Secure Virtual Enclaves: Supporting Coalition Use of Distributed Application Technologies

Deborah Shands, Richard Yee, Jay Jacobs E. John Sebes
NAI Labs at Network Associates *Kroll-O'Gara*
{dshands, ryee, jjacobs}@nai.com *ejs@securify.com*

Abstract

The Secure Virtual Enclaves (SVE) collaboration infrastructure allows multiple organizations to share their distributed application objects, while respecting organizational autonomy over local resources. The infrastructure is transparent to applications, which may be accessed via a web server, or may be based on Java RMI, or Microsoft's DCOM. The SVE infrastructure is implemented in middleware, with no modifications to COTS operating systems or network protocols. The system enables dynamic updates to security policies to support changes in both coalition membership and participants' perception of risks. While the prototype demonstrates fine-grained access control for secure collaborative computing, we have identified significant issues that remain to be addressed, particularly in the area of policy development, before such collaboration will be convenient. The SVE infrastructure offers a platform and conceptual basis for further exploration of these issues and experimentation with new solutions.

1. Introduction

The need for mechanisms to allow organizations to collaborate securely is recognized in many environments. Military alliances and joint task forces are formed to accomplish a common goal and the participating organizations engage in some form of distributed collaborative planning. After a natural disaster, crisis management collaborations are formed from an often disjoint collection of disaster/incident response teams (e.g., medical personnel, local police, engineers). In a commercial environment, companies outsource some of their operations (e.g., payroll, data center operations), employ contractors to perform certain tasks, or offer some of their data to customers. They may also form consortiums to perform collaborative research, develop standards, or battle competitors. There are at least two common elements in any of the resulting scenarios: (1) the collaborating organizations have *limited trust* in one another, and (2) the coalitions are *dynamic*.

Because the organizations may have competitive or even adversarial relationships, they do not completely trust one another. They are, however, motivated by a common goal to share some of their resources. Their trust in one another and the limits of that trust are generally specified through some extra-technological means, such as contracts, treaties, or memoranda of agreement.

Coalitions are likely to be dynamic, in that organizations may join or leave over the lifetime of the collaboration. An organization's level of trust in its partners may also change with time, impacting the degree of resource sharing–local resources may be added or removed from the sharing arrangement. The mode of access to a particular resource may also change over time.

We believe that the degree of dynamism necessary to support coalition creation, evolution, and eventual dissolution preclude a hardware-intensive solution (e.g., setting up a new, joint network). Virtual Private Networks (VPNs) can authenticate individual users, but do not support access controls on fine-grained objects (e.g., a Java interface/method). VPN-based solutions are also relatively static, as adding new coalition members requires some manual reconfiguration. To support fine-grained access controls and dynamic changes to coalition membership, the SVE project focused on software solutions.

The goal of the Secure Virtual Enclaves (SVE) project was to develop software technology to enable multiple enclaves to engage in controlled collaborative computing using distributed applications, while retaining organizational autonomy over local resources. By *enclave*, we mean a collection of computers and networks managed by the same organization and subject to the same security policy. *Collaboration* occurs when principals in partner enclaves are permitted to access selected local resources. Security *controls* are necessary to ensure that collaborators get only the intended access to local resources. Local *autonomy* is exceedingly important, as an organization's willingness to share its resources with others is influenced by the degree of control it retains over those resources. An organization that knows it can

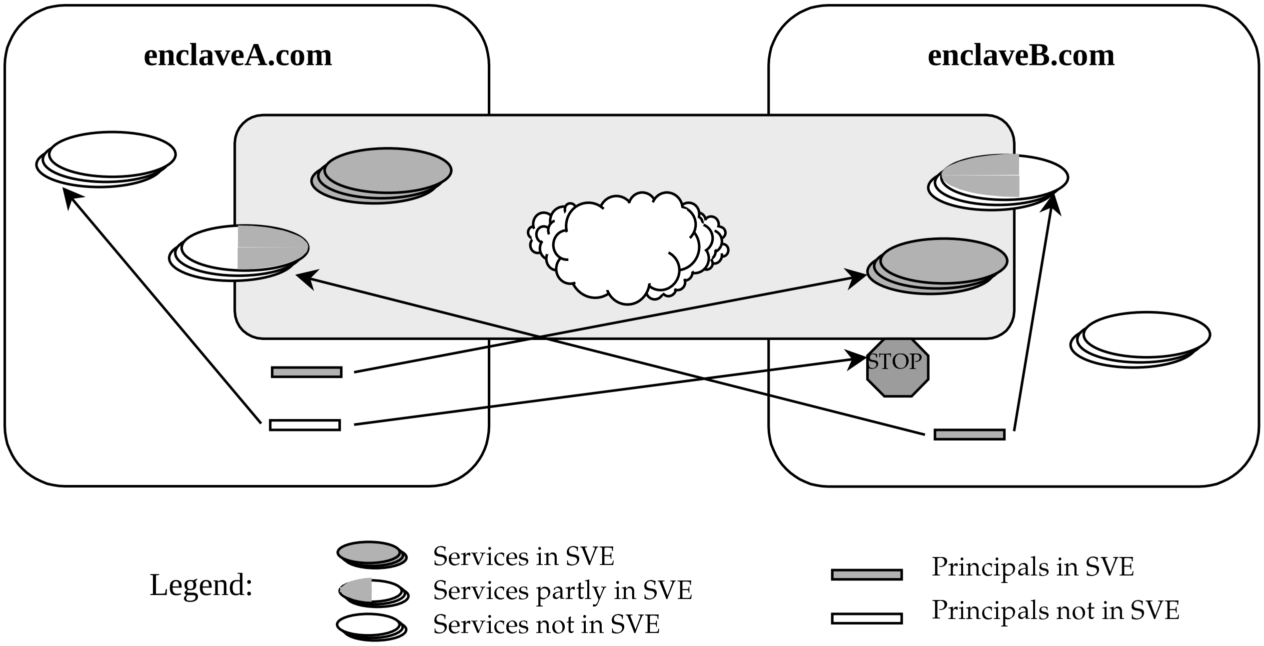

Figure 1: SVE concept of operation

unilaterally choose to withdraw its resources from a coalition at any time may be more willing to collaborate. Finally, by *distributed applications*, we mean applications built primarily on middleware infrastructures that support program and/or data object distribution. Examples include Java RMI, CORBA, Microsoft DCOM, and Enterprise JavaBeans. We may also include ordinary file system objects and resources accessible via a web server (e.g., HTML files).

Our approach was further bounded by the following constraints: the SVE infrastructure should be transparent to applications and based on commercially available operating systems and open networks. This forced us to work primarily in the realm of middleware, which, given our focus on distributed application technologies, was appropriate. We also emphasized the “Virtual” in “Secure Virtual Enclaves,” deciding against solutions that replicate resources and synchronize multiple copies. Though these approaches can improve the fault tolerance of a system and availability of its resources, they introduce undesirable system complexity. SVE resources remain within, and under the protection of, their local enclaves, while mechanisms are introduced to control accesses to these resources by external subjects. There is one caveat to our assumption about application transparency: we must be able to authenticate the identity of a requesting principal, so we insist that application traffic use some authentication mechanism.

Based on our goals and constraints, we designed an SVE to work in the following way: two or more organizations decide, through extra-technological means, to collaborate by sharing some of their local resources. The administrator of one of those organizations begins the technical and administrative process of naming and creating an SVE, and noting which other enclaves are trusted to join the collaboration. The creating enclave becomes the sole member of the SVE. The administrator then creates a resource access policy to identify the local resources to be shared and the local principals that will be authorized to access SVE resources, both local and foreign. The administrators of the remaining enclaves follow an administrative process to request to join the established SVE. After a join request has been submitted to the local SVE system, the remainder of the process is automatic—SVE system components of one enclave communicate with SVE system components of the other enclaves to establish the desired coalition membership. Finally, clients (acting for authorized principals) may begin to access SVE resources residing in any of the member enclaves.

Figure 1 shows the general SVE concept of operations. Two enclaves are participating in an (already formed) SVE. The SVE, shown in the shaded area in the center of the figure, is a projection of access rights for principals belonging to the SVE. Thus, the SVE identifies a distributed collection of related resources, along with the principals that are authorized to access those resources.

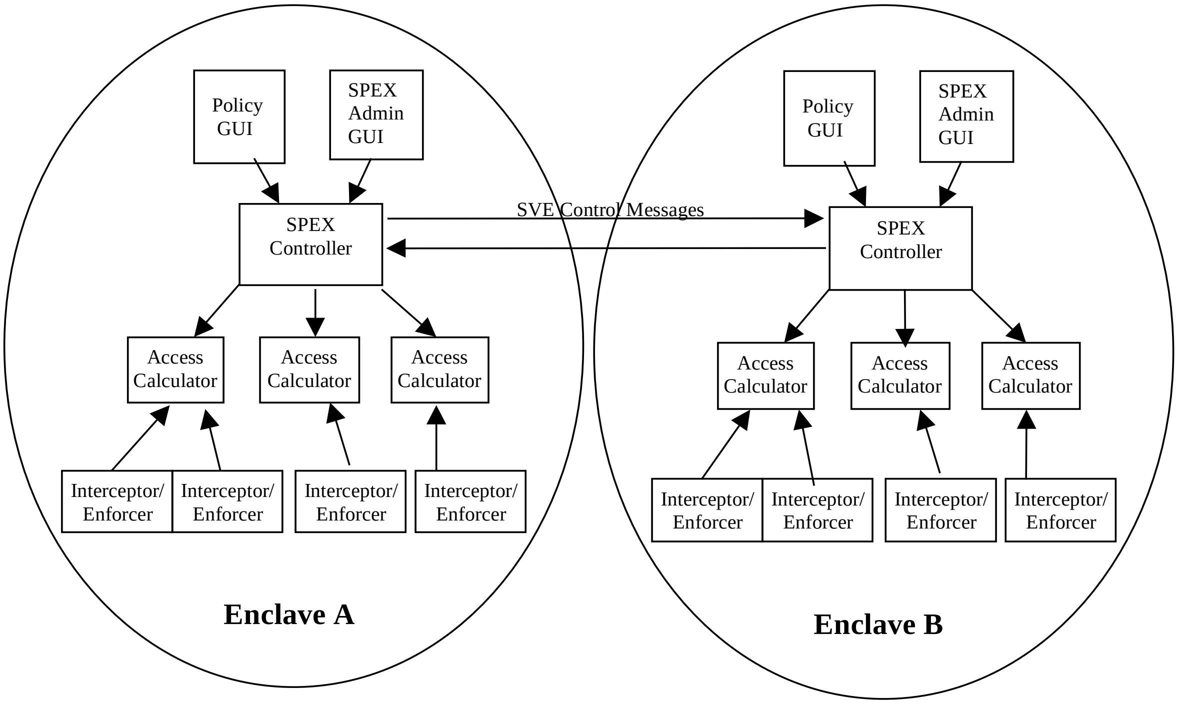

Figure 2: SVE component architecture

The resources and principals remain unchanged by the introduction of the SVE infrastructure.

2. Architecture

The SVE infrastructure consists of components that create, distribute, and enforce security policy, as shown in Figure 2. Each "egg" in the figure represents an enclave. The components in the upper halves of the eggs (policy GUI, administration GUI, and SPEX controller) are responsible for creating, maintaining, and distributing resource access policy, as well as administering SVE operations. The remaining components of the architecture interpret and enforce access policy.

The GUI components provide policy and configuration facilities to a local enclave administrator. The *Policy GUI* allows the administrator to develop and maintain access policy for local enclave resources. Through the policy tool, the administrator submits new policies or incremental updates. The *SVE Policy Exchange (SPEX) Controller* propagates policies within a local enclave to the SVE policy enforcement components, and to other SVE member enclaves. The SPEX controller also accepts SVE control commands from the *SPEX Administration GUI*, and participates in SVE control protocols (e.g., join, leave). The *Interceptor/Enforcers* capture client requests for server resources, query a local *Access Calculator* for an access decision, and enforce the decision by either allowing the request to proceed as usual, or dropping the request and returning an error message to the client. The access calculator encapsulates a local SVE resource access policy, and responds to access queries from local interceptor/enforcers. The SPEX controller provides asynchronous policy updates (either full or partial) to local access calculators.

Figure 3 shows a typical client-server application, with SVE interceptor/enforcers. The use of the SVE infrastructure is invisible to the application client, application server, and application developer. Interceptor/enforcers must, however, be installed between external clients and internal servers, via either gateways or server modifications. The remainder of the SVE infrastructure does not communicate with or affect the workings of the application.

2.1. Resource access policy

The administrator of an enclave creates and maintains the local resource access policy for the SVE by identifying the resources that may be shared and the principals that may participate. Since an enclave may choose to belong to multiple SVEs, there may be multiple local policies (one per SVE) in force at any given time. In fact, there may be several more policies lying dormant in

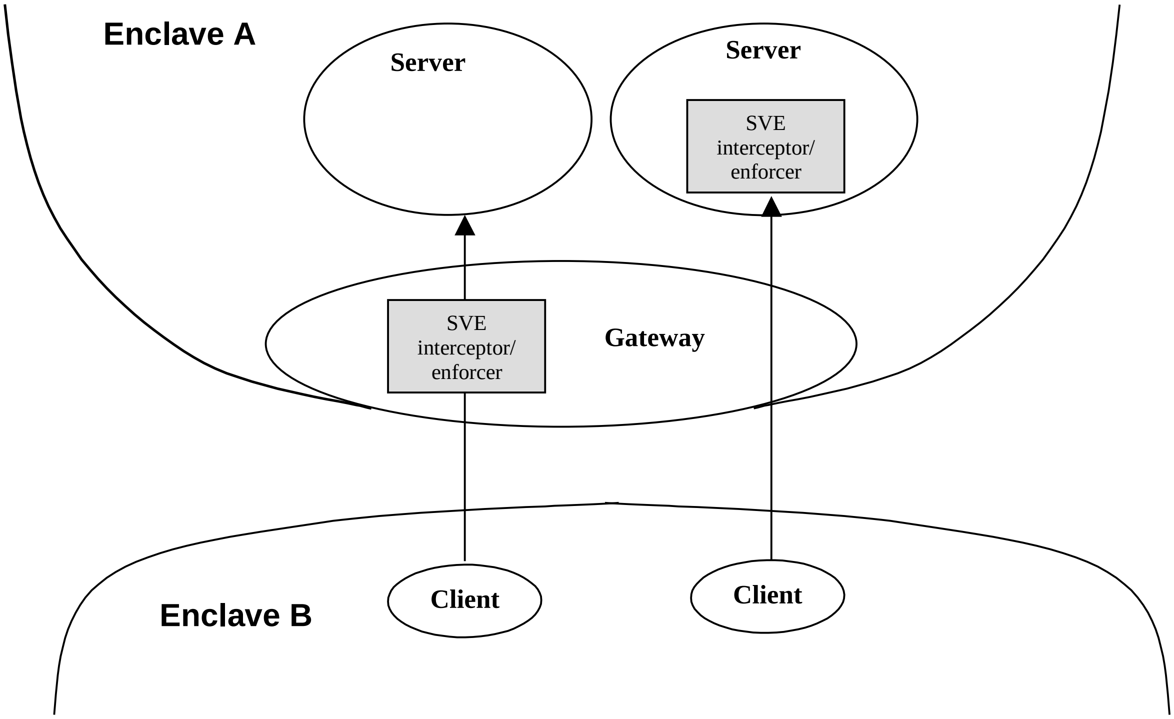

Figure 3: SVE client-server communication

case changing conditions (and levels of trust) compel the administrator to replace one of the active policies. In this section, we will describe a single SVE policy, local to the enclave in which it was created.

The SVE policy language uses concepts familiar from Domain and Type Enforcement (DTE) [5], which defines policy in terms of the access rights of equivalence classes of subjects to equivalence classes of objects. An object is a resource accessed by software. A subject is a software component that accesses resources on behalf of a principal. Principals are persons or persistent programs (such as servers). In DTE, objects are grouped into equivalence classes called *types*, and subjects are grouped into equivalence classes called *domains*.

SVE policies have four components: type definition rules, which map distributed objects into types; domain derivation rules, which map subjects into domains; a domain x type access matrix; and, potentially, a collection of additional access constraints. Figure 4 shows an example SVE resource access policy.

A type definition identifies a collection of resources that will be treated identically for access control purposes. For many object-oriented distributed systems, a resource is the pair (object interface, method). In systems that support distinguishable objects (e.g., Enterprise Java Beans), this could mean the more specific pair (named object, method). The example shows definitions for three types of resources: specification resources, source code resources, and financial resources. Thus, specification-related resources to be controlled include html files found in the `specs` directory at `eweb.com`, as well as the `view` method of the `SpecAdminI` interface. The “`_t`” appended to each type name is a mnemonic device, rather than a syntactic requirement.

Domain derivation rules uniquely identify principals on whose behalf a subject will be assigned to a particular domain. Domains are, essentially, role specifications. In the example, `frank` and `jane` (both of Acme, Inc.) are assigned to the engineering and accounting domains (roles) respectively. `sue` (of Toyco, Inc.) is assigned to the engineering domain. `frank` and `sue`, both engineers, will have identical access to resources as long as Acme and Toyco collaborate within an SVE. `jane` will also have access to SVE resources, but because she is an accountant, rather than an engineer, it is likely that her resource access permissions will be different from `frank`’s and `sue`’s. Thus, the semantics of “domain” within the context of Domain and Type Enforcement is very different from the “domain” addressed in network terminology. The example specifications show an email address (found within an X.509 certificate) prepended with a unique name for the user’s home enclave (e.g., Acme). This name identifies the SPEX controller that issued the domain derivation rule. The “`_d`” appended to

Type Definitions

specifications_t = https://eweb.com/specs/*,
    rmi://com.eweb.SpecAdminI/com.eweb.JavaSpec view(int)
source_code_t = rmi://com.eweb.sourceI/*
financials_t = dcom://123-456-789/321-654/1

Domain Derivation Rules (Principal Recognition Rules)

Acme!frank@acme.com = engineer_d
Acme!jane@acme.com = accountant_d
Toyco!sue@toyco.com = engineer_d

Access Matrix

engineer_d = specifications_t, source_code_t
accountant_d = financials_t+TimeInterval!900!1700!M!F

Figure 4: Example SVE resource access policy

each domain name is a mnemonic device, rather than a syntactic requirement.

The access matrix shows the types of resources to which subjects in a given domain are permitted access. In our example, engineers may access source code and specifications, while accountants may access financials.

Finally, an entry of the access matrix may be decorated with a *constraint*, identifying conditions that must be met before the subject (i.e., domain) can access the object (i.e., type). This allows us to extend our policies to address restrictions that may not be conveniently expressible in an ordinary access matrix. In the example, a time interval constraint is placed on accountants' access to financial data: subjects in the `accountant_d` domain may access `financials_t` type objects only during the hours between 9am and 5pm, Monday through Friday.

2.2. Policy distribution

A complete SVE policy, comprised of the four components described above, is necessary for an access control decision. An access calculator must, therefore, retain a complete policy and receive policy updates from its local SPEX controller. Some portions of the policy must also be shared among SVE member enclaves. In this section, we examine the need for inter-enclave policy distribution and discuss the role of the SVE Policy Exchange (SPEX) controller in the architecture.

An enclave defines its resource access policy, makes access control decisions, and enforces those decisions for each of its local resources, including resources it shares with other enclaves via an SVE. When a client from enclave **A** requests a resource on a server in enclave **B**, the principal responsible for the request must be identified by enclave **B** and the requestor mapped into an appropriate domain in order to determine whether the access should be permitted. **B** cannot recognize the principal behind the client without authentication and role information provided by **A**.

Information describing the requestor might be provided in a variety of ways. At one end of the spectrum, the client might present a credential at request time, containing the domain to which the requestor should be assigned. A certifying authority for **A** would have signed the credential. In this case, the client carries all of the data needed by **B**. At the other end of the spectrum, **A** might send all of its domain derivation rules to **B** in advance. When an **A** client makes a request, **B** would authenticate the identity of the responsible principal, then apply the appropriate rules to establish a domain for the requestor. In this case, the client carries almost none of the data needed by **B**. A range of hybrid solutions is possible, with some data delivered by the requesting client and some delivered in bulk, in advance.

There are advantages and disadvantages to any of these approaches. For the SVE architecture, we have chosen the approach in which an enclave sends domain derivation rules for its principals to other collaborating enclaves and only the identification and authentication data are carried by the client request. Because these rules allow enclaves to recognize "foreign" principals, we have renamed them *principal recognition rules*. Principal recognition rules are the only SVE policy component that must be exchanged among SVE member enclaves. All other policy components (resource-type mappings, access matrix, and constraints) remain within their local enclave. This provides the local administrator with the unilateral ability to control which of the local resources are shared – the administrator can change the resource-type mappings, the access matrix, or the constraints for the local resources without consulting other SVE member enclaves.

As a result, an enclave retains the right to authorize both local and foreign principals' access to local resources. In doing so, however, the enclave administrator does not authorize access directly to those principals. Rather, access is granted to a domain (role). Principal recognition rules (both local and foreign) provide the basis for determining membership in that domain. Thus, an enclave determines the roles that its principals assume within the coalition by providing appropriate principal recognition rules. This approach improves the scalability of the coalition-formation process, in that an organization need not have a priori knowledge of every foreign principal that might eventually participate. However, an enclave must trust its coalition partners to provide accurate and appropriate principal recognition rules. Note that the trust requirement for accurate domain placement does not diminish if clients carry domain designations or other forms of authorization in their credentials.

The SPEX component of the SVE architecture is responsible for distributing access policy to local access calculators, as well as communicating the local principal recognition rules to the SPEX controllers of other SVE members. Each enclave must have a SPEX controller. Since access calculators must be able to recognize principals from other enclaves, an aggregate of local and foreign principal recognition rules must be created by the local SPEX controller. The aggregate principal recognition rules, along with the local type definitions, access matrix, and constraints, are delivered to the access calculators by the SPEX controller. Local principal recognition rules are published to other SVE member enclaves via SPEX-to-SPEX communication when the enclave joins an SVE or when changes are made to the responsibilities of local personnel. When foreign enclaves update and publish their principal recognition rules, the SPEX controllers of SVE member enclaves deliver those updates to their local access calculators, without human administrative action.

2.3. Access calculation

The access calculators in the SVE architecture are responsible for deciding whether a given access is permissible. Each enclave contains one or more access calculators, though for performance reasons, we expect that an access calculator would be deployed on each host that supports SVE-sharable resources.

An access calculator presents two interfaces to components of the SVE infrastructure: an access decision interface to accept and respond to requests from interceptor/enforcers, and a policy update interface to the local SPEX controller. An interceptor/enforcer queries an access calculator for a policy decision, while the local SPEX controller pushes policy updates into the access calculator. The access policy is completely contained within the access calculator, so decision-making is accomplished strictly locally.

Access calculation is a four step process: domain derivation, type derivation, access matrix check, and constraint check. Domain derivation is accomplished using identity data, extracted from an authenticated credential, in combination with principal recognition rules, provided by the principal's "home" enclave. Type derivation is accomplished using resource request data, taken by the application interceptor, in combination with type derivation rules defined by the local SVE policy.

The premise underlying SVE access calculation is that each enclave cannot be expected to define access policy based on the individual identities of foreign principals. In order to grant access to foreign principals in a timely and scalable way, principals must be grouped into domains by their home enclaves. These groupings embody the roles of those individuals and the degree to which the individuals are trusted by their home enclaves.

In the SVE system, access authorization is granted equally to all principals (both local and foreign) represented by a domain, rather than to an individual principal. As in Figure 4, individuals acting in the `engineer_d` role will have access to the same SVE resources, regardless of their home enclaves or the location of the resources they access. In our prototype, these authorizations are represented by a domain-type access matrix. However, any policy representation that assigns access authorization to groups or roles could be used with the SVE system. If the access request is permissible according to the access matrix, then any constraints of the access are checked. The boolean result is returned to the interceptor/enforcer that initiated the query.

2.4. Request interception and policy enforcement

SVE interceptor/enforcers perform the tasks of capturing a request for a distributed system object, extracting data to identify both the target object and the requestor, forwarding this data to a local access calculator, and enforcing the access calculator's decision. One interceptor/enforcer may differ from another quite drastically, as their distributed application technologies can differ drastically from one another. We will discuss some of those implementation issues in Section 3, but will briefly identify two important classes of interceptor/enforcers, both of which are supported by the SVE architecture.

When server resources are to be protected, request interception and policy enforcement for distributed application technologies may be implemented via either a protocol gateway or a server-resident interceptor. Though the use of a gateway obviates the need for server modifications, information regarding a request is often incomplete "on the wire". In the case of distributed application technologies, this is often evident in that target identification (e.g., method call) is not resolved until the server receives the request. Gateways can sometimes be constructed to call out to a server for additional context data to circumvent this issue. When this is undesirable, when intra-enclave access control is necessary, or when client-server communication uses end-to-end encryption, server-resident policy enforcement may be the preferred approach. Note also that layered defenses can be built with combinations of gateway and server-resident interceptor/enforcers. The SVE infrastructure can support both types of interceptor/enforcer.

2.5. SVE administration

The administrator of an enclave is responsible not only for defining the enclave's resource access policies for each of the SVEs that the enclave joins, but also for performing local SVE administration tasks. In particular, the administrator must represent the enclave's trust relationships with foreign enclaves by identifying the list of enclaves with which it intends to collaborate in an SVE. This establishes an enclave-level trust policy that determines which foreign enclaves may have access to local resources. Both trust policy and resource access policy are managed by the enclave administrator via GUIs that communicate with the local SPEX controller. SVE administration commands for establishing trust policy, as well as creating, joining, and leaving an SVE, are initiated by an administrator via the GUI. Once initiated, these processes are carried out automatically by communicating SPEX controllers.

3. Implementation

The SVE infrastructure is primarily a Java-based architecture, tested on Sun Solaris, Windows NT 4.0, and Linux. Intra-enclave communication among SVE components is done via Java RMI. Inter-enclave communication is primarily accomplished using the group communication facilities provided by the Ensemble system [7], via the JavaGroups interface [3]. Except for the platform-specific binaries required by Ensemble, the SVE components are platform independent. Most of the components were engineered from scratch, using Java 1.1, with the considerable exception of the interceptor/enforcer code.

We chose to support distributed applications based on the following commonly used technologies: Java RMI, Microsoft's DCOM, and two web servers—Sun's Java Web Server, and Microsoft's IIS. In order to authenticate the identity of a requesting principal, application traffic must use an authentication mechanism. Since Java RMI and HTTP traffic can run over SSL, we use data from SSL-carried X.509 certificates to identify both web object requestors and Java RMI object requestors. The DCOM protocol does not currently run over SSL, but we can extract identity data from the Windows NT access token (created for the user at logon) which is carried with the DCOM request. If client identity cannot be established, (e.g., when a web or RMI application is not run over SSL, or a DCOM application allows an anonymous request) the SVE infrastructure will not permit access to resources it controls.

3.1. SVE Policy Exchange (SPEX) Controller

The SPEX controller is a multithreaded Java-based server that implements three interfaces for SVE administration and policy distribution: an administrative interface, which communicates with the administrative GUIs; an intra-enclave policy distribution interface; and an inter-enclave communication interface. The controller also implements repositories for resource access policies and SVE administrative data.

SVE administration (SVE policy updates and control requests) is handled through the SPEX controller's administrative interface. The administrative GUI communicates with the SPEX controller via Java RMI. At the administrator's request, the administrative GUI forwards SVE control requests (e.g., create a new SVE, join an existing SVE) to the SPEX controller. Changes in the enclave's state are pushed back to the administrative GUI for display to the administrator.

Intra-enclave policy distribution is handled through the policy service interface of the SPEX controller. Access calculators register via Java RMI as subscribers to the policy update mechanism. Policy changes originating at the policy GUI are accepted by the SPEX controller. Access calculators must enforce access policy for **all** of the SVEs to which the local enclave belongs. Thus, the SPEX controller must aggregate the local principal recognition rules, type mappings, access matrices, and constraints written for each SVE to which the local enclave belongs. Principal recognition rules received from foreign SVE member enclaves must also be aggregated with the local rules. This aggregate policy is propagated by either full or incremental updates to the subscribed calculators whenever the policy GUI gets an update from an administrator or when foreign members update their principal recognition rules.

Inter-enclave control messages and policy distribution are handled via the communication service interface of the SPEX controller. When an enclave attempts to join an existing SVE, its SPEX controller makes an RMI request to a liaison (a SPEX controller for an enclave already belonging to the SVE). The liaison launches a voting request by sending a message object through the JavaGroups interface to the Ensemble communication system. All of the current SVE members receive the voting request and consult their lists of trusted collaborators. The liaison tabulates the voting results – only a unanimous positive result will allow the prospective member to join. The result is returned to the prospective member as a response to the original RMI request. If accepted, the new member subsequently takes part in the group communication and submits its principal recognition rules for the SVE.

3.2. Interceptor/Enforcers

The SVE project implemented server-side (i.e., end-system-based, rather than gateway-based) interceptor/enforcers for Java RMI, Microsoft DCOM, and two web servers (Microsoft's IIS and Sun's Java Web Server). Both server-resident and gateway-based interception for CORBA requests were addressed by the Sigma project [13], a predecessor to the SVE project.

The implementation of interceptor/enforcers for a variety of distributed application technologies provided significant engineering challenges. Most of these technologies were not designed to allow for request filtering. Our interceptors are, therefore, highly implementation dependent, and vulnerable to version changes. Choosing to use gateway-based interception would have traded these problems for others. In particular, gateways must tolerate varying and evolving protocol implementations. For example, the developers of CORBA's GIOP/IIOP protocol did not take the need for boundary access control mechanisms into account when developing the original protocol specification. This decision created problems not only for firewall proxy development, but also for interoperability of different vendors' products when a gateway is involved. As mentioned in Section 2, identifying the requested resource before the request reaches the server is often exceedingly difficult as often only the server has sufficient contextual information to interpret the data carried by the protocol. We will now discuss some of the implementation challenges we faced in building server-resident interceptors for the various application technologies.

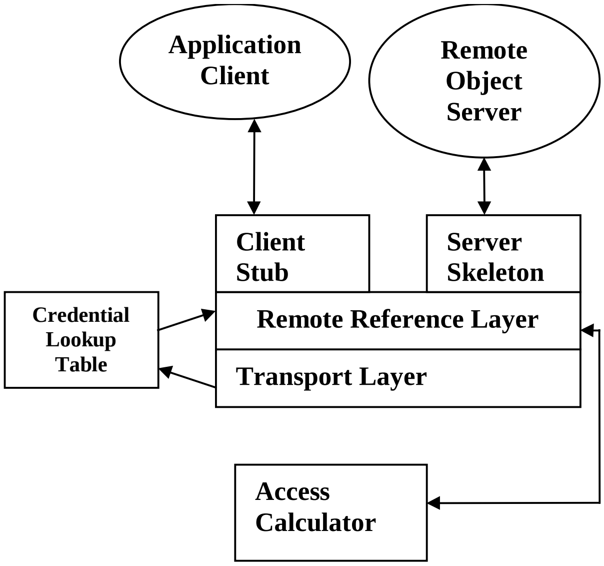

Figure 5: Java RMI interceptor

3.2.1. Java RMI. The first distributed application technology for which the SVE project implemented an interceptor/enforcer was Java RMI. Currently, Sun's RMI interface specification provides no defined application hook for intercepting client method invocation requests as they arrive at the RMI server. Consequently, providing a server-side interceptor for Java RMI required functional enhancements to Sun's RMI reference implementation.

The Sun specification describes the architecture of RMI in three layers. The topmost layer consists of the RMI stub and skeleton, which provide the client proxy and server dispatch functionality commonly found in distributed object models. The middle layer is designated as the remote reference layer and is responsible for providing specific remote invocation semantics, such as whether the remote server object will be a single object or part of a replicated object group. At the bottom is the transport layer, which is responsible for managing network connections and tracking remote server objects.

The RMI remote reference layer provides an ideal location for interceptor placement since it is considered part of the Java system API. In contrast, embedding an interceptor at the top layer would have required special skeletons to be generated for each application remote server implementation, while embedding an interceptor at the transport layer would not have provided adequate information about the invocation target. A consequence, however, of providing interception at the remote reference layer is that the interceptor is very specific to the RMI implementation. We chose to add interceptor capability specifically to Sun's RMI remote reference implementation for JDK 1.1. Figure 5 provides a high-level view of our design for RMI interception.

As shown in Figure 5, an additional complication to RMI interception was the inability to cleanly pass authentication credentials from the transport layer to the remote reference layer. The standard RMI interfaces in JDK 1.1 do not provide a means for passing credentials between layers. Thus, we were forced to implement a credential lookup table, which is shared by the transport and remote reference layers. We made an implementation-specific decision to use a thread identifier as the credential lookup key, since a single thread carries an invocation request through each layer in Sun's RMI implementation. Based on this design, the transport layer, upon receiving a client invocation request, will insert the associated credential into the lookup table using the current thread context identifier as the key. When the remote reference layer receives the same request, it will use its current thread context identifier as the lookup key for retrieving the credential associated with the request.

3.2.2. Microsoft DCOM. The second distributed application technology for which the SVE project implemented a interceptor/enforcer was Microsoft's DCOM. DCOM is the distributed specification for Microsoft's Component Object Model (COM) technology. A DCOM component can take any of three forms: a shared library (DLL), a binary executable (EXE), or a system service. A DCOM component is a collection of COM interfaces, each of which identifies methods, which are exported to applications. Using Windows NT 4.0, an access policy for DCOM components can be specified on a component-by-component basis. However, due to our goals of maximizing control and local autonomy, the SVE project required finer-grained constraints on interfaces and methods.

In order for a client to invoke a method on an interface offered by a remote DCOM component, the client must obtain an interface pointer to identify the requested object (i.e., instance of the component). Each interface of a DCOM component has a virtual table (v-table) data abstraction, which is a lookup table with pointers to method implementations. Pietrek, in [10], uses a custom DLL to modify DCOM v-tables and a kernel jump table to re-route DCOM method calls. The SVE system uses Pietrek's method of request interception. Other techniques have been developed which can be applied to the interception of DCOM requests. The method described by Balzer and Goldman in [4] replaces a portion of the application's assembly code, diverting the program to the interceptor. As we did not require the generality of the Balzer-Goldman approach, we chose to implement the simpler Pietrek method.

Once the interceptor has captured the DCOM request, it must request an access decision from an SVE access calculator. Assembly language code, such as that written for the kernel-based interceptor, cannot communicate directly with a Java-based server, such as an SVE access calculator. This problem forced us to build a bridge from the interceptor to the access calculator.

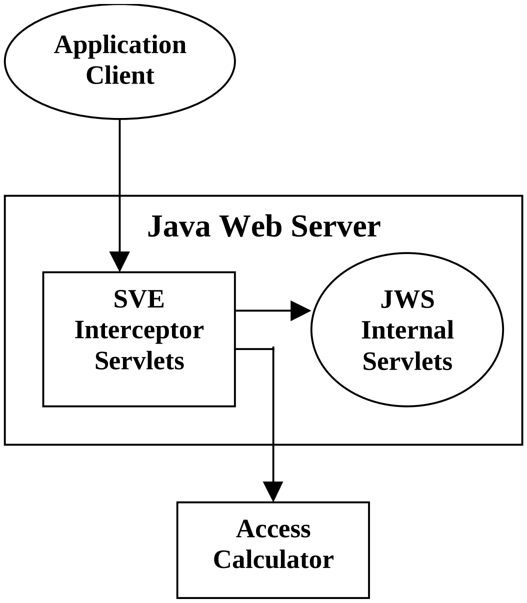

Figure 6: Java Web Server interceptor

The bridge is based on COM. A COM client and a COM server are inserted in the communication path between the assembly code-based interceptor and the Java-based access calculator. The C++-based COM client is called by the interceptor code. The COM client calls the Java-based COM server. Since Microsoft's Java Virtual Machine (JVM) does not currently support RMI, we used a collection of RMI classes developed by IBM to patch the COM server. This server can then communicate with the SVE access calculator.

Our interceptor can currently handle DCOM applications in the binary executable (EXE) form. The interceptor may be extensible to shared library (DLL)-based and system service-based DCOM applications.

3.2.3. Java Web Server. Web-based applications are another technology that SVE supports. We have developed two web server interceptor/enforcers, the first of which is Sun's Java Web Server. The Java Web Server (JWS) was designed with modular extensibility in mind and is built upon a server framework called the JavaServer Toolkit (JST). The JST allows developers to build network application services using the Java programming language. The types of services that can be built with the JST include established services such as HTTP and FTP, as well as application services that have yet to be created.

The JST supports the concept of a servlet, a service extension API that augments the capabilities of a particular service for customized application handling. Servlets run within a JST server as objects in support of a service and can be dynamically loaded on demand from any local or network source. The JWS uses servlets to manage all resources provided through its HTTP and HTTPS services. For example, the JWS uses a file servlet to provide its file-serving capabilities, a CGI servlet to execute any CGI-based scripts or programs, and an invoker servlet to execute custom application-specific servlets.

Providing an access control interceptor for the JWS was fairly straightforward. The JWS allows servlets to be chained together for the purpose of further augmenting service capabilities. Since the standard JWS utilizes internal servlets to manage all of its web resources, providing interception was merely a matter of inserting an interceptor servlet in front of each of the internal servlets. Thus, we were able to intercept all client requests for web resources managed by the JWS. Figure 6 provides a high-level view of our design for JWS interception.

3.2.4. Microsoft IIS Web Server. The final SVE interceptor/enforcer was developed for Microsoft's web server: the Internet Information Server (IIS). IIS uses Windows NT's Internet Server Application Programming Interface (ISAPI) as a customization interface. ISAPI is a server-side API with functionality similar to the Java servlet interface. An IIS *filter* conforms to the ISAPI, and is analogous to a Java servlet, used by the Java Web Server. Some filters are specially designated system filters, provided by Microsoft. Custom filters can be developed and added to the IIS server, as well. Both system and custom filters can be chained together to augment IIS services, with system filters being executed before any custom filters within the filter chain.

The SVE IIS interceptor is implemented within a custom filter, placed after the SSL system filter, but before other custom filters in the filter chain. The SVE interceptor makes use of an Active Server Pages (ASP) file, which should (according to the IIS documentation) allow the interceptor to forward the client request data to a local access calculator for an access decision.

Unfortunately, due to Microsoft's engineering problems with IIS 4, custom filters are unable to automatically execute ASP files before resource processing. Thus, the SVE filter is unable to automatically intercept resource requests. We worked around the problem by sacrificing transparency and manually modifying each of the IIS resources. During resource processing, the SVE filter first executes the SVE ASP file. Though this non-transparent approach would be unacceptable in an operational environment, the workaround was tolerable for experiments with SVE. We hope and expect that Microsoft will correct these problems with future releases of IIS.

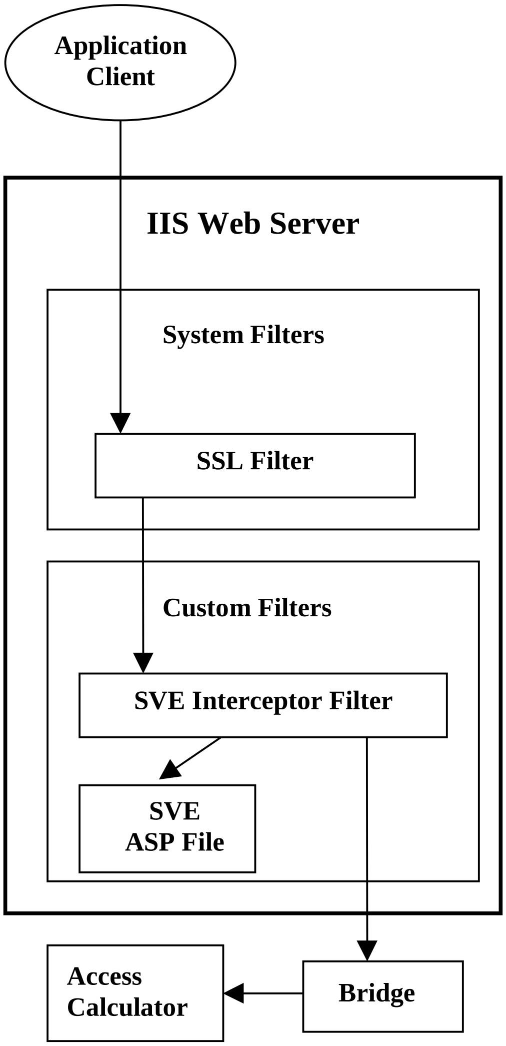

Figure 7: IIS Web Server interceptor

The lack of Microsoft JVM support for Java RMI once again forced us to develop a bridge between the ASP and the access calculator. We had hoped to use the same bridge that we built for the DCOM interceptor, but were unable to force the SVE interceptor filter to communicate directly with a COM client. We, therefore, prepended additional bridging components to enable the communication. Figure 7 provides a high-level view of IIS interception.

3.3. Communication security

To ensure that intra-enclave policy distribution takes place without danger of policy corruption or source spoofing, SVE Java RMI communication runs over SSL.

The SPEX controller authenticates the source of policy update commands and SVE administrative actions. Similarly, the access calculator ensures that the SPEX controller is, in fact, the source of a policy update. Communication between interceptor/enforcers and their access calculators has not yet been secured, due to project resource limits. To ensure that interceptor/enforcers cannot be fooled by imposters spoofing access decision results, these communications should be secured.

Inter-enclave communication has been secured to protect the integrity of policy data (in particular, principal recognition rules) and prevent source spoofing. The initial, RMI-based communication between a prospective enclave SVE member and its chosen liaison is secured using SSL. The remainder of inter-enclave SVE communication takes place using the group communication system. We had the option of using the communication security facilities provided by Ensemble (albeit not via JavaGroups); however, as we had earlier decided to insulate the SVE infrastructure from the choice of group communication facilities, we chose not to rely on Ensemble's security facilities. Instead, we chose to sign SVE message objects to provide source authentication and message data integrity.

To secure SVE communication, we used an implementation of the Java Cryptographic Extensions (JCE), produced by the Institute for Applied Information Processing and Communications (IAIK) from Graz, Austria. We used IAIK's Digital Signature Standard (DSS) implementation to ensure source authentication and data integrity for SVE messages carried over Ensemble. IAIK's iSaSiLk Java-based implementation of SSL was used to secure all of our RMI-based communication.

3.4. SVE policy

The implementation of SVE resource access policies facilitates both dynamism in policy updates and the use of a potentially broad range of policy models. An SVE policy is represented as a Java object that is created by an administrator, using the SVE policy GUI, stored and distributed via the SPEX controller, and, finally, lodged in an access calculator. The policy encapsulates both metadata (e.g., principal recognition rules, resource to type mappings, access matrix, and constraints) and rules (i.e., code) for interpreting that metadata. The policy object provides an update interface to allow the policy metadata to be changed, and an interface to respond to access queries from interceptor/enforcers. An active access policy resides in an access calculator. The calculator delegates any access decision requests to the policy object, which executes code to interpret its metadata and returns its decision to the calculator. The calculator responds to the interceptor/enforcer that initiated the query.

Because it is fully encapsulated, the policy can be passed within the SVE infrastructure and handled as an opaque object by most of the components. The access calculator provides an architectural placeholder in the SVE infrastructure to insulate interceptor/enforcers from details of policy-based decision making. The SPEX controller can push updated policy metadata into an active policy, or install a completely new policy in the access calculator without interrupting access decision requests from interceptor/enforcers. This approach ensures that the policy update process can be exceptionally dynamic.

The policy object's interface to an access calculator is quite simple: requestor and resource data are consumed and a boolean access decision is produced. Though the SVE policy language and GUIs reflect the OODTE influence, a policy **need not** be OODTE-based to be used within the SVE infrastructure. Any policy model implementation that respects the Java interface defined for the policy object could be substituted in the architecture. This offers us tremendous flexibility in selecting appropriate policy models. We hope to experiment with a variety of resource access policy models and use the SVE infrastructure to deliver and evaluate policies in working systems.

4. Discussion

The SVE project has developed a software infrastructure to enable collaborative distributed computing. During the analysis, design and implementation of this system, we identified several significant issues that impact the creation and use of secure virtual enclaves. Many of these issues (enclave autonomy, policy semantics, principal data representation and transmission, and trust policy) have both conceptual and implementation consequences. System design and implementation choices in each of these areas have significant conceptual repercussions on the relationships among coalition members, the protections offered by the system, and the complexity of establishing and maintaining a collaborative environment. Some of the issues, such as system implementation and performance and scalability, primarily impact the engineering of the SVE system and its fitness for use in particular collaborative environments. We will discuss each of these issues, identifying our results, limitations of the SVE approach, lessons learned, and some areas that deserve further attention.

4.1. Enclave autonomy

In this section, we describe how the SVE system enables enclave autonomy, and discuss security issues that arise from that autonomy. It is essential to facilitate as much organizational autonomy as possible because an infrastructure that required mutually suspicious organizations to cede significant control to potential competitors or adversaries would not encourage collaboration. The SVE infrastructure offers enclaves a great deal of autonomy in controlling access to their local resources.

The SVE infrastructure supports enclave autonomy within a coalition in two ways: (1) by ensuring that most resource access policy components are used only within the local enclave, and (2) by enabling an enclave to unilaterally withdraw from an SVE at any time.

An SVE member enclave retains full control over local resource access policy. Specifically, resource to type mappings, access matrices, and constraints are never propagated among enclaves. In the event that an enclave discovers a collaboration partner to be untrustworthy, the enclave may respond immediately by modifying any of these local policy components and updating its access calculators. This permits an enclave to unilaterally restrict access to its local resources by external entities. Alternatively, the enclave may withdraw from the SVE altogether, effectively removing all of the SVE members' principal recognition rules from the policy enforced by the local access calculator. This immediately prevents any principals in those foreign enclaves from accessing local resources. Leaving the SVE is a more drastic step as it denies local resource access to all of the SVE member enclaves.

As with any security mechanism, the protection offered by SVE policy enforcement cannot extend beyond the system itself. In particular, the design of the SVE infrastructure does not address the following two issues, both of which arise due to the autonomous operations of the collaborating enclaves: (1) despite an enclave's request to leave an SVE (or principal recognition rule update), its local principals may continue to access foreign SVE resources if other SVE members fail to update their access policies in a timely manner; and (2) a trusted SVE member enclave may inadvertently or intentionally share a copy of a resource with a non-SVE member.

In the first case, an enclave depends on other SVE members to correctly manage principal recognition rules. These rules are the only policy elements shared among SVE members, but they require careful handling by each of the SVE member enclaves to ensure that principals are granted appropriate access authorizations. For example, if principal "Alice" is transferred by her employer, enclave **A**, to a new position and no longer requires access to the "Alpha Project" SVE, enclave **A** depends on the other SVE members to remove Alice's principal recognition rules promptly. Alice may, otherwise, have continued access to Alpha Project resources held by foreign enclaves. We assume that **A** removed Alice's principal recognition rules promptly, so that she doesn't have access to the local Alpha Project resources. This problem also arises if enclave **A** leaves the Alpha Project SVE, due to new concerns about the trustworthiness of the other Alpha Project members. Alice may inadvertently continue to access Alpha Project resources held by foreign enclaves if **A**'s principal recognition rules are not purged from foreign Alpha Project members' active policies. This places Alice's, and, thus, **A**'s integrity at risk from compromised foreign resources.

In the second case, when a trusted collaboration partner shares resources (intentionally or unintentionally) with non-SVE members, we can't know which resources have been compromised, or how widely they may have been circulated. The compromise of an SVE member makes our local resources conveniently available to an attacker. We must, therefore, trust that our partners are not only honest and cautious, but also savvy about protecting their network infrastructures. These issues are not unique to the SVE system, but, rather, impact coalitions in general and are likely to be significant considerations in planning collaborative efforts.

4.2. Transmission of principal data

To enable principals to access resources in foreign enclaves, while satisfying the requirements for application-level transparency, requestor data must be transmitted to the resource owner's enclave. As discussed in Section 2, approaches to transmitting this data range from annotating principal certificates to supplying a list of all local principals' data. In this section, we discuss the advantages and disadvantages of the SVE approach.

The SVE approach to transmitting requestor's data to a resource owner's enclave is to propagate a collection of principal recognition rules in bulk before a request is made. This bulk propagation approach has the advantage that modifying a role specification (i.e., changing a principal recognition rule) is less expensive than reissuing a certificate. Certificates are usually issued off-line, so producing a new certificate can take a relatively long time. Furthermore, with each authorization change, the old certificate must be revoked and a certificate revocation list updated. While we can't avoid dealing with certificate revocation with respect to identity certificates, the use of authorization certificates might create even more significant certificate revocation issues because authorizations are likely to change more frequently than identities.

The use of bulk principal recognition rules has its disadvantages, however. When enclave **A** reorganizes its employees and changes their organizational roles, it must issue updates of its principal recognition rules to ensure that employees have access to the correct SVEs and that their subjects are mapped into the correct domains.

Whenever enclave **A** makes organizational changes, all other enclaves in the SVEs in which **A** is a member must update their systems to handle those changes. This misplaces the burden of accommodating local changes onto foreign entities.

It is possible that a hybrid approach to authorization transmission would be valuable: some role information is contained in an authorization certificate, a user manages multiple such certificates, and additional (more volatile) data is expressed in principal recognition rules.

4.3. Policy semantics

Policy semantics is a thorny issue when role/authorization data crosses enclave borders. In this section, we describe problems that may arise with regard to policy semantics, and their impact on the development of security policies for coalitions.

A major problem with policy semantics is the different interpretations of common entities across multiple enclaves. For example, though enclave **A** may designate a principal for the `manager` role, the semantics of `manager` are known only within **A**. `manager` might mean line management (e.g., matching staff to projects, reviewing salaries), or it might mean project management (e.g., project tasking, project budgets). In either case, the semantics of the `manager` designation are established within the **A** context.

Suppose, for discussion, that by `manager`, enclave **A** means project management. When the `manager` designation crosses into enclave **B**, either in an authorization certificate, or as a target domain in an SVE principal recognition rule, the critical context data is lost. At least three possible problems might arise: (1) If enclave **B** has no `manager` role, then it will not authorize resource access, as the principal designation is unknown in **B.** The foreign manager, therefore, will be denied access. (2) If enclave **B** locally defines a `manager` role with the semantics of line management, then the foreign manager may inadvertently gain access to some of **B**'s salary data. (3) If enclave **B** defines a project management role, but instead uses the spelling `project-manager`, then the local project managers may have access to different resources than the foreign manager, despite the intention to grant them identical resource access.

One might expect that this problem could be handled by treating this as a namespace issue, however, that approach is insufficient, as marking the `manager` role with an enclave qualification gives us something like "`enclaveA/manager`". This designation does not provide **B** with the information necessary to interpret the role in the **B** context. Nothing short of establishing the semantics of roles in advance of their use will completely solve this problem.

In the SVE project, we assumed that the domain names to be used in resource access policies were established by the SVE creator and agreed upon via extra-technological means by other SVE members in advance of any resource sharing. A system in which **A**'s and **B**'s roles could differ would require some translation mechanism, either directly between **A** and **B**, or into some agreed-upon intermediate specification. Translation mechanisms might be constructed by wrapping local role data with filters. Overall, the need to establish a common understanding of security policy semantics among collaborating organizations will prove crucial to the effective use of coalition enabling technologies.

4.4. Trust

Trust is the foundation of any collaboration. In this section, we consider three trust issues arising in an SVE context, and how the SVE infrastructure handles (or might be extended to handle) those issues.

The first issue pertains to the granularity of trust expressible within the system. In the SVE system, a principal's identification credentials are delivered to an interceptor/enforcer when an access request is made. The interceptor/enforcer must authenticate the credentials (e.g., check certificate signatures) and determine whether it trusts the signer before extracting the identification data and forwarding it to an access calculator. We either trust that the certificate accurately identifies the requestor or we don't. If we had used authorization certificates, we might have applied a trust designation to each authorization contained in the certificate. The certificate signer may or may not be trusted by the target enclave to certify each of the authorizations. A finer granularity of trust could be distinguished within an SVE policy, which could then influence access calculation.

The second issue pertains to the uniformity of trust relationships within an SVE. When joining an SVE, an administrator must specify to the local SPEX server the list of other enclaves that are trusted to join this SVE. These enclaves may share local SVE resources. The SVE system treats each of these member enclaves identically—each member is equally trusted. To create distinct trust relationships among enclaves, an administrator would form multiple SVEs with different resource access policies for each SVE. Extending the SVE trust model to better support asymmetric trust relationships could make the system applicable to a broader set of coalition environments.

The third issue pertains to the degree of symmetry required in SVE member relationships. SVE member enclaves are peers. For example, to admit a prospective member to the SVE, each current member must vote, and the votes must be unanimously positive to allow the prospective member to join. It would be worthwhile to consider non-peer relationships within the SVE context, to allow the concepts of "subordinate" or "subcontractor" to be represented. In an asymmetric relationship, some enclaves are primary SVE members, while, for example, other enclaves participate in the SVE because they manage resources at the direction of primary members. Secondary members that supply services to other enclaves may not require a full vote in this scheme. We have not determined how such asymmetries might be expressed within the SVE context.

4.5. Implementation

In this section, we briefly review our analysis of SVE implementation issues and identify alternative implementation options we might choose today, if the SVE project were just beginning. As mentioned in Section 3, most of the SVE infrastructure was implemented in Java 1.1, using RMI for intra-enclave communication and the Ensemble group communication system for inter-enclave communication. Though both Java and JavaGroups/Ensemble proved to be good choices for implementation, upcoming Java facilities and development tools would have eased our development task. For example, Sun Labs has recently developed a reference implementation of Java reliable multicasting (JRM) [12]. This might allow us to eliminate some of the platform-dependent multicasting facilities provided by Ensemble. Sun Labs plans to add the JRM API into future versions of the Java Development Kit (JDK).

By far, our most significant implementation problems arose during the development of the interceptor/enforcers. Neither Java RMI nor DCOM were built to allow the modular addition of security mechanisms. The problem was significantly compounded for DCOM due to its relative complexity and the lack of available internal specifications. Fortunately, the RMI interceptor was not quite as problematic, as RMI is simple and its specification is well-documented. However, both interceptors are implementation dependent and vulnerable to version changes.

The web servers were clearly designed to allow additional functions to be added. In the case of the Java Web Server, the servlet concept allowed us a simple means of encapsulating and installing SVE interception code. Microsoft's IIS web server offered a similar approach to extensibility via its filter concept. Unfortunately, several documented features of IIS did not work as specified. As a result, an administrator must manually prepare IIS resources before they can be controlled by the SVE system. We hope that Microsoft will correct the problems with IIS in a future release.

4.6. Performance and scalability

Project resources did not permit us to perform a quantitative performance evaluation of our system. We can, however, discuss our qualitative observations from the perspectives of an administrator and of a user client.

4.6.1. Performance. We developed the initial version of the system without any communication security for the SVE infrastructure. This initial version did require client authentication, however, to allow us to test our access decision-making and enforcement. We built a simple distributed application to test and demonstrate the SVE system, using Sun's JDK versions 1.1.6 and higher, with the Just in Time (JIT) compiler enabled. The application included two data repositories, one contained in a Microsoft access database and one contained in flat files. We built DCOM and RMI clients, and used web browsers to access the resources. We used a variety of machines ranging from a 166 MHz Pentium Pro with 32 MB RAM to a SPARC Ultra-1 with 128 MB RAM, up to a 450 MHz Pentium 2 with 128 MB RAM. All of these machines were connected via our local Ethernet.

When run with communication security turned on (SSL for RMI and the web browsers, and the Microsoft proprietary protocol for DCOM), users noticed a significant performance impact. When accessing Java-based resources via RMI over SSL, the delay between a request and the display of resource data was less than 2 or 3 seconds. The bridges we built to permit communication between the Microsoft interceptors and the access calculators degraded performance for these types of requests. In our system, access requests for Microsoft resources were significantly slower (perhaps on the order of 7 or 10 seconds) than requests for the Java-based resources. By inserting dummy access decisions into our enforcement code, we established that the bridges, rather than the interceptors, were at fault.

From an administrator's perspective, SVE control operations appeared to have reasonable performance (i.e., under 1 second for an SVE join operation), without communication security. Once we enabled message signing and verification, the performance degraded noticeably. An administrator might wait up to 5 seconds before receiving confirmation of a successful join operation. In summary, it appears that both intra-enclave and inter-enclave communication security introduce significant obstacles to attractive performance of the SVE system.

4.6.2. Scalability. Our system was tested with only a few SVEs (5 or fewer), and only a few member enclaves of each SVE (3 or fewer). The small size of the experiment was due primarily to the administrative overhead of developing security policies. Scalability was not, therefore, addressed by experiment. Roughly, the cost in administrative overhead and communication of joining and participating in two SVEs is twice the cost of joining only one. Thus, the scalability of the system largely hinges on how well we can support the growth of a single SVE.

SVEs grow as new member enclaves issue join requests, and are maintained as existing members issue policy update requests. The join operation has the highest messaging requirement of any of the SVE operations, as voting requests must be conveyed to each member, ballots collected from each member, and result notification sent to each member. All of these underlying operations are accomplished using Ensemble-based communication. Therefore, the scalability of inter-enclave SVE communication is heavily dependent on the scalability of the underlying Ensemble communication system, whose performance is addressed in [7].

5. Related work

Several research projects have addressed problems related to collaborative computing. Systems such as Ensemble [7], Rampart [11], Transis [1] and Enclaves [6] provide secure group communications, allowing collaborating entities to maintain privacy and message integrity in their communications. The SVE system, however, focuses on supporting policy-driven access controls for distributed object systems.

There is also research in the area of security policies for group communication. In [15], Srisuresh and Sanchez describe policy-based routing for IP security. Requirements are described for intermediate and end nodes to support security policies for packets crossing enclave boundaries. At this level, policies refer to packet forwarding rules or communication security mechanisms and their parameters. The Dynamic Cryptographic Context Management (DCCM) system [2] provides dynamically changeable mappings from low-level policy abstractions onto cryptographic mechanisms for secure group communications. DCCM offers a mechanism for inter-enclave policy negotiation. In this case, policies refer to cryptographic context specifications (e.g., ipsec 3des-cbc encryption, using sha-1 integrity verification).

Other work that has dealt with access controls for distributed objects include the Sigma project [13], which investigated the integration of security technologies into CORBA-based distributed computing environments. The Sigma project built prototypes of both gateway and server-resident (ORB plug-in) interceptor/enforcers for CORBA requests and developed the object-oriented version (OODTE) [14] of the Domain and Type Enforcement (DTE) [5] policy specification language. Sigma results were focused on policy definition (via OODTE and other policy specification languages) and request interception. The SVE project extended these results by introducing an infrastructure to support shared policy elements for collaborating organizations. The Multi-Protocol Gateway (MPOG) [9] extended the CORBA ORB gateway to enforce access policy for both CORBA and Java RM requests. We hope to extend the SVE system to provide dynamic policy updates for the MPOG to enable it to consistently enforce SVE resource access policies.

Each of the types of middleware resources (e.g., web resources, RMI resources, DCOM resources) for which the SVE infrastructure provides access controls has its own model of access policy specification and enforcement. For example, the default mechanism for specifying access policy for resources controlled by the Java Web Server is access control lists. Currently, there is no mechanism for controlling RMI access to distributed Java resources. In the future, however, the Java Authentication and Authorization Service (JAAS) [8] will be used to control this type of access. Access to DCOM applications can be controlled by Windows NT, but the finest granularity of control is at the per-application level (e.g., “launch,” “configure”). Each of these middleware systems specializes in controlling access to its own distributed resources, rather than providing uniform access policy enforcement for multiple types of distributed object resources. None of these policy enforcement systems support notions of collaborative computing among multiple organizations. The SVE system focuses on supporting collaborative computing through uniform access controls on various distributed object resources.

The SVE system was engineered using many previously developed technologies. As mentioned in Section 3.2.2, the SVE approach to DCOM interception applied a method described by Pietrek in [10]. A more general approach to mediating calls into a Windows NT API is described by Balzer and Goldman in [4]. As we did not require the generality of Balzer/Goldman, we opted for the relative simplicity of the Pietrek method.

The SVE infrastructure made use of the Ensemble group communication system to provide reliable communication for SVE member enclaves. The JavaGroups toolkit [3], developed by Ban at Cornell, provided us with a Java-based interface to the Ensemble system. Protocols for SVE management make calls through JavaGroups to communicate over Ensemble.

6. Conclusions

Our primary goal was to study software mechanisms to support coalitions in collaborative computing efforts. Because coalition partners may have only limited trust in one another, a coalition support system must provide both the means for careful control of coalition partners' access to local resources and the ability to dynamically change those controls as trust relationships evolve. In addition, the need to share current application resources required that our system solutions be transparent to applications and based on commercially available operating systems and open networks.

To meet coalition requirements for a collaborative computing environment, we designed and implemented a prototype security infrastructure. The prototype addressed the problem of limited trust relationships by providing significant resource access policy definition and enforcement autonomy to individual member enclaves. To support the changing nature of a collaborative arrangement, we provided a dynamic policy update mechanism.

The SVE infrastructure allows multiple organizations to share their distributed application resources, while retaining organizational autonomy over local resources. While we demonstrated an approach to fine-grained access control for secure collaborative computing, we also identified significant problems that remain to be solved, particularly in the area of policy development, before such collaboration will be convenient. The SVE infrastructure offers a platform and conceptual basis for further exploration of these problems and experimentation with new solutions.

Acknowledgements

We would like to thank David Chieu for his patience and persistence in implementing the DCOM and Microsoft IIS interceptors. Yongjian Wang produced the first versions of our administrative GUIs, giving us a better understanding of SVE administrative issues. Debbie Sturdevant produced several of the diagrams for this paper. Bob Rice and Dennis Hollingworth reviewed drafts of the paper, and their suggestions have helped us improve the presentation. This work was supported under DARPA contract #F30602-97-C-0269.

References

[1] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Transis: A communication sub-system for high availability. In *Proceedings of the 22nd International Symposium on Fault-Tolerant Computing*, pages 76-84, July 1992.

[2] D. M. Balenson, D. K. Branstad, P. Dinsmore, M. Heyman, and C. Scace. Dynamic cryptographic context management (DCCM) report 3: Cryptographic context negotiation protocol. Technical Report TISR #0757, TIS Labs at Network Associates, Inc., February 1999.

[3] B. Ban. Design and implementation of a reliable group communication toolkit for Java. http://www.cs.cornell.edu/home/bba/papers.html.

[4] R. Balzer and N. Goldman. Mediating connectors. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems Workshop (ICDCS '99)*, pages 73-77, Austin, TX, May 1999.

[5] L. Badger, D. Sterne, D. Sherman, K. Walker, and S. Haghighat. Practical domain and type enforcement for UNIX. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, pages 66-77, Oakland, CA, May 1995.

[6] L. Gong. Enclaves: Enabling secure collaboration over the Internet. In *Proceedings of the Sixth USENIX Unix and Network Security Symposium*, pages 149-159, San Jose, CA, July 1996.

[7] M. Hayden. The Ensemble System. Ph.D. dissertation. Cornell University, Ithica, New York. Available as technical report TR98-1662, 1998.

[8] C. Lai, L. Gong, L. Koved, A. Nadalin, and R. Schemers. User authentication and authorization in the Java platform. To appear in *Proceedings of the 15th Annual Computer Security Applications Conference*, Phoenix, AZ, December 1999.

[9] G. Lamperillo. Architecture and concepts of the MPOG. Technical Report NAI #0768, NAI Labs at Network Associates, Inc., June 1999.

[10] M. Pietrek. Learn system-level win 32 coding Techniques by writing an API spy program. In *Microsoft Systems Journal*, vol. 9, no. 12, pages 17-44, 1994.

[11] M. Reiter. Secure agreement protocols: Reliable and atomic group multicast in Rampart. *In Proceedings of the Second ACM Conference on Computer and Communication Security*, pages 68-80, Fairfax VA, November 1994.

[12] P. Rosenzweig, M. Kadansky, and S. Hanna. The Java reliable multicast service: a reliable multicast library. Technical Report #TR-98-68, Sun Microsystems, Inc., September 1998.

[13] E. J. Sebes and T. C. Vickers Benzel. Sigma: Security for distributed object interoperability between trusted and untrusted systems. In *Proceedings of the 12th Computer Security Applications Conference (ACSAC '96*), pages 158-168, San Diego, CA, December 1996.

[14] D. Sterne, G. Tally, D. McDonnell, P. Pasturel, D. Sames, D. Sherman, and E.J. Sebes. Scalable access control for distributed object systems. In *Proceedings of the 8th USENIX Security Symposium*, August 1999.

[15] P. Srisuresh and L. A. Sanchez. Policy framework for IP security. *Internet Draft* draft-ietf-ipsec-policy-framework-00.txt, February 1999.