Switchboard: A Modular Communication Substrate Supporting Multi-Configured Networks

Christian Almazan, Eric Freudenthal
cba209@nyu.edu, freudent@cs.nyu.edu
New York University

Abstract

We have constructed a flexible substrate for peer-to-peer communication. In addition to supporting a component-based layered architecture, this substrate provides a novel modular authorization model and supports explicit monitoring of liveness guarantees.

1. Introduction

Peer-to-peer systems frequently sidestep the issue of communication security. Often, the only conveniently supported mechanisms (SSL [3], TLS [2]) for establishing encrypted communication channels impose a particular authorization model, such as X.509 [9], and provide no mechanisms for alternative authorization mechanisms. The objective of Switchboard is to provide a flexible and efficient communication substrate that can easily be configured to support a wide variety of communication needs in both open and security-sensitive environments. Novel contributions of Switchboard include a flexible modular authorization component and liveness monitoring. Communication links can be flexibly constructed with only the components required for a particular application, allowing the programmer to optimize out unneeded functionality.

The rest of this paper has been organized as follows. Section 2 describes how this modular substrate is implemented. Section 3 demonstrate sample components for this architecture. Section 4 show various components we have built. Section 5 describes other work related to ours. Section 6 tells what we have learned from developing Switchboard.

2. Architecture

Our objective is to provide a flexible, extensible communication infrastructure for security-sensitive peer-to-peer applications. To this end, Switchboard layers provide and require the same interfaces, thus facilitating the construction of communication channels with a variety of characteristics.

The design of this component-based communication substrate is analogous to the OS model. Our implementation has been done in Java. All Switchboard messaging interfaces are asynchronous. The following two subsections describe how to build one particular layer in detail (by extending a class called SbLayer): how to pass messages to each adjacent layer and how the stack handles events.

2.1. Layer Messaging API in Java

To send and receive messages to adjacent layers, a simple API had been developed. Classes that extend SbLayer simply have to override the following methods (all methods have a void return signature):

- `init()` - initializes a layer with their particular constraints
- `write(Object obj)` - defines how to send messages to the layer below
- `deliver(Object obj)` - defines how to send messages to the layer above
- `close()` - defines how to shutdown this layer properly

2.2. Event Handling API in Java

Events in this architecture can either be fatal (recoverable, such as a change of authorization) or non-fatal (unrecoverable, such as a connection failure). Events must be handled explicitly by each layer. Similar to the layer messaging API, classes that extend SbLayer can implement the following methods to customize their needs (all methods have a void return signature):

- `handleEvent()` - defines how to handle events that have been passed into the layer
- `handleEvent(SbProvider layer)` - like above, except it also indicates which layer passed the event
public void setSock(Socket sock) {
    transport = new SbSocketIoLayer(sock);
    cipher = new SbCipherLayer(
        transport, PK.loadKey(...));
    auth = new SbAuthorizationLayer(
        cipher, new TimedAuthorizer(10000), cipher);
    auth.setRecipient(this);
    auth.init();
}

Figure 1. construction of a simple stack that can be used by either clients or servers

- sendEventUp(SbProvider layer) - sends an event
- setEventHandler(SbEventHandler _eventHandler) - sets where events should be forwarded to

3. Layers Implemented for the DisCo Project

The initial implementation of Switchboard [6] was a monolithic design. In this section we describe the layers constructed for this more modular second-generation substrate. A Switchboard stack is assembled bottom-up as illustrated in figure 1

3.1. Transport Layer

This layer has the responsibility of providing object messaging transport between Switchboard stacks. All objects are serialized and transmitted as MarshalledObjects. Our present transport layer utilizes TCP sockets which allows communication between peers. Alternative implementations can provide local connections, utilize other transport mechanisms, or physically secure channels.

3.2. Authentication and Cipher Layer

Authentication layers provide authenticated identity of communication correspondents. Our present authentication layer utilizes a public key exchange protocol to reliably establish the PKI identity of the correspondent and to establish a session cipher. This cipher layer has the responsibility of performing encryption and decryption on data that has been passed through it. This type of attack is impractical when the communication channel is secured (e.g. by using the cipher layer described above). Non-fatal events are transmitted when object delivery time limits have been exceeded.

3.3. Liveness Layer

In some contexts, it may be a necessity to be aware of connection liveness. TCP keep-alive is not an appropriate mechanism to ensure liveness since it is vulnerable to man-in-the-middle spoofing. This type of attack is impractical when the communication channel is secured (e.g. by using the cipher layer described above). Non-fatal events are transmitted when object delivery time limits have been exceeded.

3.4. Authorization Layer

A wide range of authorization schemes are available for peer-to-peer. Our approach is to implement an authorization layer that utilizes a generic authorization abstraction that is suitable for a wide range of applications including an interface for transmitting updated credentials (described below). Authorization in Disco uses our dRBAC [5] authorizer; other authorizers (e.g. ACL or X.509 [9]) can easily be adapted for this scheme as well.

Trust-sensitive coalition relationships are authorized using a generic abstraction called an Authorizer. Authorizer objects are instantiated for each class of potential trust relationships and evaluate authorization requests that contain only client identity and credentials. If the subject has the required access rights, an authorizer returns an AuthorizationMonitor that, like a dRBAC proof monitor, contains an interface for installing callbacks and updated credentials. An AuthorizationMonitor provides a mechanism by which an authorization component, such as dRBAC, can diligently watch the status of the authorized trust relationship and provide timely callbacks should that status change.

3.5. Multiplexer Layer

This allows for an existing Switchboard communication stack to be utilized for multiple connection sessions with similar properties. This reduces the connection time since establishing a variety of security guarantees can be expensive to setup and also permits a single liveness monitor to be shared by multiple switchboard connection between two hosts.

3.6. Remote-Procedure Call Layer

This layer supports incall and outcall handlers which support passing objects through a stacked connection using simple API calls. This can be comparable to both Java RMI [11] and CORBA [7]. A user simple creates an interface to be distributed across peers and the connecting peers connect to a server which has implemented the interface and
just make calls against the interface. Like Java RMI, our multiplexer layer is used to permit multiple calls to be concurrently issued through the same Switchboard communication channel.

4. Example Uses

Below we enumerate several uses of various Switchboard stacks in use and their purposes. Figure 2 shows two adjacent stacks communication with each other.

4.1. Secure, Authorized, and Liveness Monitoring

In addition to establishing up simple transport mechanisms, such as stream and object transports, Switchboard security guarantees often not found in other commonly used secure transports like SSL [3] and TLS [2]. Using this layered architecture we support pluggable authentication, cipher, authorized, and liveness schemes. These allow any number of permutations of security to be used.

4.2. Remote-Procedure Call Mechanism

Through the use of the RPC and MUX layers. The switchboard RPC layer utilizes the MUX layer described above to efficiently support the dispatch of multiple outstanding requests. Our RPC implementation leverages Java's introspection API. While the cost of reflection is high, it does not pose a significant performance bottleneck for the applications we considered. A more aggressive implementation using byte-code engineering is being examined.

4.3. Secure and Authorized Code Delivery

We have observed three classes of security challenges related to code delivery. When peer A transmits code to peer B, a system receiving it in a context where code distributed is restricted, and both peers do not have a previously established a relevant trust relationship. These three classes are as follows: is peer A authorized to provide the code, is peer B authorized to provide the code, and can we ensure the secrecy and integrity of the code object? Our first solution to this challenge utilizes a Switchboard communication stack which solves the problem by doing the following: the authentication and cipher layers authenticate both peer’s identity which provides transport secrecy and integrity and the mutual authorization is provided by the authorization layers. At present, the code object itself is transmitted over the Switchboard communication channel.

4.4. Pluggable Authentication Schemes and Connection Reuse

As described as a motivating reason for creating a layered architecture, pluggable authentication schemes allow peers to connect to each other with varying requirements. With a multiplexer stacked on top of an authentication layer, threads and other executable programs with the correct rights can reuse the same connection that has the same authentication layer.

5. Related Work

IPSEC [8] is inappropriate for environments where hosts must communicate with others from multiple security domains.

The embedding of key exchange mechanisms into pairwise communication links is well-treaded ground. Unlike most other systems such as TLS [2], SSL [3], SSH [13], and Yalta [1], Switchboard does not associate a particular authorization scheme.

Cactus [12] also has a modular communication stack which also provides flexible mechanisms for establishing communication links with properties appropriate for a particular context. As opposed to our simple API, Cactus’ communication model exposes a much more complex abstraction. However, it provides lower-level abstractions for communication channels than were convenient for our purpose. We had been principally interested in establishing and maintaining dynamic trust in anonymous environments (see Switchboard [6]). Also, Cactus’ Java implementation is not sufficiently mature or robust for our application.

As previously mentioned, remote-procedure call mechanisms have existed for sometime. Examples include Java RMI [11] and CORBA [7]. Unlike these models which only provide certain pluggable components, Switchboard allows any number of components to be used.
6. Conclusions

We have used Switchboard as a communication mechanism within the DisCo environment and for several applications that have been written for it. Our experience is that it has been convenient, allowing us to conveniently establish communication channels with appropriate security properties for each application. The modular design allowed us to omit unneeded components, when possible, permitting us to optimize both for security and runtime costs.

References