

Revised Submission

CORBA Firewall Traversal

Submitters:

Borland Software Corporation
International Business Machines Corporation
IONA Technologies PLC
Sun Microsystems, Inc.
Xtradyne Technologies AG

Supported by:

Network Associates, Inc.

OMG Document orbos/2001-06-04
June 18, 2001

Copyright 2001 Borland Software Corporation
Copyright 2001 International Business Machines Corporation
Copyright 2001 IONA Technologies PLC
Copyright 2001 Sun Microsystems, Inc.
Copyright 2001 Xtradyne Technologies AG
Copyright 2001 Network Associates, Inc.
All rights reserved.

The companies listed above hereby grant to the Object Management Group, Inc. (OMG) and OMG members, permission to copy this document for the purpose of evaluating the technology contained herein during the technology selection process by the appropriate OMG task force. Distribution to anyone not a member of the Object Management Group or for any purpose other than technology evaluation is prohibited.

The material in this document is submitted to the OMG for evaluation. Submission of this document does not represent a commitment to implement any portion of this specification in the products of the submitters.

WHILE THE INFORMATION IN THIS PUBLICATION IS BELIEVED TO BE ACCURATE, THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. The companies listed above shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material. The information contained in this document is subject to change without notice.

This document contains information which is protected by copyright. All Rights Reserved. Except as otherwise provided herein, no part of this work may be reproduced or used in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems— without the permission of one of the copyright owners. All copies of this document must include the copyright and other information contained on this page.

RESTRICTED RIGHTS LEGEND. Use, duplication, or disclosure by government is subject to restrictions as set forth in subdivision (c) (1) (ii) of the Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013.

1.1 Mandatory Requirements

1. Proposals shall specify interfaces and interoperability mechanisms and conventions that enable server-side firewalls to control access from clients outside the firewall to CORBA object implementations inside the firewall, and also prevent access to IIOP-based services that should not be accessible from the outside. The proposal shall indicate whether the means for doing so involves the use of IIOP message header data or IIOP message data, and, if so, which fields are used.

The specification provides a means for firewalls to be traversed in order to reach servers located in enclaves protected by firewalls. How the firewall mitigates access to an enclave is implementation dependent, but any information provided by the transport can be used for access control decisions including host/port information, SSL identities, and the IIOP headers.

2. If the proposed solution permits access control based on strong authentication (SecIOP or IIOP/SSL or any other form of authentication) it shall provide interfaces and interoperability mechanisms for the ORB to authenticate to the firewall.

Authentication can be optionally performed using the SSL authentication mechanism.

3. Proposals shall specify interfaces and interoperability mechanisms and conventions that allow client-side firewalls to provide controlled reuse of the connection used for the request from the client to the server for the corresponding response from the server to the client.

This functionality is provided by bi-directional GIOP. Fixes for the bi-directional GIOP specification have been submitted in OMG document orbos/2001-06-05.

4. Proposals shall specify interfaces and interoperability mechanisms and conventions that enable client-side firewalls to control access by inside hosts to outside CORBA-based application services.

The outbound firewall traversal algorithm is given in section 3.8.1. Like in mandatory requirement one, firewalls can control access by a number of different mechanisms.

5. Proposals shall specify interfaces and interoperability mechanisms and conventions that permit firewalls to process IIOP as an ordinary application protocol. Proposals shall explain how such processing allows firewalls to:

- *determine what network traffic is expected to be IIOP (e.g. destination hosts, ports);*
- *differentiate between IIOP traffic that is permitted to enter the enclave, and IIOP that is not permitted;*

Firewalls can determine what traffic is IIOP in two ways. First, well-known ports have been assigned for IIOP and IIOP/SSL communication. Second, firewalls can be configured to accept IIOP traffic on other ports. CORBA servers are configured with the firewall address and port information, and that information is conveyed to the client ORB via an object's IOR. Those ports must be reserved for IIOP, so any communication on those ports is assumed to be IIOP.

See mandatory requirement one for how to differentiate access for various IIOP streams.

6. Proposals shall specify interfaces and interoperability mechanisms and conventions that firewalls can use to protect inside target object servers and clients from attack by data streams that are not valid IIOP.

A firewall that is capable of validating the IIOP messages is able to do so whenever plain IIOP is in use, or IIOP over NORMAL SSL is in use. It is not possible for transport level firewalls to validate IIOP, nor is it possible for application level firewalls to validate IIOP when PASSTHRU SSL is in use. See section 3.10

7. Proposals shall specify interfaces and interoperability mechanisms and conventions for coordination of ORB and firewall configurations so that object references (IORs) received by outside clients can be used to make requests that a firewall may allow.

This functionality is provided by the TAG_FIREWALL_TRANS tagged component in the IOR, described in section 3.3.

8. Proposals shall specify interfaces and interoperability mechanisms and conventions for all interdependencies between firewalls and ORBs.

Data structures are defined for all of the information that is to be shared between ORBs and firewalls, most importantly the FIREWALL_PATH service context in section 3.4.

9. *If the proposed solution intends to use bi-directional GIOP, the proposal must provide an interoperable solution to all outstanding issues regarding bi-directional GIOP. Note also that this RFP does not require any modification of existing CORBA specifications, e.g. IIOP, SecIOP, or IIOP/SSL. Likewise, responses to this RFP should not require any modification without a very compelling justification.*

The changes to bi-directional GIOP are included in OMG document orbos/2001-06-05.

1.2 *Optional Requirements*

1. *Proposals may specify interfaces and interoperability mechanisms and conventions to limit exposure of internal network topologies by restricting the release of host network addresses.*

This specification does enable vendors a means by which to hide network topology information by using DNS names rather than network addresses in the FirewallPath information.

2. *Proposals may specify interfaces and interoperability mechanisms and conventions to support fine-grained access control at the firewall. Access decisions may be based on the target object, requested operation or any other data that may be specific to an individual object invocation request message.*

This specification does not provide any mechanism for fine-grained access control at the firewall.

3. *Proposals may specify interfaces and interoperability mechanisms and conventions that allow firewalls to differentiate access based on authentication via firewall-to-firewall SSL transport of IIOP, so that authenticated requests can be permitted more access than unauthenticated requests.*

Support for IIOP/SSL is provided in this specification, however how a firewall uses the SSL authentication information is implementation-dependent.

4. *Proposals may specify interfaces and interoperability mechanisms and conventions that allow firewalls to differentiate access when SecIOP is implemented with IIOP, so that authenticated requests can be permitted more access than unauthenticated requests.*

SecIOP is not supported by this specification.

5. *Proposals may specify interfaces and interoperability mechanisms and conventions to support SecIOP or SSL across the firewall to provide private IIOP interaction with selected authenticated outsiders.*

See optional requirement three.

6. *Proposals may specify interfaces and interoperability mechanisms and conventions to perform IOP security functions at the firewall when SSL or SecIOP is used for end-to-end privacy between invoker and target object.*

See optional requirement three.

7. *Proposals may specify interfaces and interoperability mechanisms and conventions that allow firewalls to meet the mandatory, as well as any optional, requirements of this RFP for any other OMG inter-ORB protocols.*

No other inter-ORB protocols are supported by this specification.

8. *Proposals may specify interfaces and interoperability mechanisms and conventions to support load-balancing across multiple firewalls. Load balancing for high speed networks is of particular concern for server-side firewalls.*

Load balancing is not supported by this specification.

1.3 *Issues to be discussed*

1. *What information a firewall will use to discriminate access to object-based services (e.g. source/destination address/port, user authentication, user behavior, message contents, etc.)*

This specification provides a mechanism for CORBA servers to be accessed through firewalls. How a firewall discriminates access to CORBA resources is an implementation-dependent issue.

2. *What types of firewalls (e.g. packet filtering, application-level gateway, SOCKS proxy) and firewall configurations (e.g. single-homed bastion, dual-homed bastion, screened-subnet, multiple-tiered) the proposed specification will require and/or support as minimal capabilities.*

Transport-level firewalls (including packet filters) and application-level firewalls (including application proxies) are supported by this specification. The use of SOCKS proxies is not addressed. Any configuration of firewalls can be supported due to a generic connection establishment mechanism that depends on firewall configuration information provided to the server ORB.

3. *Provide use cases of invocations on objects using the supported firewall types and configurations from discussion item 2.*

See chapter 4.

4. *How IORs of inside objects are processed by firewalls, e.g., either:*
- *inside host/port pairs are handled transparently by firewall, or*
- *firewall translates real IORs to externally consumable IORs which direct invokers to the firewall.*

Inside host/port pairs are handled transparently by the firewall. The firewall receives a list of all of the firewalls in the path to the server. The address/port identifiers are resolvable by the firewall when making a decision on where to open a connection to.

5. How IIOP multiple profiles should be handled when the same profile type is used.

Which profile is chosen by the client ORB has no effect on the firewall traversal process. An IIOP profile can contain multiple firewall components, each representing a different path to the server. A client ORB can attempt to use any of the components in any order.

6. If approaches to multiple inter-ORB protocols are specified, what mechanisms may be common to a firewall's handling of multiple protocols.

Only IIOP and IIOP/SSL are supported. All of the mechanisms are the same for either transport.

7. Whether the proposed approach would benefit from (but does not require) modification of existing IIOP, SecIOP, or IIOP/SSL specifications, and rationale for any modifications.

Our approach will require that an additional message be added to the GIOP protocol that can be used to carry firewall topology information. See section 3.5.

8. The compatibility of the proposed approach with past, present, and proposed versions of the GIOP protocol, since later versions of the protocol may provide features unavailable in earlier versions.

This specification will only be compatible with a new version of the GIOP protocol because a new GIOP message type is required. See section 3.5.

9. How a client behind a firewall is able to traverse outbound firewalls in order to reach the invocation target.

See section 3.8.1.

10. How the proposed specification relates to the adopted firewall specification (orbos/98-07-03).

This specification completely supercedes the adopted firewall specification. Some parts of the adopted specification were reused, but most of the specification is not compatible with the adopted specification. Additionally, the proposed changes to bi-directional GIOP completely supercede the existing specification for bi-directional GIOP.

2.1 Overview

The overall goal of this specification is to provide better accessibility to CORBA application servers when there is a firewall separating a client from a server. In this context, "better" means that client-firewall-server communication can be enabled and controlled more easily for a broader range of circumstances. Currently, ORBs and firewalls have a limited form of "peaceful co-existence" that provides satisfactory functionality only in some cases.

There are two reasons why CORBA poses a unique problem for firewalls: server location transparency and the peer-based communication model. One of the primary benefits of CORBA is that a client does not need to know the exact location of an object to invoke on it. Therefore the server can be relocated or the object can be provided by a new server without changing any client-side information. This location transparency introduces a problem when a firewall is used to protect the server, because normally a firewall provides protection by only allowing inbound connections to a small number of well-known addresses. Therefore, the server location must be known by the firewall a priori.

The CORBA peer-based communication model also causes a problem when communicating through firewalls. Traditional internet applications use a client-server model where the number of servers is relatively small, and the servers are under the administrative control of a single organization. In CORBA, any host could act as a client or a server, so the number of "servers" is relatively high and the "servers" are under the administrative control of multiple organizations. The problem with a large number of potential servers is two-fold. First, Network Address Translation (NAT)¹ is deployed by many organizations to extend the available address space and to hide the

1. NAT allows an organization to use a private address space for internal use, and the firewall translates those addresses into globally unique addresses when clients communicate over the Internet. This allows many clients within an organization to share a few unique addresses.

topology of the internal network from the outside. In the traditional client-server model, the servers use actual routable addresses (non-NAT) for the publicly available servers. But if a large number of hosts could potentially be servers, then this solution eliminates the effectiveness of NAT. Second, firewalls can easily be configured with a policy to control traffic to a small number of servers and deny access to all other hosts. If the number of servers becomes large, then the firewall policy becomes both unmanageable and decreasingly effective.

This specification identifies the changes to CORBA that are needed for ORBs to function in a slightly different manner, so that CORBA communication can more easily be handled by firewalls. An additional goal of this document is to provide information on how current firewall techniques can be used to control CORBA communication. This information illustrates the benefits of current techniques, and also the limitations. The need to overcome these limitations is the impetus for this specification.

Interoperable CORBA communication occurs via the GIOP protocol, which on the Internet is implemented by the IIOP protocol. Because firewalls control IP networking communication, and because ORBs communicate via IIOP, this specification is concerned with various aspects of how firewalls handle the IIOP protocol. It is important to note that there is nothing particularly problematic about IIOP as an Internet protocol in terms of firewall processing. In fact, this specification does not modify IIOP in any way. Rather, this specification adds new data elements to CORBA (for example, in IORs) that provide clients, firewalls, and servers the information needed for flexible, efficient, controlled firewall traversal. In fact, if CORBA servers use a single IP address that is routable by all of the clients, i.e. the address of the server is unambiguous within the clients' enclaves, then no additions need to be made to the CORBA specification for basic invocations on CORBA servers.

2.2 *Firewall Principles*

In a CORBA environment, firewalls are used to protect objects from clients in other networks or sub-networks. A firewall will either permit access from another network to a particular object, or it will prevent it. Access through a firewall may be permitted at various levels of granularity. For example, access could be permitted to some objects behind the firewall based on network address, or access could be restricted to certain operations on particular objects.

An enclave is a group of objects protected by a firewall. The firewall protects the enclave's network (or subnet) by separating it from other enclaves and/or the Internet at large. The separation is the result of the fact that all communication between the enclave and the outside must pass through the enclave's firewall (or one of its firewalls, if there are several). Firewalls have two distinct duties: inbound protection and outbound protection. Inbound protections are used to control external access to internal resources. Outbound protections are used to limit the outside resources that can be accessed from within the enclave.

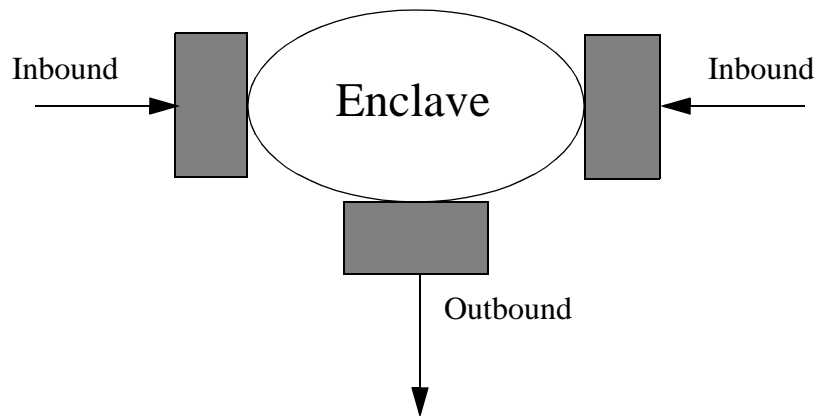


Figure 2-1 An enclave with multiple inbound and outbound firewalls.

Both aspects of firewall functionality are important for CORBA. A firewall's outbound protection functions should allow inside CORBA application clients and objects to initiate communication with objects outside the enclave. A firewall's inbound protection functions should prevent communication between outside clients/objects and inside objects that the outsiders should not be permitted to communicate with. Without a firewall's outbound protection, clients could access any resources. Without a firewall's inbound protection, all of the enclave's resources are unprotected from the outside world. Figure 2-1 illustrates an enclave with two inbound firewalls, and one outbound firewall. Note that although the firewalls are logically and functionally separate, they may share the same physical hardware, or even share the same address space.

Enclaves can be nested, such that an enclave may contain other enclaves in a hierarchical manner. This enables organizations to decentralize firewall access and have different access policies. For example, an engineering department prevents the finance department of the same company from accessing design documents. When enclaves are nested, a sequence of firewalls has to be traversed. A firewall protecting the outer enclave is called either an outermost inbound firewall or an outermost outbound firewall, depending on the direction of the invocation. The outermost inbound firewall represents an entry point into an organization. Figure 2-2 illustrates a hierarchical nesting of enclaves. The outermost "Company XYZ" enclave contains two sub-enclaves, "Finance" and "R&D". The "R&D" enclave further contains the "Research" enclave.

2.3 Types of Firewall

Broadly, there are two types of firewall: transport level and application level. A transport level firewall allows resources to be accessed via any application level protocols. Such firewalls do not understand the type of application protocol being used, rather access is based purely on addressing information in the header of transport packets. Hence, access decisions are based on the source and destination of a message

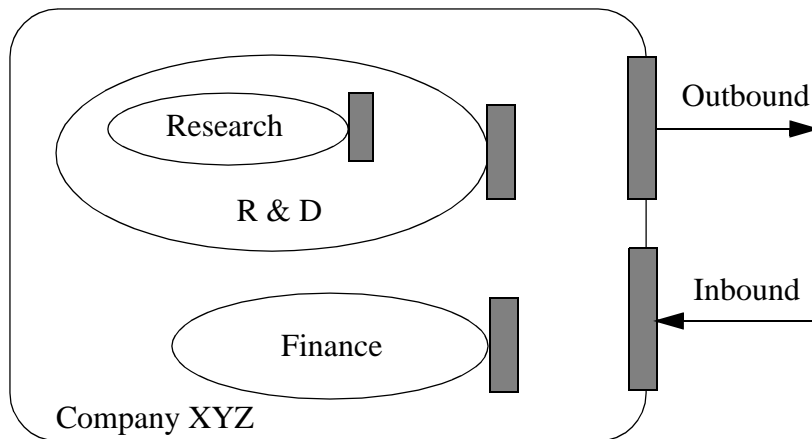


Figure 2-2 A hierarchical set of enclaves.

and not on the resource being accessed. Typically access control is performed during connection setup, and if the connection setup is successful, any application traffic may pass over the connection. A TCP firewall, for example allows access to FTP, HTTP, or IIOP resources, where access control is based on which hosts and ports traffic is travelling between.

Application level firewalls on the other hand are restricted to a particular application level protocol, such as IIOP or HTTP. As a result, access decisions can be based on both transport addressing information and on specific resources known to the application level protocol. For example, if there are two objects that can be accessed via the same host and port, it is possible for the firewall to deny invocations being sent to one object but to allow them for the other. This type of control requires monitoring the traffic after the connection has been established, and hence requires the firewall to understand the application level protocol.

2.3.1 TCP Firewalls

A TCP firewall is a very simple transport level firewall. It performs access control decisions based on address information in TCP headers. For ORB interoperability, TCP firewalls provide the simplest means to protect resources, but at the coarsest level of granularity i.e. host based control.

A TCP firewall works on a simple address mapping scheme: a connection request received on a certain port of the firewall, results in the firewall establishing a connection to a particular host/port. Once the two connections have been established, application level traffic can be sent from source to destination via the firewall. From an ORB perspective, GIOP messages will travel through the firewall uninterrupted i.e. ORB protocols are inconsequential to a TCP firewall.

The firewall can determine access control information by looking at the source address field in the TCP header, and make a decision as to whether that source host can connect through to the destination. A TCP firewall must have prior knowledge of the source to destination mappings, and conceptually has a configuration table containing tuples of the form: (<inhost, inport>, <outhost, outport>). When a connection request from <inhost, inport> is received, assuming the firewall allows connections from that particular client, a connection is set up to <outhost, outport>.

A simple form of ORB interoperability through TCP firewalls can be achieved without any additions to CORBA. Assuming a server is in an enclave protected by a TCP firewall, the server can be configured to know about this firewall and may substitute the host and port address of the server with the host and port address of the firewall in any IORs issued outside the enclave (how this is done is an implementation issue for the ORB vendor). Hence a client outside the enclave will receive an IOR that contains the address of the firewall and not the server. The client will therefore send GIOP messages to the firewall (which are forwarded to the server) thinking that the object is actually on the firewall. This scheme can be used independently of the other mechanisms described in this chapter, since it is completely transparent to clients. Often TCP firewalls are used in more complex configurations, where it is not feasible to use this scheme. In these cases the mechanisms described in this chapter can be used.

Since TCP/IP services typically use a port per service, it is common for TCP services to be identified by the port number used for the server. For example, SMTP mail is delivered on port 25, X11 traffic on port 6000, etc. As a result, most existing firewalls base their low-level access control decisions on the port used. ORB interoperability through TCP firewalls is currently impeded as there is no well-known IIOP port, therefore we define a recommended well-known IIOP port and a well-known IIOP/SSL port". Client enclaves with TCP firewalls will then be able to permit access to IIOP servers by enabling access to this port through their firewall. These ports are not mandatory, and IIOP servers can be set up to offer service through other ports if that is desired. However the ports serve as a basic guideline for server and firewall deployment, and allow client enclaves to immediately identify or filter the traffic as IIOP without requiring protocol analysis.

The well-known IIOP port is 683, and the well-known IIOP/SSL port is 684.

2.3.2 *Application Proxy*

An application proxy is an application level firewall that understands GIOP messages and the specific transport level inter-ORB Protocol supported e.g. IIOP. An application proxy firewall, or just application proxy for short, relays GIOP messages between clients and Objects. It may base access control decisions on information in the GIOP packet. For example, it could block requests to an object with a particular object_key, or it could block requests for a particular operation on an object.

To establish a connection to a server, a client first sets up a connection to the proxy. If the proxy is an outbound one, the ORB is configured with the address of the proxy. If the proxy is an inbound one, the server's IOR should contain the address of the proxy

object on the firewall. After a connection is established, the client interacts with the proxy object to establish a connection to the target server. The interaction(s) required with a proxy may be dependent on the transport mapping. Irrespective of how the client interacts with the proxy, and assuming appropriate permissions, the proxy will establish a connection with the server. Once this is done, the client and server may send GIOP messages to each other, according to the normal GIOP rules.

2.4 Rationale

There are a number of variables in establishing IIOP connections through firewalls. These variables include firewall topologies (placement of servers within networks and subnetworks), firewall types (transport, application), use of NAT, and the desire for SSL-encrypted communication. The underlying goal of this specification is to provide a means for ORBs to establish a connection from client to server in a uniform manner, regardless of these variables. Furthermore, a firewall generally maintains its security by only executing a small amount of well-tested code. For that reason, this specification makes it possible for a firewall to make access decisions in a deterministic manner without complex application-level interactions (e.g. invocations on CORBA objects) that may require an ORB implementation on the firewall.

3.1 Firewall Traversal Overview

A server indicates that the objects it serves reside behind a firewall by placing a `TAG_FIREWALL_TRANS` tagged component in the IOP profile of the IOR for that object. The information in that tagged component enables the client to make an invocation through the firewall that will eventually reach the server. This information includes an ordered list host addresses from the outermost inbound firewall to the target server.

In order to set up a connection to the target server, the client must first send a connection setup message to the target. This connection setup message is a `NegotiateSession` message discussed in section 3.5 and it contains a `FIREWALL_PATH` service context entry. The `FIREWALL_PATH` service context contains the information provided in the `TAG_FIREWALL_TRANS` component of the IOR. This information allows firewalls to open up the correct connections along the path to the server. Once the entire virtual connection has been established, the target returns a `FIREWALL_PATH_RESP` service context in a return `NegotiateSession` message, and the client and server can communicate using GIOP. The IOR entries, service context entries, and `NegotiateSession` message will be outlined in more detail in the following sections.

3.2 Host Identifiers

Each host along the path from the client to the target server can be identified as a collection of endpoints. An endpoint is an address by which a client can access the services provided by that host. The host identifier, `HostId`, of a host can be defined as follows:

```

module Firewall {

    typedef unsigned short ProtocolType;
    const ProtocolType IOP = 0;
    const ProtocolType NORMAL_SSL = 1;
    const ProtocolType PASSTHRU = 2;

    struct TransportEndpoint {
        unsigned short port;
        ProtocolType type;
    };

    struct HostID {
        boolean isIntelligent;
        string address;
        sequence<TransportEndpoint> endpoints;
    };
};

```

isIntelligent

Indicates whether or not this host is capable of processing the connection setup message. The `isIntelligent` attribute must be `true` for application proxy firewalls and servers, and it must be `false` for transport level firewalls. The one exception to this rule is when an application proxy is behaving like a transport level firewall; i.e. there is a static mapping from an incoming address/port pair to an outgoing address/port pair and the firewall does not examine the connection setup message.

address

The address of the host. Valid entries for this field are the same as the valid entries for the address field in the TAG_INTERNET_IOP profile as described in section 15.7.2 of the CORBA 2.4.1 specification. It is recommended that the value of the address field be a DNS name (if available) rather than an IP address because DNS names can be used to hide the network structure of the enclave. Using DNS names may provide some extra security since the list of firewalls to the target server will be present in the IOR, giving an attacker additional information. If the DNS names were only internally resolvable, the attacker would have less information about the topology of the network.

endpoints

Provides one or more endpoints by which this host may be contacted. Endpoints must be located at the same interface on the host, only the port may be different. Different endpoints may provide different types of SSL connections to or through the host.

port

The TCP port of this endpoint.

type

The ProtocolType supported by this endpoint. A ProtocolType of IIOP indicates that this endpoint supports only normal IIOP. A ProtocolType of NORMAL_SSL indicates that this endpoint will terminate the SSL connection, and establish a new SSL connection with the next host if necessary. One implication of using the NORMAL_SSL ProtocolType for a firewall host is that the security association is established using the credentials of the firewall rather than the credentials of the target server. The access control policy of the application must take this into consideration. A ProtocolType of PASSTHRU is only valid for firewall hosts. The PASSTHRU ProtocolType indicates that this firewall host is capable of establishing a connection such that after the connection setup message, the firewall will not process any application data. The packets will simply be forwarded through the firewall. No SSL connection is established with the firewall in this case, but an SSL connection can be established through the firewall, enabling the client and server to have an end-to-end security association.

3.3 Firewall Tagged Component

An IOR contains information about the target address of an Object, such as an address/port pair. In order to traverse a firewall, an IOR must also contain path information about the inbound firewalls as well. In a configuration where there are multiple enclaves (firewalls within firewalls) it is necessary to carry access information for all inbound firewalls. To include firewall information in an IOR, the following tagged component is defined.

```
module Firewall {  
  
    // The component with ID TAG_FIREWALL_TRANS contains a  
    // FirewallPath  
    const IOP::ComponentId TAG_FIREWALL_TRANS = xx;//OMG Allocated  
  
    typedef sequence<HostId> FirewallPath;  
};
```

The IOR Component with ID TAG_FIREWALL_TRANS contains a FirewallPath structure which is a sequence of HostId's. This FirewallPath contains the HostId's for all of the firewalls along the path to the server, including the HostId of the server. These HostId's must be in order from the outermost inbound firewall to the server.

Since transport-level firewalls use a static mapping from external host/port pairs to internal host/port pairs, it is possible in some firewall configurations to eliminate the HostId's of transport-level firewalls from the FirewallPath. For example, take the case where a server is located behind a transport-level firewall. In this case the FirewallPath could contain one entry, the HostId of the server, which contains the address and port of the firewall. However, this is only possible in the case where there are no clients in the same enclave as the server. If clients did share the enclave with the server, they would not have sufficient information to contact the server. Therefore,

this optimization should only be performed during the configuration of the firewall information on the server, and it is dependent upon the specific application and the firewall configuration.

The TAG_FIREWALL_TRANS component may appear multiple times within a profile. Each individual component specifies a separate path to the target server. For example, a network may have multiple access points through multiple firewalls. If this is the case, then multiple firewall components could be specified to allow clients to access the server through either firewall.

3.4 Firewall Service Context

The FIREWALL_PATH service context must only be sent in the connection setup request. The service context contains the ordered list of firewalls that the client chose in order to reach the server. Each application level firewall will be able to parse this service context to determine what the next hop in the path to the server should be. The FIREWALL_PATH service context is defined as follows.

```
module Firewall {  
  
    // The FIREWALL_PATH service context contains a FirewallPathContext  
    // structure  
    const IOP::Serviceld FIREWALL_PATH = xx; // OMG Allocated  
  
    struct FirewallPathContext {  
        long      host_index;  
        FirewallPath path;  
    };  
};
```

The FIREWALL_PATH service context contains a FirewallPathContext structure. The `host_index` field is an index indicating the current point in the firewall path. The outermost inbound firewall has index zero, and the index increases by one for each successive HostId with the target server having the highest index. The `path` attribute contains a list of HostId's, and the `path` is obtained from the FirewallPath in the TAG_FIREWALL_TRANS component of the IOR. The FIREWALL_PATH service context may only be sent in a NegotiateSession message and only during the connection setup period - see section 3.5.

When a client ORB needs to open a connection to an object with a TAG_FIREWALL_TRANS component in the TAG_INTERNET_IOP profile, the ORB extracts the FirewallPath entry from the IOR and places it in the FIREWALL_PATH service context. However, the FirewallPath structure in the IOR may contain multiple TransportEndpoints for each HostId. When sending the connection setup message, the client ORB must choose, based upon that client's policy, a single endpoint from each HostId to be placed in the FirewallPath structure in the service context of the message. If the FirewallPath incorrectly contains more than one endpoint for any HostId, a firewall or target server must use the first endpoint in the `endpoints` sequence when making connection decisions.

There are two reasons why it is important that a `HostId` only contain a single `TransportEndpoint`. First, a firewall or server must be able to determine whether the preceding firewall or client intends to connect using `NORMAL_SSL`, create a `PASSTHRU` connection, or neither. If multiple endpoints are available, the firewall or server may not be able to determine the preceding host's intention. Second, using a single endpoint allows the firewall to make a deterministic decision about what endpoint to connect to on the subsequent host in the `FirewallPath`. How the client ORB determines which endpoint to choose is described in section 3.6.

The `host_index` attribute is used by application firewalls and servers to locate their entry in the `FirewallPath` in order to determine the SSL policy and the next host in the path. The `host_index` attribute always indicates the next "intelligent" host on the path to the server, as indicated by the `isIntelligent` element of the `HostId`. Thus, when an application firewall or server receives the `FIREWALL_PATH` service context, the `HostId` indicated by `host_index` indicates that firewall's or server's `HostId`.

The client ORB can use any strategy to choose which firewall to send the connection setup message to. For example, a client ORB might first attempt to invoke directly on the server, and if that fails, attempt to successively open a connection to the firewalls in the `FirewallPath` until a successful connection is established. Another strategy might be to always first open a connection to the outermost inbound firewall. No particular strategy will work for all network configurations. The policy for making this decision is discussed in section 3.6.2.

After the client ORB has determined which firewall to send the connection setup message to, it must set the `host_index` field to be equal to the index of the `HostId` that is the `HostId` of the next "intelligent" host on the path to the server. If the `HostId` of the firewall that the client ORB chose to connect to has `isIntelligent` set to `true`, then `host_index` is the index of that `HostId`. Otherwise the ORB must find the first `HostId` in the `FirewallPath` following the chosen firewall that has `isIntelligent` set to `true`, and `host_index` will be the index of that `HostId`. Note that the `HostId` of the target server will always have `isIntelligent` set to `true`, so there will always be a valid value for `host_index`. Similarly, as each application proxy processes the connection setup request message, it must increment the `host_index` to indicate the next `HostId` in the path that has `isIntelligent` set to `true`. This process ensures that the `host_index` always points to an "intelligent" host.

It should be noted that a client may maliciously create endpoints for a given `HostId` when building the `FIREWALL_PATH` service context. For that reason, firewall implementations should verify that the firewall policy allows a connection to the endpoint specified in the service context before opening a connection. This suggests that firewalls might provide a means of configuring policy such that certain protocol types (`PASSTHRU`, `NORMAL_SSL`, etc.) are allowed to only specific servers.

Once the connection has been established, the last intelligent firewall in the `FirewallPath` sends a `FIREWALL_PATH_RESP` service context in another `NegotiateSession` message (see section 3.8.2). The contents of the `FIREWALL_PATH_RESP` service context are described below.

```

module GIOP {
    // The FIREWALL_PATH_RESP service context contains a
    // FirewallPathRespContext structure
    const IOP::ServiceId FIREWALL_PATH_RESP = xx; // OMG Allocated

    typedef unsigned short FWReplyStatusType;
    const FWReplyStatusType NO_EXCEPTION = 0;
    const FWReplyStatusType SYSTEM_EXCEPTION = 1;
};
union FirewallPathRespContext switch (FWReplyStatusType) {
    case NO_EXCEPTION;
    case SYSTEM_EXCEPTION: SystemExceptionReplyBody exception;
};

```

In the normal, non-exception case, the FirewallPathRespContext is empty. If a firewall is unable to setup a connection, that firewall constructs an appropriate system exception for the failure and returns that value in the FirewallPathRespContext.

3.5 Connection Setup Message

Before a client and server can begin communicating, the client needs to send a connection setup message to the server that contains enough information for the firewalls along the path to the server to open the correct connections. To provide this functionality, a new GIOP message type is needed. The definition for this message is shown below.

```

module GIOP {
    enum MsgType_1_3 {
        Request, Reply, CancelRequest,
        LocateRequest, LocateReply,
        CloseConnection, MessageError,
        Fragment, // GIOP 1.1 addition
        NegotiateSession // GIOP 1.3 addition
    };

    struct NegotiateSessionHeader {
        ::IOP::ServiceContextList contexts;
    };
};

```

The NegotiateSession message is encoded as a GIOP header followed by a NegotiateSession header. The NegotiateSession message has no body.

The NegotiateSession message can be sent by either the client or the server, regardless of whether bi-directional GIOP is in use.

The NegotiateSession message contains a set of service contexts as defined for Request and Reply messages. However, the service contexts that are to be sent in a NegotiateSession message have certain restrictions. Namely, those service contexts can be restricted such that they can only be sent at certain times in the connection lifetime. Those periods are:

-
- connection setup period
 - session setup period
 - session established period

The connection setup period occurs while a client is attempting to setup a logical connection to the server. This period only occurs when the GIOP connection consists of a sequence of transport-level connections (when firewalls or bridges are in use). Only service contexts that are used for connection setup may be sent during this period.

The session setup period occurs before any GIOP Request or LocateRequest messages have been sent on a connection. During this period, the client and server can send any number of NegotiateSession messages. After all session negotiation has taken place, the client sends the first GIOP Request or LocateRequest.

After the client has sent the first GIOP Request or LocateRequest message, the connection remains in the session established period for the duration of the connection. Service context entries that are defined to be sent during this period must take into account that a connection may have been negotiated as bi-directional, and therefore no guarantees can be made regarding the presence or lack of outstanding requests.

Session information negotiated using the NegotiateSession message is only valid for the duration of the connection on which the NegotiateSession message is sent. No session information is maintained across connections. Furthermore, there exists only one session per connection. Therefore, service contexts defined for the NegotiateSession message must indicate whether or not that context can be sent multiple times, and if so, whether the information contained in subsequent contexts is additive or whether it replaces information sent in previous contexts.

3.6 ORB Policies

3.6.1 Path Selection Policy

When building the FIREWALL_PATH service context, the client ORB must be able to select endpoints from each of the HostId's in the FirewallPath element of the TAG_FIREWALL_TRANS component. As mentioned earlier, each HostId in the service context must have only one endpoint in order for the firewalls and server to determine what type of connection is being established. The client ORB selects endpoints based on the following policy.

```

module FWPolicy {

    typedef short PathSelectionPolicyValue;
    const PathSelectionPolicyValue NO_SSL = 0;
    const PathSelectionPolicyValue GATEWAY_SSL = 1;
    const PathSelectionPolicyValue END_TO_END_SSL = 2;
    const PathSelectionPolicyValue ANY_SSL = 3;

    // Allocated by OMG
    const CORBA::PolicyType PATH_SELECTION_POLICY_TYPE = xx;

    interface PathSelectionPolicy : CORBA::Policy {
        readonly attribute PathSelectionPolicyValue value;
    };
};

```

The PathSelectionPolicy is an ORB level policy. This policy may be overridden at the ORB level, or it can be overridden for specific objects that the ORB intends to invoke upon.

A PathSelectionPolicy type of NO_SSL will cause the client ORB to choose endpoints from the HostId's that do not use any type of SSL. This policy type might be used in a trusted environment, or when performance is a greater concern than confidentiality.

A PathSelectionPolicy type of GATEWAY_SSL will cause the client ORB to choose endpoints from the HostId's such that at least the channel to the outermost inbound firewall is protected by SSL. The outermost inbound firewall may not be capable of terminating an SSL connection, but the SSL connection may be terminated at an intermediary firewall or the server itself.

A PathSelectionPolicy type of END_TO_END_SSL will cause the client ORB to choose endpoints from the HostId's such that the client and server have an end-to-end security association via SSL. This means that all intermediary firewall HostId's must have an endpoint with ProtocolType PASSTHRU.

A PathSelectionPolicy type of ANY_SSL will cause the client ORB to choose endpoints from the HostId's such that the entire communication channel between the client and server is protected by SSL. This differs from the END_TO_END_SSL policy type in that an intermediary firewall may terminate the SSL connection and establish a new SSL connection with another inbound firewall or the server; i.e. NORMAL_SSL ProtocolTypes are allowed.

3.6.2 Path Insertion Policy

The server specifies a list of HostIds in the TAG_FIREWALL_TRANS component of the IOR. Depending on network topologies and relative locations of clients and servers, different clients may take different paths to reach the server. For instance, a client located in the same enclave as the server might connect directly to the server whereas a client on the internet will connect to the outermost inbound server-side firewall. Therefore, a client must have a policy about which HostId in the list of

HostIds to attempt to connect to first. The PathInsertionPolicy is used for this purpose. The PathInsertionPolicy is a client-side policy that is defined as and ORB-level policy that can be overridden for specific objects.

```
module FWPolicy {  
  
    typedef short PathInsertionPolicyValue;  
    const PathInsertionPolicyValue OUTSIDE_IN = 0;  
    const PathInsertionPolicyValue INSIDE_OUT = 1;  
    const PathInsertionPolicyValue NO_FIREWALL = 2;  
  
    // Allocated by OMG  
    const CORBA::PolicyType PATH_INSERTION_POLICY_TYPE = xx;  
  
    interface PathInsertionPolicy : CORBA::Policy {  
        readonly attribute PathInsertionPolicyValue value;  
    };  
};
```

A policy value of OUTSIDE_IN indicates that if the client detects a TAG_FIREWALL_TRANS component, the client should build the FIREWALL_PATH service context beginning with the HostId of the outermost-inbound firewall (this first HostId). If the client fails to connect to the server using that FIREWALL_PATH, the client should attempt to build a new FIREWALL_PATH context beginning with the next HostId in the list. The client should attempt to contact the server using sequentially increasing HostIds from the HostId list until a successful connection is established or all HostIds have been tried.

A policy value of INSIDE_OUT indicates that if the client detects a TAG_FIREWALL_TRANS component, the client should build the FIREWALL_PATH service context beginning with just the HostId of the server (the last HostId). If the client fails to connect to the server using that FIREWALL_PATH, the client should attempt to build a new FIREWALL_PATH context beginning with the previous HostId in the list (this includes the HostId of the server). The client should attempt to contact the server using sequentially decreasing HostIds from the HostId list until a successful connection is established or all HostIds have been tried.

A policy value of NO_FIREWALL indicates the the client should ignore the TAG_FIREWALL_TRANS component and only attempt to directly contact the server using the normal address information in the IIOP component.

3.7 Firewall and ORB Configuration

In order for a server to place information about the firewall path into an object's IOR, the server must know about the topology of the network. How that information is supplied to the server ORB is implementation dependent. Several solutions might include using a static configuration file or dynamically discovering the firewall topology from a configuration agent. Similarly, the firewall policy regarding which hosts are accessible from outside the enclave is implementation dependent.

3.8 *Firewall Traversal Algorithm*

A server ORB can determine if an object is to be accessed through a firewall via configuration information. When a server ORB determines that an object must be accessed through a firewall, the server ORB places a TAG_FIREWALL_TRANS component into the IOP profile of the IOR that contains the firewall path information as described earlier. In addition, the ORB must place some address and port number into the IOP profile itself as described in section 15.7.2 of the CORBA 2.4.1 specification. The address provided in the IOP profile shall be the address (or preferably DNS name) and port of the HostId of the server.

The client ORB determines that an object is accessed through a firewall by the presence of the TAG_FIREWALL_TRANS component in the IOP profile. The client ORB then prepares to send the connection setup message. First the client must prepare a FIREWALL_PATH service context, extracting the FirewallPath information from the IOR as described earlier. Recall that the connection setup message will only be sent if there is an “intelligent” firewall on the path to the server. This includes any outbound firewall proxies as described in the next section. Next the client must traverse any outbound firewalls as described in the next section.

3.8.1 *Outbound Firewall Traversal*

Outbound firewall traversal is typically simpler than inbound traversal due to less restrictive policies for outbound connections. In some cases, it may not be necessary to take any additional steps for outbound firewall traversal than to just open a TCP connection to the outermost inbound firewall on the server side. However, there are cases in which a firewall security policy will not allow arbitrary outbound connections, so there must be a means to handle those situations.

The approach used for outbound IOP connections is the same as the approach for other Internet protocols like HTTP or FTP. Namely, the client must be configured with an outbound IOP proxy to which it can send its connection requests. A client ORB must determine whether or not the client-side proxy is needed when making a connection, and if so, the client will open a connection to the proxy rather than the outermost inbound server-side firewall. How a client determines whether or not an outbound proxy is needed is an implementation issue, and other Internet protocol implementations can be used as a model for this implementation. Likewise, how the client ORB is configured with outbound proxy information is an implementation issue.

If an outbound proxy is within a nested set of enclaves, that proxy must also be configured with an outbound proxy. Since the proxy can determine the target of the connection setup request (the outermost inbound server-side firewall), the proxy makes the same decision as the client did in the previous step; i.e. determine if the outermost inbound firewall can be contacted directly. If not, forward the connection setup request to the next outbound proxy.

One implication of this model is that client-side firewalls cannot terminate SSL connections (NORMAL_SSL mode connections). If the connection from the client to the outermost inbound firewall is of type NORMAL_SSL or PASSTHRU then the

outbound firewall must have a connection type of PASSTHRU. If the outbound firewall is incapable of or unwilling to allow a PASSTHRU connection, then the outbound firewall must respond to the connection setup message with a NO_PERMISSION response in the FIREWALL_PATH_RESP service context and close the GIOP connection. If the connection from the client to the outermost inbound firewall is of type IIOP, then the outbound firewall may chose a connection type of IIOP or PASSTHRU. Again, if an outbound firewall is unwilling or unable to support those types of connections for a particular client and server pair, then the client should respond with a NO_PERMISSION response to the connection setup request and close the GIOP connection.

3.8.2 Inbound Firewall Traversal

As each inbound application proxy firewall receives the connection setup request, it must first locate its entry in the firewall path using the `host_index` field from the FIREWALL_PATH service context. The firewall then takes note of the connection type and examines the `HostId` of the next host in the path to the server. The firewall can then make a determination as to whether or not the requested connection is allowed. If the connection is not allowed, the firewall must return a NO_PERMISSION exception in the FIREWALL_PATH_RESP service context. Otherwise, the firewall opens a connection to the next host in the FirewallPath.

Before forwarding the connection setup message, the firewall must increment the `host_index` value. The `host_index` must indicate the next intelligent host in the FirewallPath, indicated by the `isIntelligent` field in the `HostId` having a value of `true`. In addition, the firewall must determine if it is the last “intelligent” firewall in the path to the server. If so, then the connection setup message should not be forwarded, rather the firewall should return a NO_EXCEPTION reply in the FIREWALL_PATH_RESP service context. If not, then the firewall forwards the connection setup request to the next host in the FirewallPath.

Eventually the entire logical connection from the client to the server is set up, or an exception has been returned, and all of the GIOP connections have been closed. If the connection was successfully established, then if the client or any of the firewalls need to begin an SSL handshake, the handshake process takes place using the TCP connections that had previously been established. After the SSL handshakes are completed, the firewall traversal processes is complete.

3.9 Callback Invocations

Though this specification provides a solution to the routing and location transparency issues for a variety of network topologies including those employing NAT, policy issues still exist that make it impossible for firewalls to be completely transparent to CORBA applications. Specifically, callbacks present a unique problem. Firewalls are usually configured to allow limited access to a limited set of well-known server addresses. Most hosts cannot be accessed from outside the enclave. CORBA clients

may sometimes also act as servers, requiring that invocations from hosts outside the enclave be allowed through the firewall. There are several possibilities for dealing with this problem.

First, if a CORBA client is not behind a firewall, then the server can make callback invocations without restrictions other than the server-side outbound connection policy. This is the simplest case. A related case is if the client is protected by a firewall, but the firewall policy allows inbound connections to that client. In this case, the client and server roles are reversed for the callback invocations, and the mechanisms previously outlined in this specification can be used for firewall traversal.

Second, if a CORBA client is behind a firewall then the server can make callback invocations on the client using bi-directional GIOP (see the accompanying document orbos/2001-06-05). Bi-directional GIOP allows a client to receive GIOP messages normally intended for a server and vice-versa. In this case, the server simply uses the channel opened by the client for callback invocations, and there is no need for additional algorithms.

Third, a CORBA client may be behind a firewall and a third-party might desire to make an invocation on an object managed by the client. Third-party in this case indicates that the client has not initiated a connection to that host, instead, that host received an IOR for a client object from some other host. Firewall policy may not permit inbound connections to that client, and a bi-directional GIOP connection is not possible because the client does not have a connection open to the third-party host. In this case it is not possible for a callback invocation to occur. Keeping this in mind, application developers must write their applications such that a client is given a reference to the third-party host instead. Then the client can contact that host, and callbacks can occur via bi-directional GIOP. This is the normal application development model for Internet applications, but not necessarily for CORBA applications. CORBA applications will need to be carefully designed in order to avoid third-party callbacks through firewalls.

3.10 Implications of SSL

3.10.1 Forwarded Identity

There are a number of additional problems with CORBA firewall traversal when SSL-protected connections are desired. First, SSL establishes a security association between the hosts that terminate the connection. SSL does not provide any means for delegation of authorization or identity. It may be desirable to have a firewall terminate an SSL connection in order to inspect the protocol for errors or to provide access control to the enclave. However, if a firewall terminates an SSL connection, the identity provided to the client or server will be that of the firewall, and not of the client or server as would be desired.

In order to work around this problem, a `FORWARDED_IDENTITY` service context is defined. The `FORWARDED_IDENTITY` service context provides a means for a firewall to pass the identity of the client or server to the other when the firewall

terminates the SSL connection (i.e. a NORMAL_SSL ProtocolType). When a firewall terminates an SSL connection, it may add a FORWARDED_IDENTITY service context to the first request message before passing the message to the server. Likewise, the firewall may add a FORWARDED_IDENTITY service context containing the server's identity to the first reply message before forwarding the message to the client. The FORWARDED_IDENTITY service context is defined as follows.

```
module Firewall {  
  
    const IOP::Serviceld FORWARDED_IDENTITY = 8;  
  
    typedef unsigned short IdTag; // OMG allocated  
  
    struct Identity {  
        IdTag tag;  
        sequence<octet> data;  
    };  
  
    typedef sequence<Identity> IdentityList;  
};
```

Each security mechanism that requires the passing of identities must have an IdTag allocated, and must define for that IdTag how the data in the Identity structure is encoded. IdTag values are allocated by the OMG.

The IdentityList is a sequence because sometimes other information needs to be passed. For, example, the "role" of the client (the client could have several roles, but the same cert), and/or the extracted Distinguished Name and password, key, etc. if other identity types are defined.

The following IdTags are currently defined: TAG_ID_SSL_CERT.
TAG_ID_SSL_CERT has the value of 0.

To pass SSL certificates in a FORWARDED_IDENTITY service context, the following is defined:

```
module SSLIOP {  
  
    const Firewall::IdTag TAG_ID_SSL_CERT = 0;  
  
    typedef sequence<octet> ASN_1_Cert;  
    typedef sequence<ASN_1_Cert> SSL_Cert;  
};
```

The data field of a TAG_ID_SSL_CERT should be encoded as a CDR encapsulation of an SSL_Cert. An SSL_Cert is a sequence (chain) of X.509 certificates, ordered with the sender's certificate first followed by any certificate authority certificates proceeding sequentially upward. An ASN_1_CERT is encoded using DER.

The certificates are not encoded as PKCS #6 extended certificates; they are X.509 certificates as defined by the ITU-T and IETF. Furthermore, the chain of certificates is not a SET of ExtendedCertificateOrCertificate as defined by PKCS #7; it is a sequence as defined by the CORBA CDR encapsulation rules.

The FORWARDED_IDENTITY service context cannot be used for irrefutable identification of the client or server. The FORWARDED_IDENTITY service context does not provide a cryptographically secure means of identifying a principal, rather it provides a means for the firewall to assert the identity of the client or server given that the receiving host trusts the firewall to act on the behalf of that client or server. If the firewall is not trusted to correctly identify the client or server, then the FORWARDED_IDENTITY information should also not be trusted.

3.10.2 Connection Setup

In order to eliminate differences in setting up SSL and non-SSL connections, the connection setup message is sent using IIOP without any SSL protection. This allows proxies to setup PASSTHRU connections because they will be able to determine the desired target of the invocation. Once the logical connection from client to server has been established (the connection setup request and reply have been seen by all of the firewalls), any SSL handshakes that need to occur take place over the pre-existing TCP/IP connections.

The implication this has on application firewalls is that an application proxy can always expect to see plaintext GIOP on incoming connections. It can then later proceed with an SSL handshake over that connection if necessary. However, if a server also receives the connection setup message, the server could see plaintext GIOP messages from a firewall or SSL handshake messages from clients within the server's enclave. Therefore the server will be unable to determine whether to process incoming connections as GIOP or SSL. For that reason, the connection setup message must not ever be forwarded to a server. That is, the last "intelligent" host in the firewall path must send the correct reply to the connection setup request. This means that if there are no application proxy firewalls, the client ORB must still open a TCP connection to the server, but the client ORB will not send a NegotiateSession message containing firewall path information to the server. Further, every application firewall must check if it is the last intelligent device on the path to the server, and if so, attempt to establish a TCP/IP connection to the next HostId in the FirewallPath, and then return a response to the connection setup request. Note that in considering whether or not to create or forward the connection setup message, a client must also consider any outbound proxies as described in section 3.8.1. An outbound proxy is also considered an intelligent device for this determination. Likewise, any outbound proxies will also have to consider other outbound proxies in the path to the server when making this determination.

Firewall Traversal Use Cases

The following example serves to illustrate most of the important features of this specification. In particular, the example shows how the traversal algorithm applies to a specific firewall configuration. The example is shown in Figure 4-1.

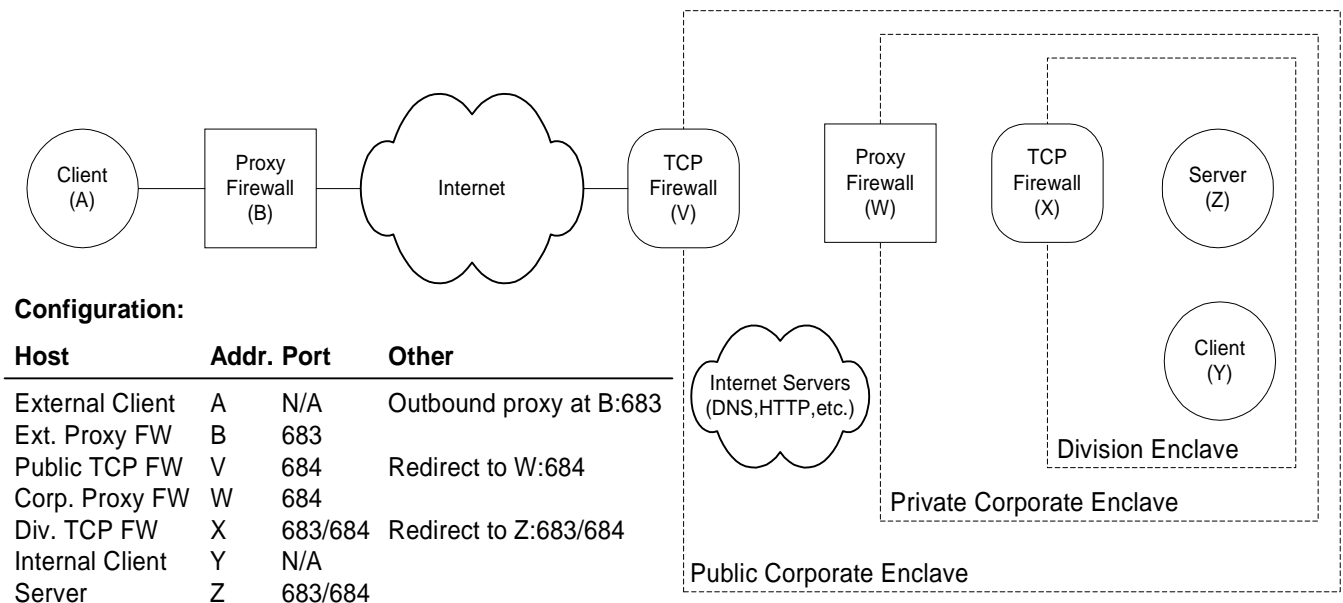


Figure 4-1 Firewall configuration for use case.

For this scenario, the IOR of an object on the server would have an IIOP profile shown in Figure 4-2. This IOR contains a TAG_SSL_SEC_TRANS component that could be used by the internal client to invoke on the server. In addition, the IOR contains a TAG_FIREWALL_TRANS component that allows external clients to traverse the firewalls in order to invoke on the server.

```

1: Profile ID:                0 (TAG_INTERNET_IOP)
2: Version:                  1.2
3: Host:                     Z
4: Port:                     683
5: Object Key:               "my_object"
6: # of Components:         2
7: Component 0:
8:   Component ID:          20 (TAG_SSL_SEC_TRANS)
9:   Target Supports:      0x7E
10:  Target Requires:      0x7E
11:  Port:                 684
12: Component 1:
13:  Component ID:         xx (TAG_FIREWALL_TRANS)
14:  # of Host IDs:       3
15:  Host ID 0:           //Refers to Corp. Proxy firewall
16:    Intelligent:      True    // using the public TCP firewall's
17:    Address:           V      // address via redirection
18:    # of Endpoints:    2
19:    Endpoint 0:
20:      Port:           684
21:      Type:           1 (NORMAL_SSL)
22:    Endpoint 1:
23:      Port:           684
24:      Type:           2 (PASSTHRU)
25:  Host ID 1:           //Refers to the division TCP
26:    Intelligent:      False   // firewall. Options for both SSL
27:    Address:           X      // and non-SSL ports
28:    # of Endpoints:    2
29:    Endpoint 0:
30:      Port:           683
31:      Type:           0 (IOP)
32:    Endpoint 1:
33:      Port:           684
34:      Type:           2 (PASSTHRU)
35:  Host ID 2:           //Refers to the server endpoints.
36:    Intelligent:      True    // One each for IOP/SSL and IOP
37:    Address:           Z
38:    # of Endpoints:    2
39:    Endpoint 0:
40:      Port:           683
41:      Type:           0 (IOP)
42:    Endpoint 1:
43:      Port:           684
44:      Type:           1 (NORMAL_SSL)

```

Figure 4-2 IOP profile of an object on the server containing firewall information

This firewall component contains three HostId's but there are actually four hosts, three firewalls and the server, that should be represented in this firewall component. The reason for this is that HostId 0 (lines 15-24) makes use of an optional optimization. Since the public TCP firewall is configured to automatically redirect connections to the

proxy firewall, HostId 0 actually represents the proxy firewall but the address in the HostId is the address of the public TCP firewall (line17). This optimization is possible in this case because there are no clients in the public corporate enclave that will make invocations on the server. If there were such clients, this particular firewall component would not have enough information for those clients to make an invocation on the server. Notice also that only NORMAL_SSL or PASSTHRU connections are allowed to the first firewall. This was configured by the administrator so that only secure connections are allowed from the internet.

HostId 1 (lines 25-34) represents the division TCP firewall. Since this node is a TCP firewall, the isIntelligent field is set to FALSE (line 26). Notice that in this HostId there are endpoints available for both IIOP and IIOP/SSL. The administrator configured this firewall so that internal corporate users could access the server without using SSL. Instead of placing the IIOP endpoints for HostId 1 and HostId 2 in the same firewall component as the SSL endpoints, the administrator could have put the IIOP endpoints in a separate firewall component altogether. This choice depends on the specific application. In this example, one use case will make use of both the SSL and non-SSL endpoints, so the administrator chose to place all of the endpoints in a single firewall component.

HostId 2 (lines 35-44) represents the server. A server will always have the isIntelligent field set to TRUE (line 36). This HostId also contains both an IIOP and an SSL/IIOP endpoint. Unlike in HostId 0, HostId's 1 and 2 actually contain the address of the division TCP firewall and the server respectively. The reason an optimization was not used in this case is that there is a client in the server's enclave that will also make invocations on the server. If the previously mentioned optimization were performed then the client may not have enough information to invoke on the server. In this case, since the client is in the same enclave as the server, the client could also use the information contained in the IIOP profile itself since the server HostId information must be the information provided in the IIOP profile. However, this example illustrates how the HostId optimization may cause the server to be unavailable to some clients.

The following use cases demonstrate the algorithm for choosing endpoints and opening a connection to the server.

4.1 Gateway SSL

Once the external client has received the IOR it must choose the endpoint from each HostId that it will use to contact the server, based on its endpoint selection policy. For this use case, the client has a GATEWAY_SSL PathSelectionPolicy. This policy indicates that the client must establish an SSL connection to the gateway. So based on that policy, the client ORB selects appropriate endpoints from the TAG_FIREWALL_TRANS component and builds a FIREWALL_PATH service context containing HostId's with those endpoints. That service context is shown in Figure 4-3.

```
1: Service ID:          xx (FIREWALL_PATH)
2: Host Index:         0
3: # of Host IDs:     3
4: Host ID 0:
5:   Intelligent:     True
6:   Address:         V
7:   # of Endpoints:  1
8:   Endpoint 0:
9:     Port:          684
10:    Type:          1 (NORMAL_SSL)
11: Host ID 1:
12:   Intelligent:    False
13:   Address:        X
14:   # of Endpoints:  1
15:   Endpoint 0:
16:     Port:          683
17:    Type:          0 (IIOP)
18: Host ID 2:
19:   Intelligent:    True
20:   Address:        Z
21:   # of Endpoints:  1
22:   Endpoint 0:
23:     Port:          683
24:    Type:          0 (IIOP)
```

Figure 4-3 FIREWALL_PATH service context for GATEWAY_SSL policy

In this case the ORB chose to make a NORMAL_SSL connection to the application proxy (line 10) and make non-SSL connections to the server (lines 17 and 24). It would also have been acceptable under the GATEWAY_SSL policy to make an additional SSL connection from the application proxy to the server, rather than to connect without SSL.

The client ORB is configured with an outbound proxy, so it first opens a TCP connection to the proxy (B:683). The client ORB then sends the NegotiateSession message with the FIREWALL_PATH service context. When the proxy receives this request message, it notes that this is an outbound connection, and attempts to determine what the destination of the message is. The proxy can determine this information by looking at the first HostId in the service context. In this case, that address is V:684. The proxy also notes that the connection type to that address is NORMAL_SSL, so the proxy will have to use PASSTHRU mode or return a system exception. In this case the proxy decides to use PASSTHRU mode and it opens a TCP connection to V:684, forwarding the original request message.

Recall that V:684 is actually the external address of the outermost TCP firewall. This connection is redirected to W:684. The proxy at that address receives the message and begins to parse it in the same manner that the client-side proxy did. But since this is an inbound connection, the proxy uses the host_index field to determine which HostId to examine. Since the host_index is zero, the proxy examines HostId zero. The proxy determines that this inbound connection must be terminated as a NORMAL_SSL

connection. It also determines that the next hop is X:683, and that this connection has a type of IOP meaning that no SSL will be used. Since this is an inbound connection, the proxy must also update the `host_index` field. Since the next hop is not an intelligent device, the proxy increments the `host_index` field by two to indicate the next `HostId` that has `isIntelligent` set to `TRUE`. The proxy then opens a TCP connection to X:683. But since the proxy is the last intelligent device other than the server, the proxy does not forward the connection setup request. Instead it must send a `NegotiateSession` message with a `NO_EXCEPTION FIREWALL_PATH_RESP` back to the client, indicating that the connection was successfully established.

The reply message is subsequently forwarded back through the firewalls to the client. At this point the connection has been established and the SSL handshakes must occur. In this case, the client begins an SSL handshake, and the server-side proxy firewall terminates the SSL connection. Once the security association between those two hosts has been established, normal GIOP communication can occur. All of the firewalls forward the messages, and the server-side firewalls that have access to the plaintext messages can examine the messages for correctness or perform access control on the requests if desired.

4.2 *End-to-End SSL*

This use case is very similar to the previous case except that the client ORB has an `END_TO_END_SSL` `PathSelectionPolicy`. This example only includes the changes from the previous example and does not give a full description of the traversal algorithm. Since the client ORB has an `END_TO_END_SSL` `PathSelectionPolicy`, the service context built for the connection setup request is different. This service context entry is shown in Figure 4-4.

In order to satisfy the `PathSelectionPolicy`, the client ORB chose a `PASSTHRU` endpoint for the proxy firewall (line 10), a `PASSTHRU` endpoint for the division TCP firewall (line 17), and a `NORMAL_SSL` connection to the server (line 24). Therefore, all of the intermediate hosts will only forward the GIOP messages, and the server will terminate the SSL connection with the client.

The firewall traversal algorithm is identical to the previous example except that the proxy firewall does not terminate the SSL connection because the endpoint type specified in the service context is `PASSTHRU` instead of `NORMAL_SSL`. Instead the server terminates the SSL connection.

These are just a couple examples of the many different configurations of firewalls that can be supported. However the process for establishing a connection is very similar in all cases.

1: Service ID: xx (FIREWALL_PATH)
2: Host Index: 0
3: # of Host IDs: 3
4: Host ID 0:
5: Intelligent: True
6: Address: V
7: # of Endpoints: 1
8: Endpoint 0:
9: Port: 684
10: Type: 2 (PASSTHRU)
11: Host ID 1:
12: Intelligent: False
13: Address: X
14: # of Endpoints: 1
15: Endpoint 0:
16: Port: 684
17: Type: 2 (PASSTHRU)
18: Host ID 2:
19: Intelligent: True
20: Address: Z
21: # of Endpoints: 1
22: Endpoint 0:
23: Port: 684
24: Type: 1 (NORMAL_SSL)

Figure 4-4 FIREWALL_PATH service context for END_TO_END_SSL policy

Conformance and CORBA Changes 5

An ORB implementation that is compliant with this specification must implement the data structures and algorithms presented in sections 3.1-3.7 and section 3.8. A firewall implementation that is compliant with this specification must implement the data structures and algorithms presented in 3.1-3.2, 3.4-3.5, 3.7, and 3.8-3.9. Though an implementation may not support the SSL transport it must be able to interpret the FIREWALL_PATH service context, regardless of the connection type, and act accordingly. Section 3.10 is optional, as is implementation of bi-directional GIOP.

This document supercedes the previously adopted CORBA firewall specification. In addition, OMG document orbos/2001-06-05 specifying changes to bi-directional GIOP supercedes the adopted specification for bi-directional GIOP. These specifications are not backwards-compatible with the previous specifications and they are intended to make it possible to create a functional protocol for the interoperation of ORBs and firewalls.

A.1 Firewall Module

```
module Firewall {

    typedef unsigned short ProtocolType;
    const ProtocolType IOP = 0;
    const ProtocolType NORMAL_SSL = 1;
    const ProtocolType PASSTHRU = 2;

    struct TransportEndpoint {
        unsigned short port;
        ProtocolType type;
    };

    struct HostID {
        boolean isIntelligent;
        string address;
        sequence<TransportEndpoint> endpoints;
    };

    // The component with ID TAG_FIREWALL_TRANS contains a
    // FirewallPath
    const IOP::ComponentId TAG_FIREWALL_TRANS = xx; //OMG Allocated

    typedef sequence<HostId> FirewallPath;

    // The FIREWALL_PATH service context contains a FirewallPathContext
    // structure
    const IOP::ServiceId FIREWALL_PATH = xx; // OMG Allocated

    struct FirewallPathContext {
        long host_index;
    };
};
```

```

        FirewallPath path;
    };

    const IOP::ServiceId FORWARDED_IDENTITY = 8;

    typedef unsigned short IdTag; // OMG allocated

    struct Identity {
        IdTag tag;
        sequence<octet> data;
    };

    typedef sequence<Identity> IdentityList;
};

```

A.2 Firewall Policy Module

```

module FWPolicy {
    typedef short PathSelectionPolicyValue;
    const PathSelectionPolicyValue NO_SSL = 0;
    const PathSelectionPolicyValue GATEWAY_SSL = 1;
    const PathSelectionPolicyValue END_TO_END_SSL = 2;
    const PathSelectionPolicyValue ANY_SSL = 3;

    // Allocated by OMG
    const CORBA::PolicyType ENDPOINT_POLICY_TYPE = xx;

    interface PathSelectionPolicy : CORBA::Policy {
        readonly attribute PathSelectionPolicyValue value;
    };

    typedef short PathInsertionPolicyValue;
    const PathInsertionPolicyValue OUTSIDE_IN = 0;
    const PathInsertionPolicyValue INSIDE_OUT = 1;
    const PathInsertionPolicyValue NO_FIREWALL = 2;

    // Allocated by OMG
    const CORBA::PolicyType PATH_INSERTION_POLICY_TYPE = xx;

    interface PathInsertionPolicy : CORBA::Policy {
        readonly attribute PathInsertionPolicyValue value;
    };
};

```

A.3 Changes to SSLIOP

```
module SSLIOP {  
  
    const Firewall::IdTag TAG_ID_SSL_CERT = 0;  
  
    typedef sequence<octet> ASN_1_Cert;  
    typedef sequence<ASN_1_Cert> SSL_Cert;  
};
```