Proc. ICDCS 2004, in press

DisCo: Middleware for Securely Deploying Decomposable Services in Partly Trusted Environments

Eric Freudenthal and Vijay Karamcheti
Courant Institute of Mathematical Sciences, New York University
{*freudent, vijayk*}*@cs.nyu.edu*

Abstract

The DisCo middleware infrastructure facilitates the construction and deployment of decomposable applications for environments with dynamic network connectivity properties and unstable trust relationships spanning multiple administrative domains. Consumers of these services, who are mutually anonymous, must be able to discover, securely acquire the code for, and install service components over the network with only minimal a priori knowledge of their locations. Once installed, these components must be able to interoperate securely and reliably across the network.

Solutions exist that address individual challenges posed by such an environment, but they rely upon mutually incompatible authorization models that are frequently insufficiently expressive. The primary contributions of DisCo are (1) a middleware toolkit for constructing such applications, (2) a unifying authorization abstraction, and (3) a realization of this authorization well suited for expressing partial trust relationships typical of such environments. This paper is primarily about the first two of these contributions, [7] presents the third.

1. Introduction

Increasingly, distributed applications are being called upon to execute in dynamic network environments spanning multiple administrative domains, and in situations where the principals involved are subject to changing trust relationships. This trend is a consequence of many factors, including an increase in user mobility and the growing popularity of “out-sourced” applications such as those advocated by the web services standardization efforts.

Such applications can involve dynamically deployed code published by one organization, executed on behalf of a user (or automated agent) in a second organization, on a computer administered by a third. Such applications raise security concerns inadequately addressed by current infrastructure.

In order to address the challenges of security and protection of system and application integrity, appropriate access control mechanisms must be implemented at all levels: hosts must be protected against corruption by “rogue” programs, programs must be protected from rogue hosts, and communication channels must be secure and authorized.

Components and systems are available that address various portions of this problem space. However, the lack of a coherent security infrastructure with a unified access-control model increases the complexity of constructing and deploying security-conscious applications and systems. DisCo is a security-aware middleware toolkit for constructing modular distributed applications that addresses these challenges through the pervasive use of a unifying authorization abstraction.

1.1. Security Challenges Addressed by DisCo

DisCo provides an API for dynamically deployable applications with the following features:

1. A modular *replaceable* abstraction for access-control that includes support for partial trust relationships.
2. A communication substrate and dynamically configurable execution sandboxes that use this abstraction.
3. Direct representation of continuous authorization relationships including mechanisms that report when authorization characteristics change.

These features address shortcomings in existing middleware, which often provide disjoint and incomplete solutions to the challenges of security-aware distributed applications. For example, TLS[5] provides an encrypted communication channel merged with an X.509 authenticator,[1] but there is no mechanism available for using it with a different authorization system. X.509 authentication is not well suited for access control problems that require transitive authorization between mutually-anonymous parties. Other combinations of secure transport and authorization schema have been recently proposed, however these solutions are not modular and therefore also only offer fixed-point solutions with no access control system accepted as being appropriate for all applications.

In the absence of a universal authorization system, DisCo utilizes a generic authorization schema for all access control decisions. While DisCo provides a realization of this

1 X.509 only provides authentication of identity. These authenticated identities are commonly used as a subjects for boolean ACL authorization.

schema using the dRBAC trust-management system, alternative realizations based on other authorization systems can be easily substituted. This generic authorization schema is used pervasively throughout DisCo including for authorizing component-host relationships.

Typical solutions that address the same problem as the second feature above have relied on sandbox abstractions in languages, such as Java, which statically associate particular code publishers with pre-set configurations of a selectively-permeable membrane isolating system resources from an object's execution environment. Our work extends this model to include configuration and maintenance of membrane permeability based on characteristics of dynamic *partial trust* authorization relationships between agents (who dynamically deploy application *components*), code publishers, and hosts.

Finally, existing access control solutions were developed for authorization of atomic transactions that occur in some instant of time. However, this transaction model is inappropriate for modular distributed components that engage in sustained security-sensitive relationships. If access permissions are only evaluated at the time a trust relationship is established, then the system suffers from time-of-check-to-time-of-use (TOCTTOU) [9] vulnerabilities. We observe that while the timely enforcement of reduced access rights may be critical to maintain system integrity, these permission changes occur infrequently and asynchronously with accesses; DisCo's authorization framework decouples the updating of access permissions from their use, thereby providing an efficient yet reliable representation of "continuous trust" well suited to sustained relationships.

2. DisCo Overview

In this section, we describe the structure of a typical application suited for the DisCo middleware and a notional application that illustrates the challenges DisCo addresses.

2.1. Application Model

DisCo applications are assemblages of modular software *components* that may be deployed on multiple network-connected hosts in response to requests from local or remote users or even automated deployment controllers (that we refer to as "agents") acting as users . As with programs running in conventional network-centric environments (e.g., using Java's sandboxes and RMI), once deployed, components are provided controlled access to system resources and other network-connected components.

DisCo's execution environments (called *execution containers*) and communication substrate (called *Switchboard*) extend these controlled access models to "mutually anonymous" deployments where neither the deployed component nor hosts are pre-configured with appropriate access restrictions. Instead, infrastructure is provided to allow access-control configurations to be determined dynamically.

The following section describes an example modular application and how it might benefit from DisCo.

2.2. TravelAnywhere Example

TravelAnywhere is a fictional Internet travel service intended for use by corporate travel departments, and illustrates a typical DisCo application. TravelAnywhere is comprised of three primary components: (1) a *query* subsystem for searching and choosing among available flights, (2) a *booking* and *billing* subsystem that allows users to purchase tickets, and (3) a *user interface* front-end component. These components have the following security and integrity characteristics:

- The query subsystem is compute-intensive, and given TravelAnywhere's interest in providing fast responses to its user queries, can benefit from deployment in proximity to users of the service. Corruption of this facility presents a liability to customers, and therefore TravelAnywhere permits the query component for a particular corporate entity C, to only be deployed on hosts trusted by C.
- The booking subsystem is responsible for both financial and reservation transactions. Incorrect client authorization or accounting is a risk for TravelAnywhere. Therefore, TravelAnywhere is only willing to have the booking component execute on hosts it authorizes.
- In order to be responsive to user needs (and adaptive to changing component availability), the user interface component prefers to be located in proximity to the user on hosts trusted by the user.

Ideally, TravelAnywhere components can be dynamically deployed onto computational hosts in a generic manner – where neither the components nor the hosts need to be pre-configured for each other, allowing computational resources to be negotiated as a commodity. However, such generic deployments must ensure that the integrity constraints of the components are preserved despite (1) the initial mutual anonymity of hosts and components and (2) changes in trust relationships, e.g., resulting from an expiration of the contract between a vendor and TravelAnywhere or between clients and their trusted hosting sites.

Once deployed, TravelAnywhere components must discover other appropriately authorized TravelAnywhere components and establish inter-component communication channels, which we view as mutually authorized pair-wise *coalition* relationships. Each component monitors the authorization of its partners to guard against access by agents whose authorization is revoked.

This deployment scenario can only be approximated by conventional infrastructures. For example, TravelAnywhere may have installed dedicated systems that they directly authorize. In addition, other vendors may contract with TravelAnywhere to provide hosting environments with acceptable security properties. A conventional approach to this deployment problem would involve explicit pre-configuration of each potential host to provide appropriately isolated "sandbox" environments for "TravelAnywhere" components. Such levels of customization substantially increase the cost and limit the flexibility of component deployment. Furthermore, sandbox permissions are typically not adjustable at runtime, thus inhibiting the ability of systems to disable already-deployed components if authorization is lost.

Furthermore, the authorization of dynamically formed coalition partnerships between deployed components is problematic with currently available communication and service discovery substrates. Insecure discovery and RPC substrates such as Java's JINI and RMI can be utilized on systems with secure networks, but such a deployment prohibits the deployment of components of multiple trust domains onto the same host. An alternative approach would utilize secure communication substrates such as TLS which do not provide mutually anonymous or continuous authorization.

DisCo middleware can facilitate flexible deployment of the TravelAnywhere application: DisCo's container abstraction provides a more general mechanism for establishing an appropriately permeable membrane between components and system resources. With DisCo, TravelAnywhere and administrators of computational resources can instead indirectly authorize and configure hosts through policies expressed as credentials in a distributed trust management system. DisCo's authorized discovery and switchboard communication library similarly provide a flexible infrastructure to support the authorized communication needs of the deployed components.

3. Facilities DisCo Provides

Below, we enumerate the facilities provided to DisCo application components.

Component Deployment and Installation

- Authorized and authenticated code distribution
- Remote and local installation of components
- Lazy code distribution
- Execution environments that impose access limitations based on attributes of component and user authorization

Monitored Authorization and Communication

- Authorized locality-aware discovery of components providing named services
- Maintenance of parameterized inter-component connection security and transport properties, including secrecy, integrity, timely delivery, liveness, and timely response to changes in the trust relationship that authorized the connection
- Ability to register a callback to handle failure if authorization and transport requirements are violated.

Adaptation to Environment Changes

- Notification from the monitored authorization and communication sub-system of changes to authorization, which facilitates adaptation or failover to alternative remote components should connectivity or authorization be lost
- An application- and connection-specific level of indirection between connected service components which can be used to implement security filtering, alternative functionality, performance modulation, etc.
- Lifetime management of data structures involved in inter-component coalitions (cleanup, garbage collection, etc.)

Functionality in the above features has been realized through DisCo's libraries, abstractions, and core components built on top of Java 1.4. The components can be conceptually grouped into four classes. First, a common authorization and access control model comprises the core of all other components and is described in section 4.1, with a specific realization of this model "dRBAC" described in Section 4.2. The second class, described in 4.3, includes components for authorization- and liveness-aware component communication and an indirection design pattern for communication. The third class, a minimal runtime system capable of on-demand component installation within a container execution environment with appropriate rights is described in 4.4. Finally, a locality-aware discovery system with failover is described in 4.5.

In the next section, we present the core components of the DisCo middleware in greater detail. In subsequent sections, we describe sample applications, a performance evaluation of our online communication components, and a description of related work.

4. Architecture

We begin with DisCo's unified authorization abstraction, followed by a realization of this abstraction named dRBAC. We then describe major subsystems that provide authorized inter-component communication channels and execution environments.

4.1. Access-Control Infrastructure

All parties that engage in authorized relationships within DisCo are identified using public keys. Trust-sensitive coalition relationships in DisCo are authorized using a generic abstraction called an `Authorizer`. DisCo components define Authorizer objects for each security-sensitive interface. Authorizer objects evaluate authorization requests that contain only the requester's public-key identity and credentials. If the Authorizer determines that the requester has the required access rights, a DisCo authorizer returns an `AuthMonitor` object representing the authorizing relationship. The simplest AuthMonitors represent

```
interface Authorizer {
  AuthMonitor authorize(
      PublicKey id,
      Credential creds);
}
interface AuthMonitor {
  boolean isAuthorized();
  updateCredentials(Object creds);
  addCallback(AuthMonitorCallback t);
  removeCallback(AuthMonitorCallback t);
}
```

Figure 1. Authorizer and AuthMonitor Interfaces

boolean authorization relationships, more advanced authorization monitors that provide parameterized authorization (for example, to authorize the access rights of a program from a partially trusted publisher) are also defined. Figure 1 shows the signatures of the DisCo Authorizer and AuthMonitor classes.

Security-sensitive interfaces can either poll authorization monitors (by calling the *isAuthorized()* method) whenever a restricted transaction is requested, or instead register interest in a change of authorization through a callback interface. Authorization monitors can retain state and efficiently monitor the status of dynamic authorization relationships. For example, an authorizer that grants access based on a time-sensitive credential can schedule itself to be invalidated when that credential expires. Alternative authorization monitor designs can subscribe to online services for credential validation, replacement or revocation.

The DisCo library includes Authorizers and AuthMonitors realized using the dRBAC system described below. However, the authorization-management system implemented within an Authorizer and the contents of its corresponding credential objects are opaque to all other DisCo core components. Alternative authorization monitors that implement other access control and authorization management systems such as ACLs or the RT [16] attribute-based access-control systems could be easily constructed and be used seamlessly throughout DisCo.

4.2. Summary of dRBAC's Features

dRBAC is a role-based "trust management" system for coalition environments. Like other distributed role-based access control systems, dRBAC's credentials, called *delegations*, express the granting of an equivalence class of access rights in one authorization domain to members of another equivalence class, possibly in another authorization domain. Each of these equivalence classes is represented by a dRBAC *role*. A summary of other relevant features of dRBAC follows; a more complete description appears in [7].

Each dRBAC delegation is cryptographically signed by

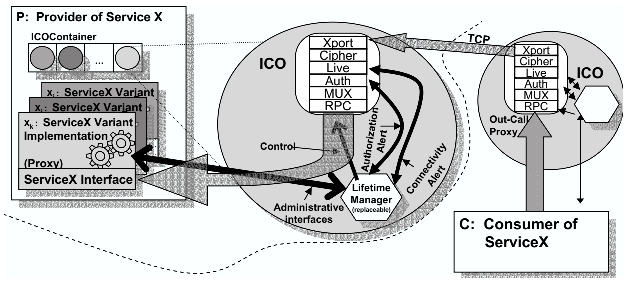

Figure 2. DisCo objects used in a SwitchboardRPC Connection Through an ICO

its issuer.[2] As with other role-based access control systems, dRBAC delegations may be transitively chained to form proof graphs indirectly authorizing a required class of access rights. A dRBAC credential can also contain

- issue and expiration dates
- requirements for (continuous) online validation
- search tags that assist in the online discovery additional transitive credentials
- attribute bindings that modulate access rights

dRBAC credentials are stored in a distributed repository. To assist in collecting dRBAC credentials that authorize a particular partnership, an automated credential discovery mechanism has also been constructed.

A dRBAC Authorizer contains the dRBAC role and limiting attenuation attributes that authorized subjects must possess. The DisCo library contains a convenient factory object to generate appropriate dRBAC Authorizers. dRBAC constructs a proof that the subject requesting access has the required authorization, returning a ProofMonitor. ProofMonitors extend the AuthMonitor interface, and are responsible for detecting and reporting changes in authorization.

dRBAC can express and enforce multiple permissions at modulated levels. For example, the following delegation:

$$\textit{MIT.stu} \rightarrow \ \textit{NYU.stu} \ \ \textit{w/ NYU.libLoan} < 30$$

could express an authorization relationship providing an MIT student the rights available to an NYU student, but with library loan privileges limited to less than 30 days. In our demonstration applications described in Section 5, this representation of modulated access rights is used by DisCo's authorization management components for a variety of purposes including the configuration of an execution container's permeability.

4.3. Inter-Component Communication

The Switchboard library, an earlier version of which is described in [8], provides a substrate for secure and autho-

2 Additional credentials may be required as evidence of the issuer's authorization to administer the rights granted by the delegation.

rized inter-component communication. Switchboard coalition partnerships can be secure, authenticated, continuously *and* mutually authorized connections whose channels are monitored for connectivity.

Switchboard communication channels are dynamically constructed from modular internal components that implement encryption (and identity validation), liveness monitoring and authorization, and modular interface components that implement stream, object, and RPC interfaces. These components are assembled automatically as specified by application-specific constraints. For example, if a Switchboard communication channel has no connectivity requirement, it will contain no liveness monitor.

Authorization in Switchboard is mutual: switchboard connections are only consummated if authorized by both parties. Potential partners specify Authorizers; authenticated identities and credentials are conveyed during a communication channel's negotiation phase,

Components may require that clients be provided customized interfaces. For example, some interfaces may require per-partner accounting and enforcement of dynamically changing access limitations. Other interfaces may publish a single remote interface to all eligible partners. These two interface policies can be effectively achieved by following the well-known adapter and proxy design patterns [6], respectively. To permit the efficient implementation of both patterns, we encapsulate partnership policy for coalitions established using RPC within LifetimeManager objects that (1) examine the authorization monitor associated with the connection; (2) import and export appropriate interfaces; and (3) respond to changes in authorization and liveness. The Switchboard libraries include a standard LifetimeManager class which implements a proxy service exporting a single interface to all eligible suitors.

The construction of components that export RPC interfaces to multiple clients is simplified through the use of the Switchboard "hydrant" class. Customization is achieved through the use of a LifetimeManagerFactory class that manufactures a LifetimeManager for each connection.

Figure 2 illustrates the relationship between a Switchboard hydrant (server) *S* implementing an adapter pattern and a typical client *C*. In this case, a unique interface object $X_1..X_n$ has been manufactured for each client by the connection's LifetimeManager based on each client's authorization relationship with X. As illustrated in this Figure, a Switchboard interface, authorization monitor, and lifetime manager are all encapsulated within a per-connection indirect communication (ICO) object.

4.4. Runtime System

Hosts require protection against rogue objects loaded from external sources. To achieve this, DisCo's runtime subsystem provides mechanisms to securely control the distribution, integrity, and installation of code corresponding to an object provided by an external source. Once an object is instantiated, its execution container is limited by a membrane whose permeablity is derived from authorization relationships with both the code's publisher and the agent on whose behalf it is executing.

`ContainerSecurityAuthorizers`, extend the authorization abstraction used throughout DisCo, and provide a bridge between access control systems (such as dRBAC) and administrative security policy. As with `Authorizers`, ContainerSecurityAuthorizers authorize a code publisher, and are factories for `ContainerSecurityMonitors`. These monitors represent authorization to download publishers' class files and to define Java security contexts associated with each object.

As with authorization monitors used throughout DisCo, ContainerSecurityMonitors can, dynamically modulate permissions after instantiation. Our extensions to Java security contexts, described below, synthesize permissions from authorization relationships with both the code provider and the agent (local or remote user) that instantiates each object.

Java's SecureClassLoader interface is responsible for the secure retrieval of remote code and associating an access control object called a "security context" with every code source. Standard Java security enforcement mechanisms are provided that are consulted each time an object attempts to perform some restricted operation. Our approach is to extend this model to provide both "user" and "class" security as follows: As with Java's design, a security context is associated with each code source. In addition, at the time an agent "logs in" to a system, a new thread group is established, which is also associated with its own security context. This model permits any host security policy to modulate access rights based on authorization relationships between both users and code publishers.

DisCo's extends Java's secure class loader to utilize Switchboard to obtain class files and establish authorization relationships with code providers. AuthMonitors for the host attempting to obtain code provide a front-line defense against the instantiation of objects from unauthorized sources. These same AuthMonitors are also used to define container security policy. Switchboard's symmetrical authorization of both correspondents permits a code provider to restrict distribution of their code to authorized hosts.

As with other components of DisCo, ContainerSecurityMonitors (which extend AuthMonitors) can enforce a range of security policies, as appropriate for a particular application. A trivial ContainerSecurityAuthorizer can mimic a conventional Java security policy by returning a static set of permissions based on the authorization relationship between the host and the code provider, thereby deferring access control decisions to standard Java security enforcement mechanisms. More aggressive implementations can evaluate each request for privileged access in the context of a dy-

```
class Locator {
  Object      serviceName;
  InetAddress serviceAdministrator;
  InetAddress publishingServer;
  PublicKey   publishingServerID;
  Credentials publishingServerCreds;
  Signature   sig;
}
class ServiceDescriptor {
  Object      serviceName;
  InetAddress serviceAdministrator;
}
```

Figure 3. Locator and ServiceDescriptor interfaces

namic trust management system.

Thread security policy is managed by `UserSecurityMonitors`. The same underlying mechanism is used to establish thread-level security for local and remote users: a login control object authorized to construct thread security contexts associates a new thread group with a UserSecurityMonitor. We refer to this process as "user context activation." Secure component deployment to remote hosts is implemented using Switchboard connections to "activation agents" that establish authorized user contexts.

4.5. Locality-Aware Discovery

In order to minimize communication latency, it is often preferable to obtain network services from nearby providers. Disco's discovery subsystem provides a mechanism for components to locate nearby providers of needed services. Distinct services are identified with a `ServiceDescriptor`.

Providers publishing services register signed and credentialed `Locators` referencing themselves as authorized servers for an enclosed ServiceDescriptor. Locators are self-certified using dRBAC authorizers in order to limit the propagation of rogue locators.

Clients interested in connecting with a particular service generate a discovery query specifying a specific service descriptor. To provide reliable fail-over in cases where the discovery mechanism does not locate a local authorized provider, ServiceDescriptors also identify a default "service home." The locator interfaces are presented in Figure 3.

The *Service Advertisement* interface controls active broadcast of Locators, describing the service to be advertised. The *Service Publishing* interface allows a service to tell the local Discovery module to passively wait for and respond to requests that match the ServiceDescriptor elements of a provided Locator. Finally, the *Service Discovery* interface sends out a request for Locators that offer the service specified in a ServiceDescriptor. The `find` method takes an Authorizer that can evaluate the credentials provided by any returned Locators.

5. Sample DisCo Applications

We have constructed several applications to evaluate DisCo's usefulness as an infrastructure for secure deployment of decomposable applications in dynamic partly-trusted network environments. These include a secure video distribution service, a multi-player game, an Internet-access provider for transient wireless users, a secure multi-resolution imagery distribution system, and a secure mail distribution system. The last two are described below. Our secure multi-resolution imagery distribution system directly utilizes DisCo subsystems to deploy components and enforce security constraints. Our secure mail application leverages an optimizing automated deployment planner [13].

Both of these applications utilized dRBAC, Switchboard communication channels, and DisCo activation of component installation. As was anticipated, application "code bloat" for both applications due to the inclusion of security constraints was dominated by the definition of trust relationships rather than their enforcement.

5.1. Secure Multi-Resolution Image Distribution

This application distributes imagery to users with varying levels of permission. Users with high "security clearance" are permitted to access finely detailed imagery, users with lower clearance are only provided correspondingly lower levels of detail. Our master image server provides image data to authorized clients and proxies at a variety of resolution levels. Caching image proxies are automatically "activated" on hosts near to clients; during the activation process, the master server constructs a customized proxy instance with permission to access and provide imagery at resolutions appropriate to the activation host. Clients similarly install image viewers into execution containers on their hosts from "viewer provider" services. These viewers prefer to obtain imagery from nearby proxies when available, and default to the master when no nearby proxy has sufficient authorization to provide data at a needed resolution.

RPC-over-switchboard was used to implement image delivery. Interactive response latency was largely due to the online computation required to generate foveated tiles for transmission, communication latency, and the algorithm used to render images as opposed to features of the DisCo infrastructure.

5.2. Secure Mail

The "secure mail" application provides a component-based realization of a mail service that is automatically deployed in a dynamic network spanning multiple trust domains. This secure mail service is part of a related research effort (see [26]) that is examining strategies for deployment planning, which satisfy both security and performance constraints. DisCo is used to implement and enforce security constraints between components deployed by this system.

The hosts and network resources available for this application provide varied levels of performance and security that may vary over time. The component-based structure of the mail service enables component deployments that are customized to the properties of the network. In particular, the application consists of server, cache, and client components, each customized before delivery. Additional cipher components help realize a security model where each user and mail message has an associated security level. The latter indicates a need for encryption when a message is being transferred through or stored in insecure components.

Unlike the application described above that deploys components in a greedy manner, this mail service relies on a *planning* subsystem to compute optimized deployment configurations. DisCo's trust management, communication, and activation facilities are well matched to this automated deployment application, allowing the developer to avoid writing security-enforcement code.

5.3. How DisCo Helps

"Code bloat" due to security considerations can be partitioned into the unavoidable definition of security constraints and code that enforces them. Security-conscious systems will frequently contain substantial amounts of code in both categories. As was anticipated, this bloat, for all the DisCo applications we constructed, was dominated by the definition of former, with only a negligible amount of code dedicated to the latter. In most cases, the only direct exposure of communication authorization mechanisms was as arguments to the constructors for Switchboard communication interfaces. In addition, because it is difficult to determine the correctness of security code, it is beneficial for commonly used idioms to be provided in standard libraries.

6. DisCo Status and Performance

DisCo is being implemented as a set of Java libraries, building upon functionality contained in Sun Microsystems' JDK 1.4. The latest snapshots of the DisCo code can be downloaded from our web site: http://www.cs.nyu.edu/pdsg/ (follow the Software tab).

The DisCo middleware continues to be a work in progress, but has been under development now for over two years. The design of some of the underlying abstractions have been described in detail elsewhere [7, 8], but not as part of a larger system.

Prototype implementations exist of all major DisCo subsystems and they have been exercised through our construction of well-behaved security-aware distributed demonstration applications. We are examining our code for weaknesses against components that maliciously attempt to circumvent our security infrastructure.

Size	Transports			
	Switchboard-Obj		Socket	
	null	Blowfish	null	TLS
1KB	1.2(0.9)ms	2.5(2.2)ms	0.1ms	0.8ms
8KB	1.9(1.3)ms	5.9(4.8)ms	0.6ms	5ms
32KB	4.5(2.0)ms	16.3(13.8)ms	2.2ms	failed

Table 1. Switchboard communication latency

6.1. Performance Implications

The computation performed at the time an authorized relationship is established is setup cost that is (1) dependent on the authorization mechanism used and (2) amortized over the life of the authorized relationship. In this section, we examine the non-amortized "ongoing" cost of DisCo's continuous authorization and security mechanisms, and its RPC and object-transfer APIs.

6.1.1. Switchboard Switchboard has a modular internal design inspired by Cactus [27] that permits the construction of communication channels with security and interface characteristics appropriate for a range of applications. The authorization monitor model enables low-cost dynamic authorization of communication channels by decoupling the (re-)evaluation of credentials from communication. All measurements presented in this section were made using a 1.2GHz Athlon running Sun Java 1.4 under Linux.

Table 1 presents a comparison between the communication latency of Switchboard's object delivery transport and Java sockets. In these experiments, we used the TLS implementation provided in Sun's Java 1.4 distribution, which was unable to transfer 32k payloads. The payloads for the switchboard object transport are byte arrays, that are serialized using Java RMI MarshalledObjects.

To minimize effects outside of our implementation, data was transferred via a loopback (localhost) device. Parenthesized values are normalized by serialization time to account for the differences between stream and object APIs. Switchboard latency with a null cipher is generally 1-2ms slower than sockets for small payloads and is approximately equal for larger payloads, indicating its relative efficiency.

The Switchboard cipher presently relies on Bouncycastle's pure Java implementations of Blowfish and RSA and therefore remains a performance bottleneck. Substantial speedup can be achieved through replacement of these components with native code linked using the JNI interface.

Table 2 presents RPC latency for a variety of payload sizes and security parameters. Authorization and liveness monitoring contributes 1 and 2ms respectively.

Switchboard's implementation of RPC uses Java's introspection, which is notoriously slow, resulting in a latency twice that of Sun Java's RMI. Bytecode engineering techniques, which have been employed by other projects to

Payload Size	-	cipher	cipher auth	cipher auth live
1KB	5ms	7.2ms	8ms	11ms
8KB	5.5ms	10ms	10.7ms	13.3ms
10KB	5.7ms	10.9ms	11.6ms	14.5ms
32KB	7.8ms	20.4ms	21ms	23.8ms

Table 2. Switchboard RPC latency.

achieve significantly higher performance, are directly applicable to Switchboard's implementation of RPC.

Switchboard provides continuous mutual authorization semantics not available from other systems, with acceptable performance for inter-host component communication. However the latency of a switchboard RPC is two hundred times the latency of a direct method call, and therefore imposes a large penalty for intra-host communication. Our initial experiments indicate that alternative techniques that automatically generate customized outcall "proxies" and incall "skeletons" indicate that these techniques will reduce the latency of this indirection by more than an order of magnitude.

6.1.2. Containers and Secure Class Loading DisCo's two-tiered security enforcement mechanism, which consults both class and user privileges, is inherently more expensive than the class-only approach intended by the Java design. DisCo's approach first evaluates class security policy and then defers to user security policy when required.

Since authorization relationships typically change infrequently, the cost of inserting a dynamic trust management system into object access control mechanisms is normally small. Our approach is to define new code-source and user permissions only at the time the authorizing trust relationship is established (such as when users log in or code sources are first referenced) and at times that the underlying authorization system detects a change in access rights, (such as when credentials are revoked). Following the Java security model, these permission objects are only checked when a restricted operation is requested.

Initial experimental results, which measure the time required to repeatedly execute a privileged operation (opening a file) do not detect a significant timing difference between DisCo's and Java's default security policy.

7. Related Work

DisCo is an integrated framework for constructing distributed security-aware applications on systems administered by multiple administrative authorities. This framework is built as several subsystems implementing authorization, secure communication, dynamic deployment, sandbox configuration, and locality-aware discovery. In this section, we place our work in context by discussing other systems that address similar challenges.

7.1. Component Deployment Systems

The Java Enterprise Edition (J2EE)[25] provides a component-based application model that deploys components expressed as "beans" into containers. The J2EE model is designed for server farms within a single trust domain, where a bean deployment specification is manually generated at the time the application is configured. All code is unconditionally trusted and inter-bean communication is provided using unchecked (trusted) RMI. While J2EE does provide a level of protection between well-behaved program elements, authorization is only performed at the start of a user transaction. Thus, the J2EE framework as currently implemented does not appear to provide appropriate inter-component guarantees required for systems spanning multiple administrative domains with asymmetric trust relationships.

More general are frameworks such as .NET [21] and the emerging Web Services effort, which are developing standards to orchestrate interactions among static components resident across multiple administrative domains. However, web services security-related standards such as WS-Security [20] only provide mechanisms for encoding security credentials without explicitly specifying policies for enforcing various guarantees. Additionally, these frameworks have not explored how long-lived connections between components should respond to changes in authorization or connectivity.

Dynamic component deployment onto secure containers is being explored in some research projects such as Ninja [10] and Ajanta [14]. DisCo shares several of its objectives with such systems. In particular, DisCo's secure code loading facilities provide an alternative to Ajanta's mechanism for authenticated, encrypted communication with the code base.

DisCo differs from the above systems in two important ways. First, it supports dynamic deployment in environments where all entities do not trust a single authority. Second, it provides mechanisms to continuously monitor the authorization of credentials and connection characteristics, while making it convenient for applications to adapt as necessary. Thus DisCo's mechanisms are complementary to what is offered by the above systems.

7.2. Secure and Quality-Assured Transports

DisCo utilizes Switchboard as its principal inter-container and inter-host communication abstraction. Switchboard, which builds upon JDK 1.4's cryptographic interfaces, provides novel security attributes including continuous monitoring of connectivity and authorization. These attributes are either not available in current transport-layer alternatives or are implemented in a non-modular fashion.

SSH [28] and TLS [5] are the most widely available alternatives to Switchboard. The authorization model they provide is not modular, but instead is tied closely to ACLs and X.509 certificates, respectively. TrustBuilder [19] extends the TLS model to use Winsboro and Li's more expressive role-based authorization model RT. OO-DTE [4] utilizes DTE-based authorization. None of these systems provides mechanisms to monitor connectivity.

IPSEC [15] provides a mechanism to securely construct a secure network among sites connected via the public Internet. While IPSEC does not provide liveness guarantees, its security guarantees permit TCP to reliably detect the failure of a connection. As a network-level mechanism, IPSEC is unaware of the protection domain of components, which originate packets or listen to ports and therefore is inappropriate for the security needs of containers on the same host that require insulation from each other.

Modular research systems such as Cactus [27] and JXTA [22] provide alternate mechanisms that can be utilized to implement or further generalize the abstractions provided by Switchboard. However suitable toolkits for these systems were not available at the time our implementation of Switchboard was developed.

7.3. Authorization and Access Control

DisCo provides a mechanism for a trust management subsystem to continuously monitor a coalition partner's authorization. In addition, DisCo relies on authorization subsystems to set access level parameters. Both of these mechanisms are realized using dRBAC, a decentralized role-based access-control system [7].

X.509 and several role-based access-control infrastructures such as SPKI/SDSI [24], PolicyMaker[2], OASIS [11], and the RT_n systems of Li and Winsboro [17, 16] are similar to dRBAC in providing trust management mechanisms that allow access privileges to be chained or delegated among multiple trust domains. Also related, but significantly less expressive, is Project JXTA's Poblano [3] trust-management system which provides a PGP-like *web-of-trust* model. DisCo's generic authorization model can be easily extended to support any of these alternatives.

DisCo's dRBAC differs from the above systems in two respects. First, it provides valued attributes, permitting convenient and flexible modulation of access classes instead of requiring explicit enumeration of all combinations of access parameter values as different roles. (RT_1, which was developed at around the same time as dRBAC, has a somewhat related feature as well).

More importantly, dRBAC provides facilities to continuously monitor an entity's authorization. With the notable exception of the OASIS [11] system, the above trust management systems have not addressed this challenge at any level. Moreover, OASIS' abstractions and implementation are intended for management within a single trust domain and are inappropriate for systems involving multiple domains. The X.509 specification [12] notes a need for continuous monitoring for credential expiration and revocation, however mechanisms to achieve this are not provided in standard implementations.

7.4. Discovery

DisCo's locality-aware discovery service is a hybrid between a centralized approach and local flooding. Flooding is also used in other discovery systems such as Jini[1] and Gnutella[23]. Like, Ninja's SDS[10], DisCo's discovery service annotates referral credentials with public-key signatures, allowing others to authenticate their integrity. In order to permit authorized parties to issue their own discovery credentials, DisCo's discovery credentials also include the issuer's credentials which indicate the issuer's right to provide the discovered object.

The P2P community has recently begun exploring mechanisms for incorporating locality-awareness in distributed hash tables (for example [18]). While these locality-aware hashing techniques are presently immature, they are likely to be more efficient than flooding and therefore more effective for our purposes.

8. Conclusion

The DisCo framework facilitates the construction of decomposable applications that can be deployed into an environment characterized by dynamic network connectivity properties and unstable trust relationships that span multiple administrative domains.

Multiple mechanisms are currently available that provide partial solutions to the challenge of security in such distributed systems and generally do not inter-operate well. In contrast, the DisCo framework provides a composable and extensible set of primitives that express access rights, facilitate secure inter-component communication, enable component deployment, and provide locality-aware service discovery. In addition, mechanisms are provided to detect and respond to loss of authorization and connectivity.

DisCo permits application components to discover, securely acquire the code for, and install service components over the network with only minimal a priori knowledge of their locations. Once installed, these components can securely interoperate and respond to changes of authorization and connectivity between hosts, users, and software agents.

In addition to the construction of an extensible framework, DisCo's contributions include a novel universal abstraction for continuous authorization and secure inter-component communication used throughout the system. This research effort has also resulted in the development of a novel distributed trust management system which provides mechanisms for discovering and expressing authorizing relationships that modulate multiple access control attributes.

Acknowledgments

We thank Christian Almazon, Hesham Hassan, Sean Holohan, Edward Keenan, Oliver Kennedy, Tracy Pesin, Larry Port, Robert Schatz, Nick West and Kenji Yoshihira for their assistance implementing and integrating DisCo's components.

This research was sponsored by DARPA agreements N66001-00-1-8920, and N66001-01-1-8929; by NSF grants CAREER:CCR-9876128 and CCR-9988176; a DARPA seedling grant for "Stable Trust Management". The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of DARPA, SPAWAR SYSCEN, or the U.S. Government.

References

[1] K. Arnold, A. Wollrath, B. O'Sullivan, R. Scheifler, and J. Waldo. *The Jini specification*. Addison-Wesley, Reading, MA, USA, 1999.

[2] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proc. CCS*. ACM, 1996.

[3] R. Chen and W. Yeager. Poblano: A Distributed Trust Model for Peer-to-Peer Networks. Available at `http://www.jxta.org/project/www/docs/trust.pdf`, 2001.

[4] D. Sterne, G. Tally, C. McDonell, D. Sherman, D. Sames, P. Pasturel, and E. Sebes. Scalable Access Control for Distributed Object Systems. In *Proc. of USENIX Security Symposium*, 1999.

[5] T. Dierks and C. Allen. The TLS Protocol, Version 1.0. IETF RFC 2246, 1999.

[6] E. Gamma, R. Helm, R.Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[7] E. Freudenthal, T. Pesin, L. Port, E. Keenan, and V. Karamcheti. dRBAC: Distributed Role-Based Access Control for Dynamic Coalition Environments. In *Proc. ICDCS*, 2002.

[8] E. Freudenthal, L. Port, E. Keenan, T. Pesin, and V. Karamcheti. Credentialed Secure Communication Switchboards. In *Proc. of IEEE Wkshp. on Resource Sharing in Massively Distributed Systems*, 2002.

[9] S. Garfinkel and G. Spafford. *Practical UNIX and Internet Security*. O'Reilly and Associates, Inc., 1996.

[10] S. D. Gribble and et al. The Ninja Architecture for Robust Internet-Scale Systems and Services. *Special Issue of IEEE Computer Networks on Pervasive Computing*, 2000.

[11] J. H. Hine, W. Yao, J. Bacon, and K. Moody. An architecture for distributed OASIS services. In *Middleware*, pages 104–120. ACM/IFIP/USENIX, 2000.

[12] R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and CRL Profile. IETF RFC 2459, 1999.

[13] A.-A. Ivan, J. Harman, M. Allen, and V. Karamcheti. Partitionable Services: A Framework for Seamlessly Adapting Distributed Applications to Heterogenous Environments. In *Proc. HPDC*. IEEE, 2002.

[14] N. M. Karnik and A. R. Tripathi. Security in the Ajanta mobile agent system. *Software — Practice and Experience*, 31(4):301–329, 2001.

[15] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. IETF RFC 2401, 1998.

[16] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust management framework. In *Proc. 2002 IEEE Symposium on Security and Privacy*, pages 114–130. IEEE Computer Society Press, May 2002.

[17] N. Li, W. Winsborough, and J. Mitchell. Distributed credential chain discovery in trust management. In *Proc. CCS*. ACM, 2001.

[18] M. Freedman, Eric Freudenthal, and David Mazieres. Democratizing Content Delivery with Coral. In *Proc. NSDI*. USENIX, 2004.

[19] M. Winslett, T. Yu, K. E. Seamons, A. Hess, J. Jacobson, R. Jarvis, B. Smith, and L. Yu. Negotiating trust on the web. *IEEE Internet Computing*, 6(6):30–37, 2002.

[20] Microsoft. Web Services Security (WS-Security) Version 1.0. Technical report, Microsoft Corporation, April 2002.

[21] Microsoft Corporation. Microsoft .NET Framework SDK Beta 2. Available at `http://www.microsoft.com/net`, 2001.

[22] Project JXTA. JXTA Version 1.0 Protocols Specification. Available at `http://spec.jxta.org`, 2001.

[23] M. Ripeanu and I. Foster. Mapping the gnutella network: Macroscopic properties of large-scale peer-to-peer systems. *IEEE Computing Journal*, 6(1), 2002.

[24] R. L. Rivest and B. Lampson. SDSI – A simple distributed security infrastructure. In *Proc. of CRYPTO'96*, 1996.

[25] Sun Microsystems, Inc. Java$^{\text{TM}}$ 2 Platform, Enterprise Edition Specification, Version 1.3. Available at `http://java.sun.com/j2ee/docs.html`, July 2001.

[26] T. Kichkaylo, A. Ivan, V. Karamcheti. Constrained Component Deployment in Wide-Area Networks Using AI Planning Techniques. In *Proc. IPDPS*. IEEE, 2003.

[27] G. T. Wong, M. A. Hiltunen, and R. D. Schlichting. A configurable and extensible transport protocol. In *INFOCOM*. IEEE, 2001.

[28] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. SSH Protocol Architecture. Available at `http://www.ssh.com/tech`, 2001.