: (a) determining if a requester is who it claims to be againist all kinds of threats, and (b) determining if a request comes from an authenticated requester without changing. The access control should only allow authorized service accesses.

The design of the ARMS aims at providing a global secure shell while keeping client simple and stateless. It is required that the authentication and access control infrastructure can be seamlessly integrated with our web service based resource management system without affecting service API so that the authentication and access control infrastructure can be independently extended and upgraded.

Authentication methods fall into two categories based on their key cryptography techniques: the symmetric key cryptography based authentication, such as the Kerberos authentication[5][20][25][27], and the asymmetric (or public) key cryptography based authentication, such as the certification based authentication[6][9][19]. The major differences between these two categories of authentication methods as summaried by B. Tung [27] are (a) the asymmetric key cryptography takes orders of magnitude longer to encrypt and decrpt the same amount of information than it does with symmetric key cryptography; (b) comparing to the primes based asymmetric key ciphers, such as RSA, symmetric key ciphers are much stronger; (c) the asymmetric key cryptography is more scalable than the symmetric key cryptography in key distribution; (d) the asymmetric key cryptography needs some way to do key certification while symmetric key cryptography needs not; (e) in the asymmetric key cryptography based authentication methods, a large number of certifications usually needs to be maintained by each services. These motivate us to build our authentication infrastructure based on the symmetric key cryptography.

The Kerberos authentication method is a widely deployed symmetric key based authentication method. In order to improve the scalability, the Kerberos method is extended with a hierachical KDC structure and is further enhanced by combining with certificate based authentication methods [6][27]. However, the three-way authentication protocol of the Kerberos treats the authentication system as an external security component to a protected system. This not only complicates a client component, but also prevents the use of a web based user interface that can only talk to the connected server.

In the design of our system, we modify the Kerberos authentication protocol so that (a) the authentication system can be integrated into the resource management sysem, instead of working as an external component, for better management capability; (b) the client has single access point to the resource management system which allows applet-based client interface and web-based client interface; and (c) the three-way authentication phase of the Kerberos can be improved into a two-way authentication phase without compromising security. Except the modification on the Kerberos authentication protocol, our Kerberos-like authentication method keeps compatible with the Kerberos v5 [20] in all other aspects: private key, session ticket, authenticator, and cryptographic algorithms.

For a global scale access control infrastructure, the role-based access control model has been shown to be successful [27]. This paper has designed and implemented a role-based access control model in an integrated way with the authentication system. By encapsulating principal roles into the service ticket, the principal roles can be securely transferred to the service without extra encryption and decryption overhead.

What is the suitable secure model for the web-service model used in our resource management system is still an open question right now. In this apper, we introduce two secure mechanisms: passport and service guard to transparently integrate the authentication and access control functions into the web service model. Passport and service guard are two security plugins in the web service's messaging engine, cooperatively enforcing principal authentication, message integrity checking, and service access control. The two mechanisms separate enforcement and policy management. Furthermore, the passport and service guard mechanisms structrure workstation into protected resource domains and enforces protection domains through access control checks.

The authentication infrastructure implemented in our resource management system can also be further extended to interact with other existing authentication domains using the similar approaches as described in [3][27]. This paper focuses on addressing our authentication infrastructure for single security domain.

This paper is organized as follows: we compare our work with those related work in Section 2. We describe the resource management system, the Kerberos, and the security requirements in Section 3. Section 4 presents the design of the seucirty infrastructure. The implementation is decribed in Section 5. Then, we will conclude the paper in Section 6.

2 RELATED WORK

It has been known that the basic authentication [10] and the digest authentication [12] in the HTTP server are not suitable for use at large [15]. A stronger authentication infrastructure is demanded.

The Kerberos authentication system [5][20][25][27] is a symmetric key cryptography based third-party authentication service with the assumption that its clients believe Kerberos's judgement as to the identity of each of its other clients to be accurate. It consists of four distributed components: user/application client (UAC), administration client (AC), authentication server (AS), and ticket granting server (TGS). When a client wants to access a service, it first needs to get a session key and a ticket to the TGS by sending its login name and TGS service name to the AS. The AS returns an encrypted message containing a TGS session key and a TGS ticket, which is locally decrypted by using the client password. Then, the client contacts the TGS to get a ticket to the requested service by sending the service name, the TGS ticket, and an authenticator that is encrypted by using the TGS session key. The client will receive a service session key and a service ticket, which allows the client to authenticate itself to the service by turning in the service ticket and an authenticator encrypted by the service session key. The Kerberos system protects credential replay by limiting the life-time of a service ticket and timestamping an authenticator while allowing the reuse of a ticket in its lifetime.

The Kerberos authentication system has been publicly considered to be highly secure. However, its three way authentication procedure complicates the client interface and separates the authentication function from the service. This is not well fit into web based applications. Our Kerberos-like authentication infrastructure integrates the authentication function into our global resource management system while still keeping a simple client interaction that can be easily built into different kinds of client interfaces, such as the web browser, the Java applet-based interface, and independent client applications.

Apach [1] also has a Kerberos module [26] which enables the Web server to act on a client's behalf. The Web server has complete knowledge about a client's service session key. This delegation approach gives the Web server an unlimited power to impersonate users, significantly compromising Kerberos's security strength. In our authentication method, the cipher text returned from the KDC to a client is encrypted first by the client's private key, and then by the resource management server's private key. When the resource management server decrypts a received cipher message, the server knows which client the message will go to, but not the content.

Version 3.0 of the Secure Sockets Layer protocol [11] (or TLS[8]) provides public key based services for mutual authentication, and key exchange for privacy. SSL 3.0 allows a client and server who are each in possession of a public key certificate signed by a trusted CA to mutually authenticate and establish a shared symmetric session key. Moreover, SSL specifications cover the use of multiple public key algorithms (DSA, RSA) and multiple session key algorithms (DES, RC4) as determined by the parties. The certification based authentication method needs a client to maintain its certifications which doesn't satisfy our requirement for a stateless client. PKINIT uses a digital certificate in the initial Kerberos authentication. In our Kerberos-like authentication infrastructure, we use certificate based SSL for the client and service registration.

In order to integrate SSL based web applications into a Kerberos based back-end system, A Kerberized credential translation service is proposed to bridge SSL and Kerberos in [22]. This service is also useful to our authentication infrastructure when we consider the interaction with existing authentication systems.

Regarding the role-based access control model, a lot of work have been done as summarized by J. S. Park, etc. [27]. This paper addresses how to integrate the role-based access control model with our Kerberos-like authentcation infrastructure in the Web service model.

Web service is a new emerging technolgy to provide a loosely-coupled, language-neutral, and platform-independent approach for software integration and interaction. The security model of the Web service is being developed by a joint effort between IBM and Microsoft [24]. WS-security [2] describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. In this paper, we introduce two service level security mechanisms: passport and service guard, to fullfil the authentication and integrity checking for our web service based global resource management system. Passport and service guard are two general security mechanisms that can be used with various kind of authentication methods and low-level messaging mechanisms such as that proposed by WS-Security [2].

3 Overview of System Background

In this section, we first describe ARMS: our global resource management system, and its security requirements. Then, we describe the authentication protocol, ticket, and authenticator of the Kerberos method, by modifying which our authentication protocol is built in next section.

3.1 ARMS: Automatic Resource Management System

ARMS is an on-going research and development system in HP Labs with aims to automatically manage all kinds of heterogeneous resources. The current basic working system is able to manage a pool of heterogeneous workstations.

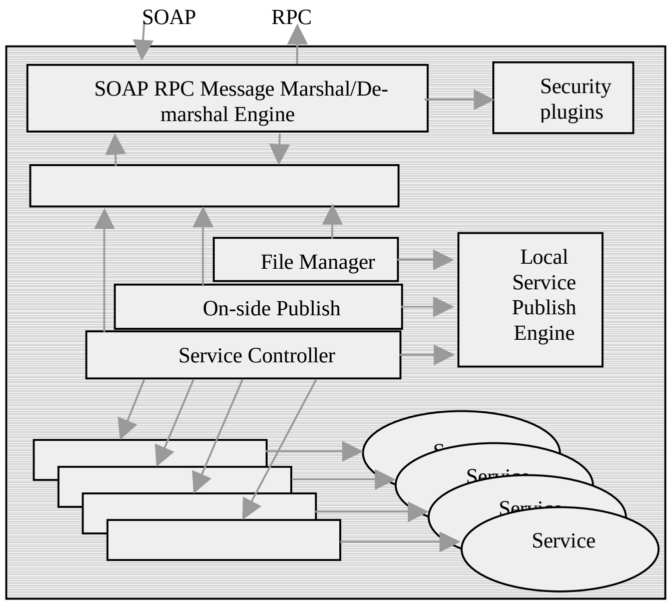


Figure 1. Component Architecture of the Service Agent.

Different from many existing global resource management systems as summarized in [21], our system abstracts all functions and resources as web services, and implements them on a uniform and extensible web service engine: called service agent. The loosely-coupled, language-neutral, and platform independent features of the web service standard make our system be portable, extensible, and easily managed.

The service agent whose component architecture is shown in Figure 1 has the following core functions:

- It is based on standard portable technologies: Java for programming, WSDL for service description, XML-based SOAP RPC for messaging.
- It can publish and unpulish Java classes as services, locally and remotely.
- It provides an service, called on-site publish service, for remote Java classes to be hosted and run as services.
- It provides an interface to intercept and modify SOAP messages. This allows us to plug in security modules: the passport and the service guard.
- It provides service control management to control and monitor executions of different services in policy-based service specific ways. A service controller is dynamically created for each published running service.

Based on the service agent, the management functions of the ARMS are implemented as services and deployed on multiple workstations. The ARMS system has five distinct types of services as shown in the component architecture in Figure 2:

User GUI interface: This is a Java swing based interface that has been implemented as an applet application and an independent application (this implementation is shown in Figure 2). It is designed to be simple and stateless. It performs three major functions: (a)interpreting user inputs and generating requests to the management server; (b) returning management server responses to the user; (c) remotely accessing allocated workstations for interactive applications.

User and resource management service: This service is mainly responsible for (a) managing user sessions that accepts user requests and keeps track of user sessions, (b) allocating resources based on resource requirements, and (c) managing resource domains (the resource domain is discussed in the following). Because this service is responsible for enforcing the global view of the system, it is also called the ARMS service.

Authentication service: This is the only trusted identity in the system, which will be responsible for providing authentication service the all other components in the system. This is the focus of this paper.

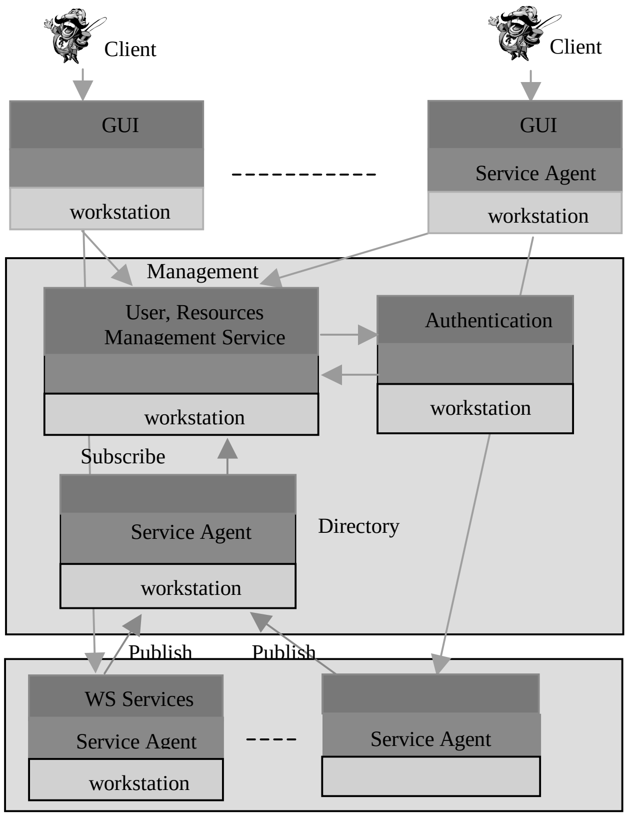


Figure 2. Architecture of the ARMS.

Directory service: The directory service maintains three types of information about the system: user profiles that contain all user information, workstation profiles that contain all static information about workstations, and resource domain indexing files where each file indexes the workstations in the global resource pool that belong to a resource domain. Publish and subscribe interfaces are used for workstations to be published into the system and for the management server to subscribe workstations for the user.

Workstation service: Workstation service is an extended service layer to the basic services provided by the service agent. The current implementation only has a simple workstation monitoring service.

In order to support the dynamic resource sharing and the resource management, a secure domain based resource management model has been implemented. All workstations in the global workstation pool are partitioned into a protected sharing resource domain and multiple protected private resource domains. A workstation belongs to exact one resource domain at a time. A resource domain has a owner. The sharing resource domain is owned by the system and a private resource domain is owned by a unique user. All workstations in the sharing resource domain is accessible to all users (or their applications). Workstations in a private resource domain is accessible to the domain owner and the users granted by the owner.

3.1.1 Security Requirements

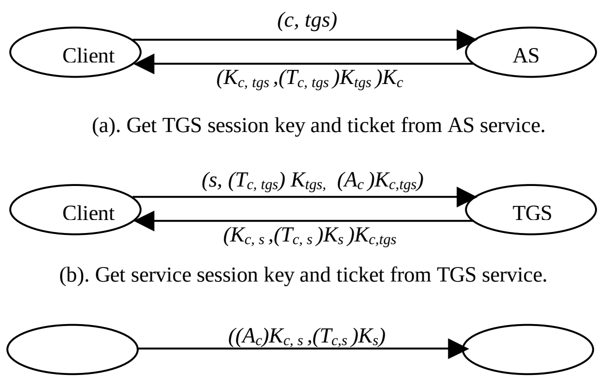
In the ARMS, clients, services, and workstations are geographically distributed and tend to be attacked by all kinds of threats. For a wide-area system, the security is a crucial system requirement, and the authencation infrastructure is the secure foundation. In order to provide a secure shell around the ARMS while keeping a simple and stateless client, the following design goals are identified for the authentication infrastructure:

- *Integrate authentication service into the ARMS system.* From the user point of view, the management server will provide an illusion of a secure and pervasive service with a uniform user interface that can be a browser-based, an applet based, or an independent interface application. This requires that the user interface must be simple and statelss. The management server will be responsible for authenticating user, but it cann't authenticate the user on behalf of the user.
- *Separate authentication infrastructure from ARMS functionality and enforce it transparently.* This allows the security functions to be extended and changed without affecting the ARMS functionality.
- *Separate access control policy from enforcement.* This allows the changes to access control policy without changing core services of the system so that the policy can be dynamically managed.

The authentication infrastructrue proposed in this paper aims at achieving the above goals.

3.2 Kerberos Authentication

Kerberos [5][20][27] is a symmetric key cryptography based authentication system. Authentication is achieved when one party proves knowledge of a shared secret, called ticket, to another. Kerberos relies on two services: the authentication service (AS) and the ticket granting service (TGS), together named as the key distribution center (KDC). AS is responsible for generating password-based symmetric keys for kerberos principals, generating symmetric session keys for TGS communication sessions, and issuing TGS tickets. TGS is responsible for generating symmetric keys for service communication sessions and issuing service tickets.


(c, tgs)
Client
AS
$(K_{c,tgs}, (T_{c,tgs})K_{tgs})K_c$
$(s, (T_{c,tgs})K_{tgs}, (A_c)K_{c,tgs})$
Client
TGS
$(K_{c,s}, (T_{c,s})K_s)K_{c,tgs}$
$((A_c)K_{c,s}, (T_{c,s})K_s)$

(a). Get TGS session key and ticket from AS service.

(b). Get service session key and ticket from TGS service.

(c). Access service using service ticket.

Figure 3. Three-way authentication in Kerberos

The Kerberos ticket and the authenticator are two types of credentials that cooperatively fulfill the authentication function. The ticket is for multiple uses and tells a server if the ticket is valid and who is the client. a Kerberos ticket is an encrypted message of client name (*c*), server name (*s*), client address (*addr*), ticket issuing time (T_{start}), ticket deadline (T_{end}), ticket renew time (T_{renew}), ticket lifetime (T_{life}), and client-server session key ($K_{c,s}$), which is shown as follows:

$(c, s, addr, T_{start}, T_{end}, T_{renew}, T_{life}, K_{c,s})K_s$.

The authenticator is generated by a client, which tells a server who is the client. Because the authenticator is timestamped for one-time use, it is used to prevent the ticket replay. The authenticator has the message format of $(c, addr, T)K_{c,s}$ that is encrypted by a client-server session key.

4 DESIGN OF SECURITY INFRASTRUCTURE

Driven by the design goals for the security infrastructure of the ARMS, in this section, we first describes our Kerberos-like authentication system and the access control for the ARMS. Then, we address the security mechanisms which achieve transparent enforcement and the separation of enforcement and access control policy. At last, we will address the integrated security infrastructure in the ARMS.

4.1 Principals

A principal is a system identity to be protected in the ARMS. The ARMS system has the following two types of principals:

- **User**. Users are the principals who access and manage the services of the ARMS. Based on the role of a user in using the ARMS, users are divided into *general users* who use the services and *administors* who manage the services. Each user principal has two security properties: security name, denoted as *sname* and password. Following the name convention of the Kerberos, the security name of a user has the format of "*name/role@DomainName*", where *name* is a system-wide unique name that the user presents to the system in the registration, *role* is either "general user" or "adminstrator", and *DomainName* is the security domain name to which the user belongs. The security name is generated by the system for security management that is transparent to the user.
- **Service**. Based on the web service model, all resources and management functions are abstracted as services. The ARMS has the following types of services:
    - Multi-instance Management Services: These services manage the system, including user management, resource management, security management, and service management. The ARMS has three different kinds of management services: user/resource management service, directory service, and authentication service. Management services can be replicated to run over multiple nodes for scalability and reliability. Each management service has a security name and a password. The security name has format: "*ServiceName/ {(NodeID)+}@DomainName*". Here, *ServiceName* is a predefined name. *(NodeID)+* represents one non-empty list of workstation node IDs on which the service runs where *NodeID* can be either a URL or a IP address.
    - Single-instance Workstation Services: The workstation service visualizes the workstation on which the workstation service runs. A workstation service runs on a unique workstation. Each workstation service has a security name in the form of *"WorkstationName/NodeID@ DomainName"* and a password.
    - Multi-instance Domain Services: The ARMS abstracts a resource domain as a service with a security name and a password. The security name of a resource domain has the format of "*OwnerName/{(NodeID)+}@DomainName", where *OwnerName* is the unique user name of the owner of a resource domain. A resource domain service is a distributed service that runs over the set of workstations in a resource domain to enforce domain protection.

4.2 Authentication

Our Kerberos-like authentication system is a modification to the Kerberos authentication system with aims to facilitate a simple and consistent global user interface of the system while keeping the security strength of the Kerberos authentication. Our authentication system uses Kerberos credentials: the ticket and the authentication. A ticket carries the KDC-proof identity information of a client for a specific service. An authenticator is a proof that the ticket is originally issued to the client, not stolen.

Different from the Kerberos, our authentication system uses intermediate-server based authentication protocols, which is required to build a single global system view. The functionalities of our authentication system are partitioned into three parts: client component, admin service, and KDC (key distribution center) service. The detail component functionalities and authentication protocols are described in this section.

4.2.1 Components Functionalities

To keep the Web service model as our single software development model, the KDC service is designed to be a Web service that manages two transaction-protected databases: authentication database and tickets database. The KDC service is the unique trust identity in the ARMS. All other principals are authenticated againist it. To ease the management of the authentication system, the KDC service is only accessible to the admin principals of the KDC. Initially, a default admin principal is registrated in the authentication database of the KDC. Admin principals are used by admin services that are integrated with the instances of the ARMS service.

The KDC service mainly performs three functions: (a) *Principal registration and update*. For each principal registration request, the KDC verifies the uniqueness of the principal name and generates a password-based private key (see the algorithm in [20]. For a multi-instance service, the addresses of its multiple instances are not used in the key generation in order to flexibly scaling and migrating the service.) into the authentication database. Initially, the KDC service registrates itself into the authentication database and gets a password-based private key that is used for encrypting and decrypting database entries. (b) *Session ticket generation*. For a session request between a client and a service, the kDC verifies the principal names of the client and the service, and generates a random session key and a ticket for the communication between the client and the service. (The same reason as that in the private key generation, the instance addresses of the multi-instance service are not used in the session key generation.) (c)*Ticket renew*. The KDC renews expired tickets and invalidates old tickets in the ticket database.

To facilitate the uniform global user interface of the system, an admin service functions like a proxy of the KDC service by residing in the ARMS service. The admin service will be replicated while the ARMS service is replicated over multiple hosts for scalability. The admin service fulfills all the KDC service functions: principal registration and update, ticket request, and ticket renew for other services. Even though the admin service is integrated into the ARMS service, the admin service and the ARMS service are two different services that the later is authenticated through the former.

In the deployment, the first deployed instance of the admin service uses the default admin service name in the KDC service to get a service ticket to the KDC service. Then, the first instance updates the admin service name to include all instances by using the updating protocol as described in Section 4.2.2.4.

The client component takes care of all client-side authentication functions: maintaining service tickets and principal name, generating authenticators for requests, and renewing tickets.

4.2.2 Authentication Protocols

There are five major protocols: principal registration protocol, getting service ticket protocol, service request protocol, principal updating protocol, and ticket renew protocol. The ticket renew protocol is the same as that in the Kerberos (which can be found in [20]).

4.2.2.1 Principal Registration Protocol

In the ARMS, every principal needs to registrate into the KDC service to generate its private key before it can communicate with other principals. As described in section 4.2.1, the KDC service initially registrates itself into the authentication database. The KDC service is the first service to be deployed in the system.

The registration protocol is descrbed in Figure 5. For registration, a principal first needs to have the certificate of the admin service so that it can have a security way to submit its name and password to the admin service [6]. In the ARMS, the https protocol is used for the initial secure transfer.

When the admin service gets the name and password of a principal, it encrypts them using admin-to-KDC session key $K_{admin,KDC}$ and sends the cipher message to the KDC service. While the admin service requests the KDC service, it also needs to authenticate itself to the KDC service by using the service request protocol as described in Section 4.2.2.3. For an incoming registration requestion, the KDC service verifies the uniqueness of the principal name and generates a private triple-DES key based on the given principal password. When everything succeeds, the KDC acknowledges the registration request with *ok*, which is further forwarded to the client.

4.2.2.2 Getting Service Ticket Protocol

In the ARMS, a client first needs to get a service ticket by communicating with the ARMS service before it can access a service. The getting service ticket protocol is illustrated in Figure 4 which is the major differentiating part of our authentication system from the Kerberos.

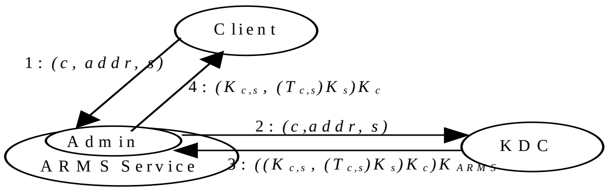

Client
1: (c, addr, s)
4: $(K_{c,s}, (T_{c,s})K_s)K_c$
2: (c,addr, s)
Admin
ARMS Service
KDC
3: $((K_{c,s}, (T_{c,s})K_s)K_c)K_{ARMS}$

Figure 4. Getting service ticket protocol.

To get a ticket to a service, a client uses her principal name and password. Similar to the Kerberos, the client component of the auhentication system keeps the password in the local memory and sends a message to the ARMS containing the client principal name, the client IP address and the service principal name. The ARMS service does access control verification and then invokes the ticket granting service of the admin service.

In turn, the admin service requests the ticket granting service in the KDC by sending a clear messaging containing the client's principal name, the client's IP address, and the service's principal name.

Then, the KDC service verifies if principals of both the client and the service exist. If they do, the KDC generates a new random session key to be used for message encryption and decryption between the client and the service. Moreover, it builds a service ticket for the client to access the service. The ticket contains the client's principal name, the client's IP address, the service's principal name, the session key, the ticket issuing time, the ticket life time, and the ticket renew time.

The ticket is encrypted by the private key of the service so that the service is the only one to decrypt the ticket. The session key and the ticket are encrypted together first by the private key of the client and then by the private key of the ARMS service. The resulting cipher message is sent back to the admin service which subsequently returns the received message to the ARMS service.

Furthermore, the ARMS service decrypts the cipher message using its private key. If the decryption succeeds, the ARMS authenticates itself to prevent a makeup ARMS service. Finally, The decrypted message is sent back to the client, which can only be read by the client.

When the client gets response to its ticket request, it extracts the session key and the service ticket using its private key that is produced at runtime based on the password.

4.2.2.3 Service Request Protocol

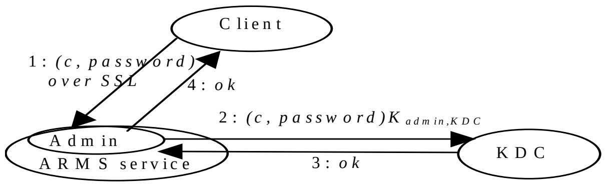


Figure 5. Principal registration protocol.

To access a service, the client first needs to generate a authenticator containing the client's principal name, the client IP address, the current time, and the checksum of the request. It then encrypts the authenticator using the service session key and sends the encrypted authenticator along with the service ticket to the service as shown in Figure 6.

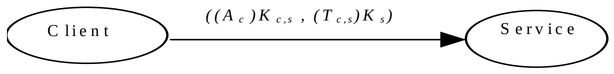


Figure 6. Service request protocol.

When the service gets a service request, it decrypts the received ticket by its private key and extracts the service session key to decrypt the authenticator. Then, the service compares the content in both the ticket and the authenticator to make sure the client is who it acclaims to be and the request comes from the client. The service also needs to make sure that the current host address is one service instance contained in the service name. The communication data integrity is protected by the checksum.

A service may have multiple instances running on multiple hosts. In the Kerberos model, instances of a service are separately treated as different services in the KDC. Accesses to different instances of a service need to use different instance-specific tickets. Even though the separation among multiple service instances is good for security, it complicates the single sign-on of the service. To facilitate the single sign-on over multiple instances of a service without compromising the security, the ARMS system uses a unique global service name for a servic that contains all of its instances as described in section 4.1. To access a service with multiple instances, a client only needs to get a service ticket which can be used for all instances. This makes it easy for the service to dynamically redirect clients' requests to different instances for load balancing and reliablity. The principal name of the service may need to dynamically update in the KDC when the instance set of the service changes, such as adding new instances or deleting some faulted instances.

4.2.2.4 Principal Updating Protocol

For updating the principal name and the password, a client first produces an encrypted message of its old name (*c*), the new name (*c'*) and new password (*p*)(or old password if no password change is needed) by its private key. Then, it sends its name and the encrypted message to the ARMS service and the ARMS service forwards them to the KDC through the admin service. Because the new name and the new password is encrypted, they are not readable to either the ARMS or the admin service.

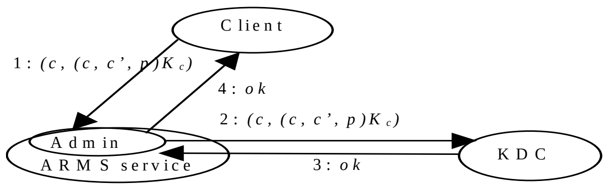


Figure 7. Principal updating protocol.

When the KDC gets the principal updating message, it retrieves the private key of the client based on the clear name in the message from the authentication database. It then decrypts the cipher data in the message. If the client name in the cipher data is the same as the clear name, the correction of the decryption is confirmed. Next, the KDC will update the client name and the private key by generating a new key based on the new name and password. This updating procedure is described in Figure 7.

4.3 Access Control

For management flexibility and scalability, the ARMS system uses the role-based access control model [23] in the web service model. In the role-based access control model, identities are separated from permissions on protected operations through roles so that identities and permissions on operations can be independently changed and managed.

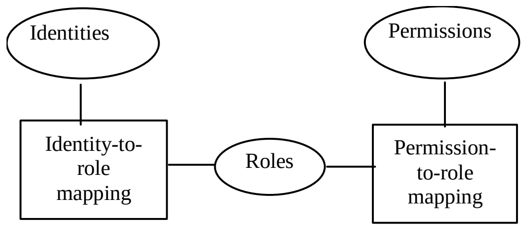


Figure 8. Access control model in the ARMS.

Figure 8 shows the role-based access control model in the ARMS. Under the web service model used in the ARMS, the service is the basic access control unit in the model. The model has five major parts: identities, permissions, roles, permission-to-role mapping, and identity-to-role mapping. Identities refer to principals in the system. Permissions to a service refer to operations of the service.

4.3.1 Roles

Each service has an 1-dimensional role space that contains all roles specific to the service. Table 1 shows the role spaces for the services in the ARMS system. In the ARMS system, the role spaces of all services are kept in a role file in the directory service that is only accessible to the admin of the ARMS system.

Table 1. Role spaces for system services.

Service	**Role Space**
`ARMS`	(*admin, system user, general user*)
`Directory`	(*admin, system user*)
`Workstation`	(*owner, general user*)
`Resource domain`	(*owner, general user*)

4.3.2 Identity-to-Role Mapping

If A principal is able to access a service, the principal must have been assigned to a role in the role space of the service. The assignment of roles to principals is managed by the ARMS service (i.e., the user/resource management service) while principals request service tickets. The ARMS system embeds the assignment of roles to the principal in the service ticket. The service ticket is transferred in an encrypted form so that the assigned roles can't be changed by malicious users.

For the ARMS service, the "*system user*" role is assigned to other services in the system. The "*general user*" role is assigned to the users of the ARMS system.

For directory service, only "*admin*" and "*system user*" roles are allowed. The "*admin*" role is used for management and the "*system user*" role is used for other services (e.g., workstation services) to talk to the directory services. The outside users are not allowed to access the directory service directly.

In both the workstation service and the resource domain service, the "*owner*" role is assigned to the user who owns the workstation or creates the resource domain. The "*general user*" role is assigned to a principal who is allowed to access these services.

4.3.3 Permission-to-Role Mapping

Access control on the operations of a service is guarded by an 2-dimensional permission-to-role mapping matrix that defines which roles can perform which operations. The permission-to-role mapping matrix is managed as a service permission file by the ARMS service.

In a service, access control is conducted only when the authentication verification has been passed because the roles are embeded in the service ticket.

4.4 Security Mechanisms: Passport and Service Guard

To achieve good system extensibility and manageability, it is desired to separate the security infrastructure from system functionalities and the security policy from the enforcement. From the design of the authentication and the access control in previous two sections, the security functionalities of the ARMS system fall into four components: the client component, the service component, the KDC component, and the management component of role file, service permission files, identity-to-role assignment, and the admin of the authentication. The KDC component is an individual service in the system. Regarding the security management component, the identity-to-role assignment function fits well into the admin service of the authentication because the identity-to-role assignment executes at the same time as the ticket issuing. The management of the role file and service permission files is the only task that will be integrated into the ARMS service.

Beside the KDC component and the security management component, the client component and the service component are the most used components in the system, which are encapsulated respectively by two security mechanisms: passport and service guard with aims to achieve enforcement transparency and the separation of policy and enforcement in Web service messaging engine as shown in Figure 9.

4.4.1 Passport

The passport is the client-side security module which consists of an authentication information table (AIT) for all the services requested by the client and an message processing engine.

For each service requested by the client, the AIT maintains an entry of the service URL, the service session key, and the service ticket. The current service entry is pointed by the current service index.

The message processing engine is responsible for encapsulating authentication information into the request message. For each service request, the message processing engine first extracts the session key and the service ticket from the AIT for the requested service. Secondly, it generates an authenticator encrypted by the session key where the authenticator contains client name, client IP address, and the MD5 checksum of the message. Last, it encapsulates both the encrypted authenticator and the service ticket in a SOAP message header (which is shown as the header H(n+1) in Figure 9).

4.4.2 Service Guard

The service guard is the service-side security module which is responsible for the authentication and the role-based access control for the service. The service guard consists of the service private key, a service permission table, an authentication engine, and an access control engine. The service permission table is set by the ARMS service based on the permission file of the service, which assigns permissions (or operations of the service) to roles. This separates permission management from the access control for better manageability and scalability.

For an incoming SOAP request message to a service, the service guard of the service is triggered to conduct authentication and access control.

First, the authentication engine extracts the authentication message header from the incoming SOAP message. It decrypts the service ticket using its private key, decrypts the authenticator using the session key in the ticket, and conducts authentication by peforming the following verification:

- *Data integrity verification*: the authentication engine computes the MD checksum of the message and compares it with the checksum in the authenticator. If results don't match, the service guard sends authentication failure exception back to the client.
- *Client principal authentication*: the authentication engine validates the authenticator and the service ticket based on their timestamps. If valid, the client name and IP address in the authenticator are verified against that in the service ticket. If failed, an authentication exception is sent back to the client.

Secondly, after the authentication passes, the access control engine extracts the roles embeded in the service ticket. It conducts access control by verifying the roles agaiinst the service permission table. When access is allowed, the requested service operation is finally invoked.

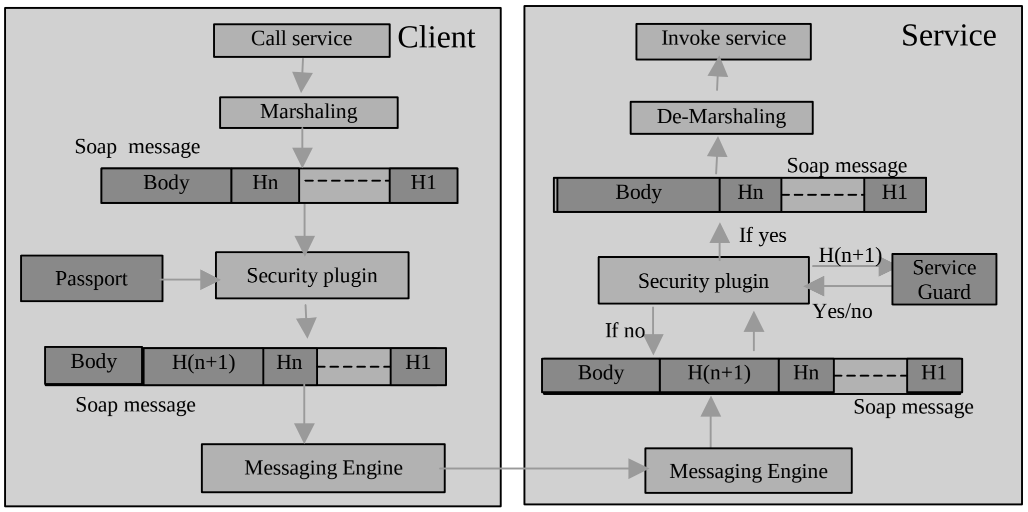


Figure 9. Transparent authentication and access control procedure in Web service messaging.

4.4.3 Transparent Enforcement in the Web-Service Model

In order to separate the security function from the ARMS function, the passport and the service guard are designed to be two security plugins in the SOAP messaging engine as shown in Figure 9.

On the client side (in the left of Figure 9), the passport inserts authentication information in the outstanding request SOAP message after the client's request is marshed into a SOAP message. On the service side (in the right of Figure 9), the service guard conducts authentication and access control on the incoming SOAP message before the message is delivered to the de-marsh engine. This makes the enforcement completely transparent to the application.

5 IMPLEMENTATION

The ARMS system is implemented using pure JAVA based on the Glue 2.3 Web service engine from The Mind Electric Inc. [16]. GLUE was designed and built 100% around web services standards like XML, SOAP, WSDL and UDDI [17].

5.1 Authentication

Our Kerberos-like authentication system was built based on the modification to The Kerberos V5.0 [20]. We adopted the basic mechanisms: authenticator, ticket (with an extension to include principal roles for role-based access control), password-based private key generation algorithm, session key generation algorithm, and checksum generation algorithm. We rebuilt the key distribution center (KDC) and implemented our new authentication protocols.

The KDC was built as a Web service using Glue and Bekelery-DB 4.0.14 [3]. The KDC contains two full transaction protected databases: authentication database for storing private keys and ticket database for storing tickets and session keys. Both databases support locks for consistency control. All data entries were encrypted using the private key of the KDC service.

The KDC has five major functions: registrate a principal, deregistration a principal, change principal password/name, get service ticket for a client, renew a ticket.

The admin service was implemented as the only client of the KDC. The admin service was published in the ARMS service agent, working as a proxy to the KDC. It performs all the functions of the KDC for other principals.

5.2 Access Control

The access control functionalities were implemented in three system components: the ARMS service, the KDC, and the service guard.

The ARMS service manages the role file and service permission files in the directory server. The assignment of principals to roles was implemented in the ARMS service. The KDC encapsulates the assignment of a principal to roles in the service ticket. The access control were enforced in the service guard of the service.

5.3 Security Plug-ins: Passport and Service Guard

The passport and the service guard were plugged into the SOAP messaging engine using Glue's interceptors. By registering an instance of the interceptor using appropriate Context object, the interceptor instance can intercept SOAP message and invokes the processing engines of the passport and the service guard.

The interceptor instance with a passport is registered with the sendRequest of a client so that each outstanding request from this client is intercepted and processed by the passport engine. The interceptor instance with a service guard is registered with the receiveRequest of a service so that each incoming SOAP message is intercepted and processed by the service guard.

6 CONCLUSION

We have designed and implemented a security infrastructure for a Web service model based global resource management system. Our security infrastructure enforces Kerberos-like strong authentication and role-based access control. By encapsulating principal-to-roles assgnment into the service ticket, our system first seamlessly combines the Kerberos-like authentication with the role-based access control model. With the help of two security mechanisms: passport and service guard, transparent enforcement and separate of security policy and security engine have been achieved so that our system can be easily extended and enhanced.

In order to provide a secure shell around the global resource management system while keeping a simple user access interface to the system, a Kerberos-like authentication web-service infrastructure was built based on the Kerberos V5. While using the basic security mechanisms of the Kerberos: authenticator, ticket, private key, and session key, our authentication infrastructure comes up with a new set of authentication protocols so that it can achieve a global secure view of the system.

For access control in the Web service model, our authentication infrastructure integrates the role-based access control model with the Kerberos-like authentication.

Furthermore, our seucirty infrastructure incapsulates the client-side security function and the service-side security function into two security mechanisms: passport and service guard. With the interceptor mechanism of the Web service engine, the passport and the service guard can be transparently plugged into the Web service messaging engine so that the transparent enforcement can be achieved. In addition, the web-service based security infrastructure is also designed to be separated from the resource management functionalites so that the future evolution of the security infrastructure will have less impact on the resource management system.

The current authentication mainly focuses on a single domain. It will be further enhanced to cover multiple seucirty domain and integrate with other existing authentication systems. For the access control, further enhance is to formalize the rules for principal-to-role assignement.

Acknowledgment
I would like to thank Raj Kumar for his discussion and suggestions to this work.

References

[1] Apache Software Foundation. Apache web server. http://www.apache.org.
[2] B. Atkinson, G. Della-Libera, S. Hada, M. Hondo, P. Hallam-Baker, J. Klein, B. LaMacchia, P. Leach, J. Manferdelli, H. Maruyama, A. Nadalin, N. Nagaratnam, H. Prafullchandra, J. Shewchuk, D. Simon, Web Service Security (WS-Security): Version 1.0 05, April 2002. ftp://www6.software.ibm.com/software/developer/library/ws-secure.pdf.
[3] Berkeley DB Tutorial and Reference Guide, Version 4.0.14. http://www.sleepycat.com/docs/reftoc.html.
[4] R. Bulter, V. Welch, D. Engert, I. Foster, S. Tuecke, J. Volmer, C. Kesselman, "A National-Scale Authentication Infrastructure," IEEE Computer, December 2000.
[5] B. Clifford Neuman and Theodore Ts'o. "Kerberos: An Authentication Service for Computer Networks," IEEE Communications, 32(9):33-38. September 1994.
[6] B. Clifford Neuman, Brian Tung, and John Wray. "Public Key Cryptography for Initial Authentication in Kerberos," Internet Draft ietf-cat-kerberos-pk-init-09, July 1999.
[7] B. Clifford Neuman and Glen Zorn. "Integrating One-time Pass words with Kerberos," Internet Draft ietf-cat-kerberos-passwords-02, April 1995.
[8] T. Dierks and C. Allen, "The TLS protocol: Version 1.0," RFC2246, January 1999.
[9] C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. M. Thomas, and T. Ylomen, "Simple Public Key Certificate," Internet Draft draft-ietf-spki-cert-structure-05.txt, 1998.
[10] R. Fielding , J. Gettys, J. Mogul, H. Frystyk , T. Berners-Lee. "Hypertext Transfer Protocol—HTTP/1.1," RFC2068, January 1997.
[11] Alan O. Freier, Philip Karlton, and Paul C. Kocher, The SSL Protocol: Version 3.0, November 18, 1996. (http://www.netscape.com/eng/ssl3/draft302.txt).
[12] J. Franks , P. Hallam-Baker, J. Hostetler, P. Leach, A. Luotonen, E. Sink, L. Stewart. "An Extension to HTTP: Digest Access Authentication," RFC2069, January 1997.
[13] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," International Journal of Supercomputer Applications, 15(3), 2001.
[14] I. Foster, C. Kesselman, J. Nick, S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," http://www.globus.org/research/papers/ogsa.pdf, January, 2002.
[15] K. Fu, E. Sit, K. Smith, N. Feamster, "Dos and Don'ts of Client Authentication on the Web," Proceedings of the 10th USENIX Security Symposium, August 2001.
[16] Glue, The Mind Electric Inc. http://themindelectric.com/docs/glue/guide/index.html.
[17] S. Graham, S. Simeonov, T. Boubez, G. Daniels, D. Davis, Y. Nakamura, and R. Neyama. Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI, Sams, 2001.
[18] Marc Horowitz. Kerberos Change Password Protocol, Internet Draft ietf-cat-ke rb-chg-password-00, March 1997.
[19] International Teclcommunications Union, "ITU-T Recommendation X.509: The Directory: Authentication Framework," Technical Report X.509, ITU, WWW.itu.int, 1997.
[20] J. Kohl and C. Neuman, "The Kerberos Network Authentication Service (V5)," RFC1510, September 1993.
[21] K. Krauter, R. Buyya, and M. Maheswaran, "A Taxonomy and Survey of Grid Resource Mangement Systems for Disitributed Computing," Software – Practice and Experience, 32(2):135-164, 2002.
[22] O. Kornievskaia, P. Honeyman, B. Doster, and K. Coffman, "Kerberized Credential Translation: A Solution to Web Access Control," Proceedings of the 10th USENIX Security Symposium, August 2001.
[23] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. youman, "Role-based Access Control Models," IEEE Computer, Feb. 1996.
[24] Security in a Web Services World: A Proposed Architecture and Roadmap. A joint security whitepaper from IBM Corporation and Microsoft Corporation: Version 1.0, April 2002. ftp://www6.software.ibm.com/software/developer/library/ws-secmap.pdf
[25] J. G. Steiner, B. Clifford Neuman, and J. I. Schiller. "Kerberos: An Authentication Service for Open Network Systems," Proceedings of the Winter 1988 Usenix Conference. February, 1988. (Version 4) text , postscript .
[26] Stone Cold Software. Apache Kerberos Module. http://modauthkerb.sourceforge.net/.
[27] J. S. Park and R. Sandhu, "Role-Based Access Control on the Web," *ACM Transactions on Information and System Security*, Vol. 4, No. 1, February 2001.
[28] B. Tung. Kerberos: A Network Authentication System. Published by Addison Wesley Longman, Inc., 1999.
[29] Web Service Architecture Requirements, W3C Working draft 29, *http://www.w3.org/TR/2002/WD-wsa-reqs-20020429#N100CB*, April 2002.
[30] J. Wilkes, J. Janakiraman, P. Goldsack, L. Russell, S. Singhal, A. Thomas, "Eos, the Dawn of the Resource Economy," *HotOS'2001*, May 2001.
[31] Yong Yan, Zhichen Xu, and Raj Kumar, "Functional Architecture of AMSES: An Automatic Management System for E-service Systems," *HP Lab Technical Report*, HPL-2001-185, July 25, 2001.