A Framework for Heterogeneous Middleware Security

Simon N. Foley, Thomas B. Quillinan, Maeve O'Connor,
Barry P. Mulcahy, and John P. Morrison
Department of Computer Science,
University College, Cork, Ireland.
{s.foley, t.quillinan, b.mulcahy, j.morrison}@cs.ucc.ie

Abstract

With the advent of Web Services, achieving seamless interoperability between heterogeneous middleware technologies has become increasingly important. While much work investigating functional interoperability between different middleware architectures has been reported, little practical work has been done on providing a unified and/or interoperable view of security between the different approaches.

In this paper we describe how Secure WebCom—a distributed metacomputing system—provides interoperability support between the COM+/.NET, CORBA and Enterprise Java Beans middleware security architectures. Secure WebCom uses the KeyNote trust management system to help coordinate the trust relationships between the different middleware systems and their associated security policies. Middleware authorisation policies can be encoded in terms of KeyNote cryptographic certificates, and vice-versa. This provides a unified view of security across heterogeneous middleware systems and also provides the basis for decentralised support of middleware security policies.

KEYWORDS: Middleware; Security; Web Services; Role Based Access Control; Trust Management; Interoperability.

1 Introduction

Interoperability between heterogeneous middleware technologies is typically achieved through the creation of Middleware to Middleware (M2M) bridges [16, 23]. Separate bridges are created for each combination of differing middleware. Utilising these different bridges to provide smooth interoperability between disparate technologies is bridge-dependent and is difficult to generalise. While these bridges provide *functional* interoperability between different middleware architectures [15], little practical work has been achieved on providing a unified and/or interoperable view of *security* between the different approaches.

The provision of a unified view of security policy facilitates the specification and integration of heterogeneous middleware components across complex application systems. To properly address this unification of security policy we propose that the following characteristics should be addressed.

Policy Configuration. Configuring a collection of security policies across different middleware environments is a difficult problem. Each middleware architecture requires a different process for specifying security policies. An approach to specifying a global security policy that can, in turn, be commissioned in a consistent manner across the supported range of middleware architectures is required.

Policy Comprehension. Given an collection of independent security policies that have been configured across a range of middleware systems, then an approach to synthesising these disparate policies into a single unified security policy is required. This synthesis promotes ease of understanding of the current state of the overall system security configuration, and the ability to enforce middleware policies at a more abstract level. Security policies for different middleware architectures can be cataloged and relevant portions enforced in a system-wide basis. In this way, principals in one system can seamlessly utilise components from other systems in the network.

Policy Migration. Given the ability of policy comprehension comes the ability to migrate security policy configurations from one middleware architecture to another. For example, the ability to comprehend a COM+/.NET access policy and synthesise an equivalent access policy for an Enterprise Java Beans Server. Policy migration from one heterogeneous middleware to another requires translation of the security policy. This translation should be generalised to a common format and then applied to the destination system.

Policy Maintenance. Security policy configurations change with time as user and application authorisation requirements change. Such maintenance must be properly coordinated across the different middleware systems to ensure that a consistent view of security policy is maintained. Changes to individual middleware security policy configurations must be propagated across the system to ensure consistency; changes to the global policy must be coordinated with changes to the individual middleware security policies. Providing a means to delegate authority between principals within the system is also useful. In this way, a more localised control of the policy is possible.

Policy Decentralisation. While providing distributed client-server architectures, traditional middleware security mechanisms such as CORBA [2], Enterprise Java Beans (EJB) [27] and Microsoft COM/.NET [20], are effectively based on centralised security policies. Access is determined either by centralised authorisation servers, or by (possibly replicated) policies at local clients; there is little opportunity to coordinate differing access-controls and policies across different domains and regions. Relying on centralised authorisation servers when supporting heterogeneous middleware creates a bottleneck. The ability to decentralise the security policies is desirable.

Emerging trust management schemes [5, 4, 24] use public key authorisation certificates to specify delegation of authorisation between public keys and can be used to help decentralise authorisation policies. Trust Management is an approach to constructing and interpreting the trust relationships between public keys that are used to mediate security critical actions. Cryptographic credentials are used to specify delegation of authorisation among public keys.

Providing a means to distribute these security policies throughout the different Middleware technologies requires a generic bridge between the different systems. WebCom [22] is a distributed metacomputing architecture that provides interoperability between different middleware. WebCom applications are developed using a variety of CORBA, Enterprise Java Beans (EJB) and Microsoft COM/.NET components; WebCom coordinates their execution across the appropriate client/servers, according to a condensed graph [21] that defines the execution sequencing rules for the application. Secure WebCom [14] uses Trust Management [4, 6, 9, 24] to manage authorisation within WebCom. Cryptographic authorisation credentials are used to determine whether it is permitted for a middleware client/server to execute a component. In [14], Secure WebCom uses KeyNote [4] to manage these trust relationships[1].

---

[1] Secure WebCom includes support for SPKI/SDSI. While we use KeyNote in this paper, our results are applicable to SPKI/SDSI.

In this paper we outline how the current version of Secure WebCom is used to provide interoperable security, in particular authorisation, across different middleware environments. This support integrates the authorisation schemes provided by the underlying operating system, the middleware system and KeyNote/trust management based authorisation. The approach provides a number of advantages, addressing the characteristics of Policy Configuration, Comprehension, Migration, Maintenance, and Decentralisation. While conventional middleware security architectures typically rely on a centralised notion of authorisation, KeyNote certificates may be used to promote a decentralised approach. In this paper we show that middleware Role Based Access Control (RBAC) policies can be encoded in terms of KeyNote credentials and vice-versa. This provides a unified view of authorisation within a heterogeneous middleware application system, and also provides the basis for the support of centralised and decentralised RBAC middleware security.

The paper is organised as follows. Section 2 provides an interpretation of authorisation policies in CORBA, EJB and COM middleware in terms of a simple Role Based Access Control Model. An introduction to Trust Management and the KeyNote system in particular, are given in Section 3. Section 4 describes how KeyNote is integrated into WebCom and how interoperability between KeyNote and the underlying middleware security policies is supported. WebCom facilitates the support of both trust management based policies and underlying middleware policies, as described in Section 5. WebCom provides an environment for developing distributed applications built from different middleware components; Section 6 outlines how such applications can be developed when taking security into consideration.

2   Middleware RBAC Security

Role-based access control (RBAC) [26, 29] is widely used to provide access control in Database management systems, operating systems and Middleware architectures. In RBAC, access rights (permissions) are associated with roles, and users are members of these roles. When a user is assigned to a role, they gain all the permissions of that role in the system. This allows an organisation to model its security infrastructure along the lines of its business. The model suits large organisations very well, since roles, like the job positions that they describe, tend to remain relatively static [26]. A role relates closely to organisation's physical job functions, increasing the usefulness of an RBAC security model. RBAC provides administrators with the ability to easily select appropriate roles for users and, when required, the ability to revoke individual user's rights without modifying the permissions on individual objects.

Role Based Access Control (RBAC) is defined in terms

of *Users*, *Roles* and *Permissions* [26, 29]. For the purposes of this paper we extend this to include *Domain* and *Object-Type*. A Domain is a logical grouping of roles, analogous to departments in an organisation. Roles are members of these Domains. The same role name may be present in different domains. Permissions are defined in the context of particular object types. For example, the permission `read` on objects of type `SalariesDB` entitles the holder to have read access on objects of that type. An RBAC policy is defined in terms of the following relations.

$$
\begin{aligned}
RolePerm \quad &: \quad (Domain \times Role) \\
&\leftrightarrow (ObjectType \times Permission) \\
RoleUser \quad &: \quad (Domain \times Role) \leftrightarrow User
\end{aligned}
$$

where $RolePerm((d, r), (o, p))$ means that the role r (in domain d) holds permission p on an object of type o, and $RoleUser(d, r, u)$ means that user u is assigned to domain-role pair (d, r). This extended model is equivalent to the standard RBAC model when $Domain \times Role$ and $ObjectType \times Permission$ are regarded as concretisations of abstract $Role$ and $Permission$. In order to define an appropriate common definition of the RBAC security policies, analysis of the different implementations considered is required.

Enterprise Javabeans [27] EJB components encapsulate the business logic of Sun Microsystems Java 2 Enterprise Edition (J2EE) application model. Components reside in a bean container located on a server, running on a host machine. The combination of host, EJB server, and the relevant bean container JNDI [28] name provide the domains of the policy. Roles are bean specific on each server. Users exist globally in each EJB server, and as such can be members of roles in different domains. Users are unique to each EJB server, and are managed by that server. Permissions represent method calls that a role is permitted to make on an EJB object.

Microsoft COM+/.NET [20] The DCOM specification [20] extends the COM model to provide remote invocation capabilities. COM was designed to allow application developers to create reusable software components. DCOM allows these software components to be called remotely. COM objects can be anything from a simple software program right up to a complex program, such as Word. COM's RBAC model is an extension of the Windows security model and provides Windows NT *Domain*s, roles unique to each domain, and permissions. For the purposes of this paper, COM permissions are `Launch`, `Access`, `RunAs`.

Common Object Request Broker Architecture (CORBA) [2] CORBA is a general and open industry standard for working with distributed objects. It allows the interconnection of objects and applications, regardless of computer language or machine architecture. Using its standard protocol (IIOP) a CORBA-based program on any heterogeneous setup, can interoperate with a CORBA-based program on almost any other heterogeneous setup.

The CORBA RBAC model consists of Roles, Users and Permissions. We consider a Domain to be the name of the machine and the Corba ORB server name, similar to the EJB system. Again Roles are unique to each Domain, and Users can be members of one or many roles. Permissions relate to the method calls on objects of the given object type.

The RBAC policy can be implemented in each of these Middleware systems in a common manner. Figure 1 provides a simple example of an RBAC policy for `SalariesDB` objects in an organisation with domains (departments) Finance and Sales, and roles Clerk, Manager and Assistant.

3 Trust Management

Unlike identity based authorisation systems, for example those using X.509 [8] certificates, where authorisation is based on linking an identity to a public key, Trust Management addresses the need to associate abilities to public keys. In other words, identity based certificates answer the question "Who is the holder of this public key?" whereas ability based certificates answer the question "What can I trust this public key to do?".

Conventional secure applications verify that certificates have not been revoked, and are signed by a recognised and trustworthy source. The names are then extracted from the certificates and a database is queried to determine if the requested action is authorised. This is cumbersome and aspects, such as the database lookup, are outside of the scope of the certificate system. Furthermore, there is the problem of determining the correct identity of an individual: there may be more than one John Smith in a particular organisation[10].

Trust management systems eliminate the extraction of names and database lookup. The certificates holding the abilities of the public key requesting the action are instead submitted to the trust management system, along with the action request. The trust management system verifies that the action is authorised by the certificates provided. For the purposes of authorisation, trust management systems are not concerned with verifying personal identity of a requester. These questions, while valid security questions,

Domain	Role	Permission
Finance	Clerk	write
Finance	Manager	read/write
Sales	Manager	read
Sales	Assistant	*no access*

Domain	Role	User
Finance	Clerk	Alice
Finance	Manager	Bob
Sales	Manager	Claire
Sales	Assistant	Dave
Sales	Manager	Elaine

Figure 1. RBAC relations for a Salaries Database

are not relevant to an application attempting to determine whether an action is authorised.

Trust Management systems have a number of advantages compared to the traditional systems created using X.509. Policies and certificates are created and maintained separately from the application in a very natural way. The attributes used within the policies/certificates are application defined, and they are represented in a free-form fashion, allowing the application designer to decide what characteristics are required. Changing the format of the attributes does not require changes to the trust management system used. By removing the traditional lookup of an identity's authority, and instead representing that authority within the certificate, applications no longer need to consider the security of where this authority is stored. An additional benefit of utilising a trust management system within applications is that designers and implementers of these applications are required to consider trust management applications explicitly. This in itself encourages good practices when considering the overall security of such applications. Trust management policies are easy to distribute across networks, helping to avoid reliance on centralised configuration of distributed applications.

KeyNote [3, 4] is an expressive and flexible trust management scheme that provides a simple credential notation for expressing both security policies and delegation. A standard KeyNote Application Programming Interface is used by an application to make queries about whether security critical requests (to the application) have authorisation or not. The formulation and management of security policies and credentials are separate from the application, making it straight forward to support trust management policies across different applications. KeyNote has been used to provide trust management for a number of applications including active networks [7] and to control access to Web pages [1].

When a request from an untrusted principal (key) is made to a networked application to execute a particular action, then, authentication notwithstanding, the application must determine whether the key(s) that made the request is authorised. Authorisation comes in the form of digitally signed public key credentials that bind public keys to the authorisa-

tion to perform various actions. In practice, authorisation is achieved by a collection of credentials that exhibit the necessary trust relationships between their keys. Given a policy (public keys, trusted in known ways), and a collection of credentials, a network application must determine whether a particular public key is authorised to request a particular operation.

Example 1 A simple Salaries application runs on a server and accepts requests from clients to access the salaries database of a company (see Figure 1). The request for operation read is made to request an employee's salary information while the request write is used to make modifications to the database. We expect that, in practice, a clerk will have the authority to write to the database (in order to add new employees) and managers will have the authority to both read and write to the database; this authority will be delegated by senior management.

Assume that the owner of public key Kbob is trusted to make requests to the Salaries application. This is specified by the following KeyNote credential [2].

```
Authorizer: POLICY
licensees:  "Kbob"
Conditions: app_domain=="SalariesDB" &&
        (oper=="read" || oper=="write");
```

Figure 2. Policy Credential allowing Manager Bob to read from and write to the database

This is a special *policy* credential that defines the conditions under which requests from the licensee key Kbob may be trusted by the application SalariesDB. These conditions are defined using a C like expression syntax in terms of the *action attributes*, in this example, app_domain and oper which are used characterise the circumstances of a request.

The owner of public key Kbob (The company's finance manager) has the authority to delegate this trust to other

---

[2]Note: this credential does not represent an RBAC role. Such credentials are considered in Section 4.
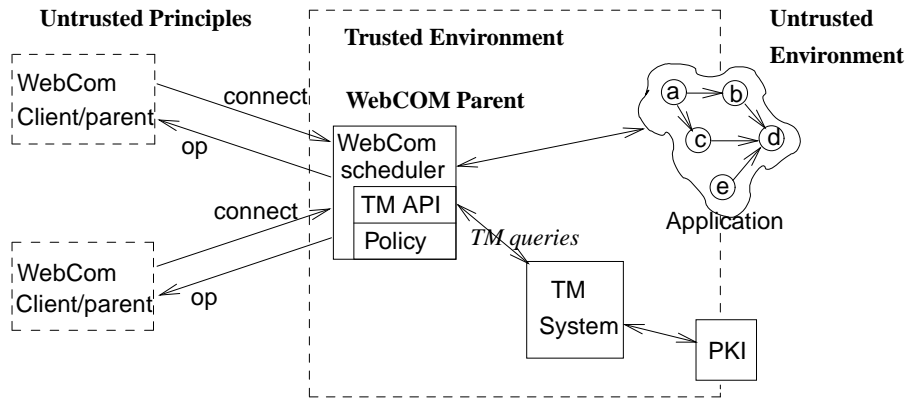
Figure 3. WebCOM-KeyNote Architecture

keys and does so by signing the following credential for a clerk who owns public key `Kalice`.

```
Authorizer: "Kbob"
licensees:  "Kalice"
Conditions: app_domain=="SalariesDB"
            && oper=="write";
```

Figure 4. Credential allowing Clerk Alice to write to the database

In signing this credential, authoriser `Kbob` delegates authority for writing to the database to the key `Kalice`. When Alice attempts to write to the database (sending a request signed by `Kalice`), she presents this credential as proof of authorisation. We can confirm that this key is indeed authorised since, by default (policy), we trust `Kbob` to read and write to the database and `Kbob` has delegated some of this trust to `Kalice`, by virtue of signing the credential. △

An application may use a Trust Management (TM) scheme such as KeyNote [4] to determine whether requests to it are authorised, without the application having to know about how that determination is made.

Example 2 When the `SalariesDB` application (Example 1) queries the KeyNote TM system to determine whether it is safe to execute a particular request, it must specify the circumstances of the query. These circumstances include: *action authorizers*, corresponding to the key(s) that made the request; *action attribute set*, which is a set of action attribute name and value pairs that characterise the request; *policy* credentials, representing the keys that are trusted, and other *credentials* as provided by the requester and/or PKI.

For example, when `Kalice` requests to write to the database then the order application queries KeyNote with action authorizer `Kalice`, action attribute set {app_domain ← "SalariesDB", oper ← "write"}, the policy credential for `Kbob` above, and a set of signed credentials provided by Alice. KeyNote must determine if the given request is authorised based on the evidence provided. The application interacts with KeyNote via calls to the KeyNote API. △

The KeyNote architecture provides a level of separation between the provision of security policy authorisation and application functionality. As a software engineering paradigm, techniques that support separation of concerns for security [4, 11], synchronisation [19], and so forth are desirable since they lead to applications that are easier to develop, understand and maintain.

4 Secure WebCom

Secure WebCom [14, 22] is a distributed secure and fault-tolerant architecture that can be used to coordinate the distributed execution of middleware components across a network. A Secure WebCom environment [14] uses KeyNote to help manage the trust relationship with other Secure WebCom environments.

Figure 3 illustrates how the KeyNote trust management scheme is integrated into WebCom by regarding WebCom as an application.

The WebCom master authenticates its clients and uses their credentials to determine what operations it may schedule to them. Each WebCom client has a trust management architecture that is similar to the KeyNote architecture in [4], authenticating the master and using the master's credentials to determine whether it is authorised to schedule the operation. We originally selected KeyNote because of

5

```
Authorizer: Policy
Licensees:  "KWebCom"
Conditions: app_domain == "WebCom" &&
  ObjectType == "SalariesDB" &&
   (Domain=="Sales" && Role=="Manager" && Permission=="read") ||
   (Domain=="Finance" && Role=="Manager"
      && (Permission=="read"||Permission=="write"))||
   (Domain=="Finance" && Role=="Clerk"  && Permission=="write");
```

Figure 5. WebCom's Policy for the Salaries Database

its simplicity and expressiveness; as previously noted, we have since used the SDSI/SPKI system in a similar way. KeyNote provides a simple notation for specifying both local security policies and credentials that can be sent over untrusted networks.

A Secure WebCom environment can automatically convert middleware RBAC policies to their equivalent KeyNote policies/credentials, and vice-versa. This provides a high degree of policy interoperability, between the middleware and trust management layer, and also within the different middlewares. In addition to providing a uniform way of specifying RBAC policies for different middleware systems, it also becomes possible to enforce standardised RBAC middleware policies across middleware systems that do not have a configured RBAC policy.

RBAC-like policies can be encoded in terms of equivalent cryptographic certificates/policies [18]. Secure WebCom is configured to support middleware RBAC-like security policies by effectively encoding the $RolePerm$ and $RoleUser$ relationships (from Section 2) within KeyNote authorisation credentials. This is unlike [18] which integrates authorisation certificates as part of the lower-level middleware system.

Secure WebCom uses KeyNote to determine whether it is safe to execute a middleware component. The circumstances of the action are described in terms of attributes coded within the credentials. WebCom uses the attributes: `Domain`, `ObjectType`, `Role`, `Permission` which correspond to the RBAC attributes described above.

The $RolePerm$ table of Figure 1 is encoded as KeyNote Policy credential shown in Figure 5.

This in effect specifies that the WebCom administration key `KWebCom` is authorised to administer rights in connection with this policy.

Key `KWebCom` can delegate authorisation for role `Manager` in domain `Finance` to Claire by writing and signing the credential shown in Figure 6. While the policy above centralises the $RolePermission$ table, it could alternatively be decentralised and spread across a number of credentials and additional authorisations and role memberships delegated to other keys. For example, Claire can delegate her role to `Kfred` by writing the credential shown

```
Authorizer: "KWebCom"
Licensees:  "Kclaire"
Conditions: app_domain == "WebCom" &&
    Domain=="Finance" && Role=="Manager";
Signature: ...
```

Figure 6. Claire is authorised to be a Manager in the Finance Domain

```
Authorizer: "Kclaire"
licensees:  "Kfred"
Conditions: app_domain=="WebCom" &&
     Domain=="Sales" && Role=="Manager";
Signature: ...
```
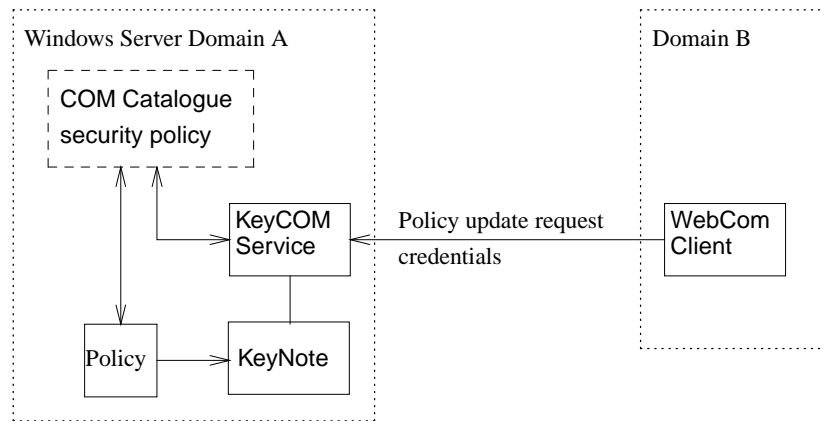
Figure 7. Claire delegates her Role membership to Fred

in Figure 7.

4.1 Policy Configuration

It is not necessary to rely on just the security/KeyNote mechanism of Secure WebCom; the underlying middleware and operating system also provide RBAC security mediation. In this case it is necessary to translate the specified KeyNote RBAC policy into its equivalent middleware RBAC security configuration. This support is provided for by WebCom.

On each WebCom environment a secure automated administration service accepts KeyNote credentials and updates the local middleware security policy configuration to reflect the authorisations granted by the credentials. This is currently implemented for COM and EJB middleware systems. For example, Figure 8 outlines how a user, currently registered only in Domain B, is integrated into a COM+ RBAC policy within Domain A. The KeyCOM service of WebCom accepts a policy update request (plus KeyNote credentials) and if valid it updates the security policy in the

Figure 8. Decentralised Middleware Architecture

COM Catalogue (middleware/Windows RBAC policy) with the equivalent authorisation. KeyCOM acts, in effect, as an automated Windows/COM administrator, processing client authorisation requests, while the KeyNote cryptographic credentials facilitate users in delegating authorisation without requiring assistance of non-automated (that is, human) administrators. Similar services exist for other supported middleware systems.

4.2 Policy Comprehension

It is also useful to translate middleware RBAC policies into equivalent (WebCom) KeyNote RBAC policies. In this case the middleware RBAC policy can alternatively be enforced by Secure WebCom. A middleware RBAC policy comprised of relations $RolePerm$ and $RoleUser$ is converted into KeyNote credentials as follows.

The $RolePerm$ is encoded as a KeyNote Policy that authorises the WebCom Key as administrator for the RBAC policy and is authorised for the given values over attributes `domain`, `role`, `ObjectType` and `Permission`. For each user (public key) in the $RoleUser$ table, a credential is generated, and signed by the WebCom key, authorising the user to be a member of the corresponding roles from $RoleUser$.

This process aids comprehension of the overall policy through the definition of the entire policy in one common format. Using this process it is possible to enforce an equivalent security policy at a more abstract layer.

4.3 Policy Migration

Migration of existing policies from one middleware system to another is also possible. This interoperability of disparate security policies allows, for example, a new system to be configured with the same policy as an existing system.
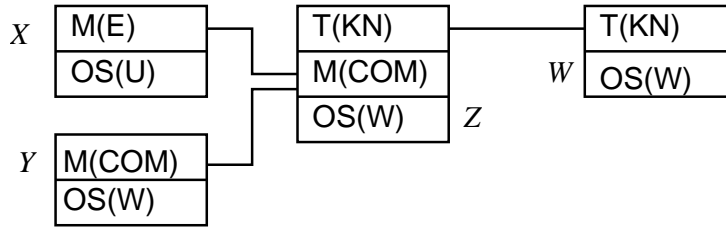
Figure 9 illustrates this interoperability with a sample configuration. System Z is a WebCom Server, while systems X, Y and Z are WebCom clients. System Z security relies on the Windows (W) operating system, COM middleware and KeyNote trust management. These systems could have independent policies, or might require a more homogeneous policy across the different platforms. A WebCom client running on Windows with COM middleware security policy inter-operates with the server. If required, the KeyNote RBAC credentials held by users of System Z can be used to update the COM+ catalogue of System Y. On the other hand, the COM middleware RBAC policy on System Y can be translated to equivalent KeyNote credentials and these, in turn, used by System W which does not have a middleware security mechanism. In addition, if System Y was a legacy system under migration to System X, then the KeyNote credentials generated from the legacy COM policy can be used to automatically configure the replacement EJB RBAC policy.

It should be noted that migration of policies between different middleware technologies does not consist of a simple one-to-one mapping. Some interpretation of the security policies must be considered by the translation tools, using techniques such as similarity metrics [13].

4.4 Policy Maintenance

The maintenance of a consistent global policy across the different heterogeneous middlewares is important for the overall security if the system. Making changes to the underlying middleware security policies can lead to inconsistencies between the authorisation of principals on different systems.

Maintaining both middleware and trust management policies is an important aspect of the Secure WebCom system. For example, if a new employee is to be added to

7

Figure 9. Interoperating Security Policies

the existing policy, changes must be propagated to all the relevant heterogeneous system policies. It is possible to add entries to the underlying middleware security policies and then utilise the translation services to propagate these policy changes to the other systems. However, we recommend changing the trust management policy to reflect required changes in the system. This enables the changes to be propagated down the security stack where necessary, while maintaining the consistency of the overall security policy. Making the changes to the trust management policy has additional benefits. For example, a manager can delegate authority to a new employee without requiring detailed knowledge of the underlying systems in use.

The manager can achieve this by the creation of new credentials, assigning the employee the roles required. Figure 7 shows an example of such role delegation. The administrative translation services can update the middleware policies to permit the new employee to use the services appropriate to their function. Maintaining the policy in this fashion has the additional benefit that it provides a high-level view of the overall policy of the organisation.

4.5 Policy Decentralisation

The trust management system used by WebCom provides the means to make policy decisions in a decentralised manner.

5 Stacked Authorisation

Using the trust management architecture in Secure WebCom requires that the WebCom environment be trusted in the sense that part of the security mediation (authorisation) is performed by the WebCom environment and not the underlying operating system. An advantage of this approach is that since it is independent of the security architecture of the underlying system then it provides a better opportunity for interoperation between heterogeneous platforms that run the WebCom environment. However, since it does not rely on the underlying operating system/middleware authorisation mechanisms, a result is that it increases the software in

the trusted computing base.

In the case where the WebCom system is not trusted by the operating system, the security policy of the middleware applications is enforced. This implies that a choice between enforcing the middleware and WebCom security policies must be made.

We address this property by considering how the security mechanisms of the underlying middleware/operating system can be used to provide the basis of security mediation and form a part of the overall WebCom security architecture. This provides a stack of security layers, as depicted in Figure 10. Note that the Level 3 security corresponds to mechanisms encoded within the condensed graph that is used to coordinate the application components. It is used to implement application level workflow security, for example [12], and is not considered in this paper.

These stacked layers of secure WebCom are 'pluggable' in the sense of [17, 25]; for example, in the absence of CORBASec support for a particular ORB, a WebCom environment could be configured so that authorisation is based only on a combination of KeyNote (trust management) and underlying operating system policy.
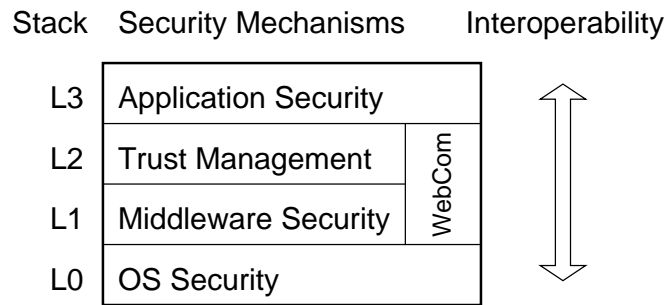
Using this stackable architecture, applications can be developed that use trust management policies for new components and the middleware security architectures for existing security mediations. This has the benefit of using legacy code and policies, minimising inconsistencies in the conversion of applications.

6 Secure Heterogeneous Application Development

A distributed application is constructed as a condensed graph of components using an Integrated Development Environment (IDE) for WebCom. By integrating secure middleware components into the IDE, it provides a platform to build distributed middleware applications that leverage existing middleware business logic and security policies. Through this approach, programmers can easily build secure and complex distributed applications.

To incorporate the existing middleware components as

8

Stack   Security Mechanisms   Interoperability

Stack	Security Mechanisms	
L3	Application Security	
L2	Trust Management	WebCom
L1	Middleware Security	
L0	OS Security	

Figure 10. Stacked Security Architecture in WebCom

part of a WebCom application, the middleware services need to be interrogated. That is, extract the relevant middleware components, and make them available to application developers through the use of a component palette on the WebCom IDE. Interrogation is achieved using a WebCom plugin specific to the middleware service in question.

To facilitate secure heterogeneous application development the middleware interrogation process also extracts security policy information related to the middleware components. This information is presented in the IDE as an additional palette (Figure 11). The IDE analyses the middleware component currently highlighted, and determines which combinations of domain, role and user is suitably authorised (holds permissions) to execute the selected component.

The programmer may specify any valid combination of domain, role and user for a component to execute under. The Webcom scheduler ensures components are scheduled to the specified domain, role, and user. A partial specification is also supported, for example, allowing the programmer to specify a domain and role for a given component, in which case it will be scheduled to any authorised user in the specified domain and role.

7 Discussion and Conclusion

We have outlined how to provide interoperable security support for the different middleware environments that are supported by WebCom. With this approach, KeyNote certificates provide a decentralised approach for authorisation. We have also outlined how middleware RBAC policies can be encoded in terms of KeyNote credentials and vice-versa. This provides a unified view of security of a heterogeneous middleware application system, and also provides the basis for the support of centralised and decentralised RBAC middleware security.

This system provides the infrastructure to allow different technologies to contribute to the overall security of the system. It provides the ability to construct richer security

policies. For example, utilising the existing setup present in legacy systems. Providing such interoperability also raises the potential to transfer relevant parts of security policies, either to enforce them in KeyNote or in the middleware RBAC system, where appropriate.

Traditionally, work in this area has focused on replacing existing security models with Trust Management systems. We have introduced a system that harnesses the existing RBAC systems, and uses Trust Management to create more integrated policies. There are advantages to this approach over existing work, such as [18]. For example, we have the ability to both abstract existing security policies into Trust Management policies and conversely enforce portions of Trust Management policies in terms of middleware RBAC policies.

The system as outlined above has some limitations: in order to maintain a coherent security policy, we must have the ability to name objects in the entire system in a consistent and reliable fashion. The current system provides for making mediation decisions purely on the identifier of the components. Extending this to consider the environment of the component, its inputs, and so forth, is a topic of ongoing research.

References

[1] Apache-ssl release version 1.3.6/1.36. Open source software distribution. http://www.apache.org.

[2] B. Blakley. *Corba Security. An Introduction to Safe Computing with Objects.* Object Technology Series. Addison-Wesley, 2000.

[3] M. Blaze. Using the KeyNote trust management system. http://www.crypto.com/trustmgt, December 1999.
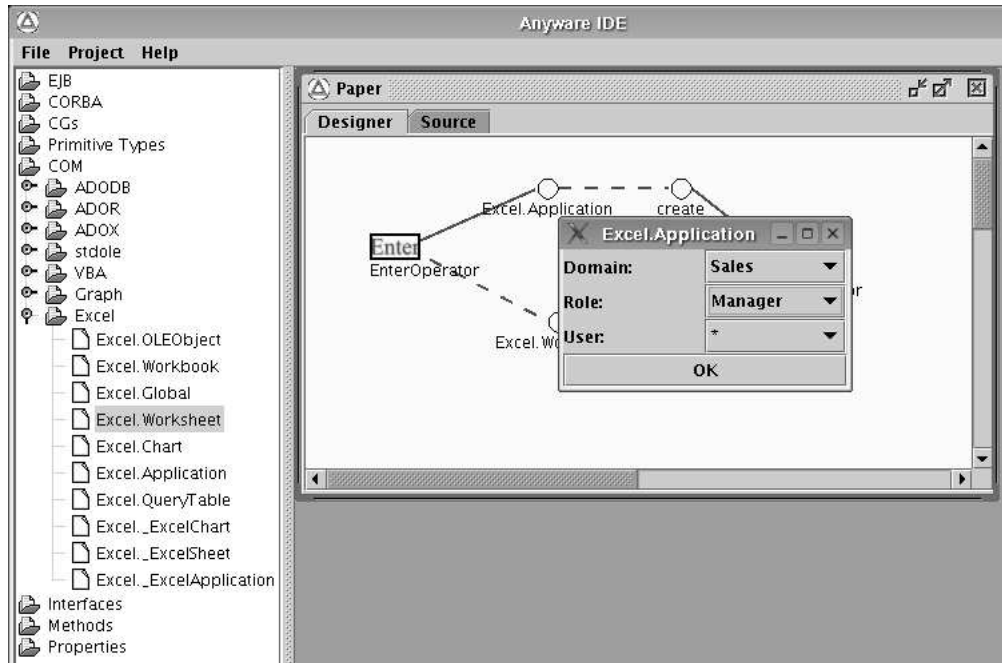
9

Figure 11. The WebCom Integrated Development Environment

[4] M. Blaze et al. The keynote trust-management system version 2. Sept. 1999. Internet Request For Comments 2704.

[5] M. Blaze et al. The role of trust management in distributed systems security. In *Secure Internet Programming: Issues in Distributed and Mobile Object Systems*. Springer-Verlag Lecture Notes in Computer Science, 1999.

[6] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proceedings of the Symposium on Security and Privacy*. IEEE Computer Society Press, 1996.

[7] M. Blaze, J. Ioannidis, and A. Keromytis. Trust management and network layer security protocols. In *Security Protocols International Workshop*. Springer Verlag LNCS, 1999.

[8] CCITT Draft Recomendation. *The Directory Authentication Framework, Version 7*, Nov. 1987.

[9] C. Ellison et al. SPKI certificate theory. Sept. 1999. Internet Request for Comments: 2693.

[10] C. M. Ellison. The nature of a useable PKI. *Computer Networks*, (31):823–830, 1999.

[11] S. Foley. A kernelized architecture for multilevel secure application policies. In *European Symposium on Research in Security and Privacy*. Springer Verlag LNCS 1485, 1998.

[12] S. Foley and J. Morrison. Computational paradigms and protection. In *ACM New Computer Security Paradigms*, Cloudcroft, NM, USA, 2001. ACM Press.

[13] S. N. Foley. Supporting imprecise delegation in keynote using similarity measures. In *Proceedings of The Sixth Nordic Workshop on Secure IT Systems*, pages 101–119, Copenhagen, November 2001.

[14] S. N. Foley, T. B. Quillinan, and J. P. Morrison. Secure component distribution using webcom. In *Proceeding of the 17th International Conference on Information Security (IFIP/SEC 2002)*, Cairo, Egypt, May 2002.

[15] R. Geraghty, S. Joyce, T. Moriarty, G. Noone, and S. Joyce. *COM-CORBA Interoperability*. Number ISBN: 0-130-96277-5. Prentice Hall PTR, 1998.

[16] J. Hugues, F. Kordon, L. Pautet, and T. Quinot. A case study of middleware to middleware: Mom and orb interoperability. In *Proceedings of the 4th International Symposium on Distributed Objects and Applications (DOA'02)*, Irvine, CA, USA, Oct. 2002. University of California, Irvine.

[17] N. Itoi and P. Honeyman. Pluggable authentication modules for Windows NT. In *Proceedings of the 2nd USENIX Windows NT Symposium*, pages 97–108, Seattle, Washington, August 1998.

[18] T. Lampinen. Using SPKI certificates for authorization in CORBA based distributed object-oriented systems. In *4th Nordic Workshop on Secure IT systems (NordSec '99)*, pages 61–81, Kista, Sweden, November 1999.

[19] C. Lopes and K. Lieberherr. Abstracting process-to-process relations in concurrent object-oriented applications. In *European Conference on Object-Oriented Programming (ECOOP)*. Springer Verlag LNCS 821, 1994.

[20] Microsoft Corporation. *Microsoft Platform SDK. The COM Library. Microsoft Developer Network.*, 0.9 edition, October 1995. http://www.msdn.microsoft.com.

[21] J. Morrison. *Condensed Graphs: Unifying Availability-Driven, Coercion-Driven and Control-Driven Computing*. PhD thesis, Eindhoven, 1996.

[22] J. Morrison, D. Power, and J. Kennedy. A Condensed Graphs Engine to Drive Metacomputing. Proceedings of the international conference on parallel and distributed processing techniques and applications (PDPTA '99), Las Vagas, Nevada, June 28 - July1, 1999.

[23] T. Quinot, F. Kordon, and L. Pautet. From functional to architectural analysis of a middleware supporting interoperability across heterogeneous distribution models. In *Proceedings of the 3rd International Symposium on Distributed Objects and Applications (DOA'01)*. IEEE Computer Society Press, Sept. 2001.

[24] R. Rivest and B. Lampson. SDSI - a simple distributed security infrastructure. In *DIMACS Workshop on Trust Management in Networks*, 1996.

[25] V. Samar and R. Schemers. Unified login with pluggable authentication modules (PAM). Request for Comments 86.0, Open Software Foundation, October 1995.

[26] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[27] Sun Microsystems. *Enterprise JavaBeans(tm) Specification, Version 2.1*, June 2003. http://java.sun.com/products/ejb/docs.html.

[28] Sun Microsystems Inc. *Java Naming and Directory Interface*, 1.2 edition. http://java.sun.com/products/jndi/.

[29] X. Zhang, S. Oh, and R. Sandhu. PDBM: A flexible delegation model in RBAC. In *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT 2003)*, Como, Italy, June 2003.

Author Biographies

Simon N. Foley holds a PhD on the modeling of security-critical systems and is a Statutory Lecturer in Computer Science at University College Cork where he teaches and directs research on computer security. He serves on the editorial board of the Journal of Computer Security and has served as Program chair of the IEEE Computer Security Foundations Workshop and the ACM New Security Paradigms Workshop.

Thomas B. Quillinan holds a B.Eng in Computer Engineering from the University of Limerick and a M.Sc. in Computer Science from University College Cork. He is currently pursuing a Ph.D. in Computer Security with the Centre for Unified Computing in University College Cork. His academic interests include secure naming, trust management and security in distributed systems.

Maeve O'Connor holds a B.E. and a M.Eng.Sc. in Bio-Engineering from University College Dublin, and has published papers on bread manufacturing. She has extensive experience working in Industry in the area of Software Engineering. She also holds a M.Sc. in Computer Science from University College Cork.

Barry P. Mulcahy holds a B.Sc. in Computer Science from University College Cork. He is currently pursuing a Ph.D. in Computer Security with the Centre for Unified Computing in University College Cork. His academic interests include decentralised security policies, distributed security mechanisms, and secure workflows.

John P. Morrison is a Statutory Lecturer in the Computer Science Department in University College Cork. He has previously worked in Phillip's Natuurkundig Laboratorium, Eindhoven, Holland. Dr. Morrison received his Ph.D. from the Technische Universiteit Eindhoven. He is a Senior Member of IEEE, a member of the ACM, Research director of the Centre for Unified Computing, Co-director and co-founder of Grid-Ireland and a Co-director of the Boole Centre for Research in Informatics in UCC.