CODO: Firewall Traversal by Cooperative On-Demand Opening

Sechang Son,[1] Bill Allcock,[2] and Miron Livny[1]
[1]*Computer Science Department, University of Wisconsin*
[2]*Mathematics and Computer Science Division, Argonne National Laboratory*
sschang@cs.wisc.edu, allcock@mcs.anl.gov, miron@cs.wisc.edu

Abstract

Firewalls and network address translators (NATs) cause significant connectivity problems together with their benefits. Many ideas to solve these problems have been explored both in academia and in industry. Yet, no single system solves the problem entirely. Considering diverse and even conflicting use cases and requirements from organizations, we propose an integrated approach that provides a suite of mechanisms and allows communicating peers to choose the best available mechanism. As an important step toward the final goal, we categorize previous efforts and briefly analyze each category in terms of use cases supported, security impacts, performance, and so forth. We then introduce a new firewall traversal system, called CODO, that solves the connectivity problem more securely than other systems in its category.

1. Introduction

A network address translator (NAT) [1] provides easy address planning as well as a solution to the IPv4 address shortage problem. Firewalls play a vital role in protecting networks and are ready to play an even more important role as the security headquarters of integrated security systems that generally include anti-virus checking, intrusion detection, logging, and content investigation.

These devices come at a price, however, notably nonuniversal (and asymmetric) connectivity to the Internet. Because of impaired network connectivity, applications written with the assumption of universal connectivity may not work. In addition, the development of new software becomes harder, and design may be inefficient because of the new constraint of Internet connectivity.

Grid [2], VoIP (Voice over IP), and P2P (Peer-to-peer) applications may be the areas damaged most by the connectivity problem because they are generally bidirectional in communication, interorganizational, huge in scale, and geographically distributed. Grid applications may not be able to share resources with the world, VoIP applications may miss calls, and P2P applications may be unintentionally free riding [3] because nodes behind a firewall or NAT cannot contribute to the file sharing, while they can generally download files from the world.

One possible solution to this problem is to redesign the Internet. However, making even a small change to the Internet is very hard. Blumenthal and Clark [4] discuss various factors that require reconsideration of the end-to-end argument [5], one of the most important design philosophies of the Internet [6]. In their paper, they state that firewalls and NATs are the major factors that require change, not only to the Internet edge, but also to its core. Their statement implies that to solve the problem, we may need to change the Internet in a fundamental way. Furthermore, a firewall's multilayer nature[1] will make an elegant solution difficult even with a new Internet architecture. Thus, we may have to concentrate on application layer approaches that require no change to the Internet. Furthermore, insights and experiences gained from those approaches may become a good basis for more fundamental approaches.

Many systems and ideas have been proposed and studied to solve connectivity problems, but no single system solves the problem entirely. Each system supports different types of use cases and has different characteristics in terms of security, performance, and so forth, still leaving holes that no system supports. We claim that this situation is not because those systems were poorly designed but because organizations use firewalls and/or NATs for different purposes with different levels of security, performance, deployability, scalability requirements, and so on, such that no single system supports all of them.

Considering the diverse and even conflicting requirements from different organizations, we need to have a *suite* of solutions such that all (major) functionality is supported by at least one component of the suite. Each solution ideally supports a partition of use cases with minimum overlap. We then need to have an integrated mechanism through which a solution is chosen for a communication channel

[1] Firewall functionality does not fit into any single layer of the OSI model because it needs to inspect packets from the IP header to the application payload.

through a negotiation between communicating parties. This is one of our major research goals.

As an important step toward this goal, this paper classifies existing and expected solutions based on use cases supported and analyzes each category in terms of security, deployability, and performance. Since we analyzes systems in a class as a whole, we do not provide any performance data or detailed security analysis of individual systems. We also introduce CODO (Cooperative On-Demand Opening), a new firewall/NAT traversal system that will become a major component of the integrated system. CODO enables communications over firewalls or NAT via on-demand creation/deletion of pinholes at the device. CODO provides efficient data communication but requires administrative power to be able to deploy. CODO's mechanism is the most secure within its class. It also provides a less secure but more easily deployable and efficient method of traversal, called the promiscuous mode.

Section 2 categorizes and analyzes firewall/NAT traversal approaches. Section 3 explains CODO in detail. Sections 4 and 5 analyze CODO's deployability and security, respectively. Section 6 presents performance data for CODO. Section 7 explains related research, and Section 8 briefly summarizes our work.

Throughout, we use the term "firewall" to refer both firewalls and NATs. The term "NAT" is used to specifically denote NAT.

2. Categorization of firewall traversal approaches

This section classifies firewall traversal approaches based upon the use cases that each class supports. Use cases are mainly determined by the questions of (1) what changes must be made to what components in order to use the system, (2) what security implications the system has, and (3) how efficient and scalable it is. The first question in particular plays an important role because in some situations an individual may not be able to make required changes to the components for political or technical reasons. For example, an employee may not be permitted to use a mechanism that requires a rule change to the company's firewall because she does not have administrator privilege on the device. The other questions are rather obvious. Efficiency and scalability generally vary from system to system even within a class. Therefore we discuss properties that systems in a class have in common.

2.1 Manual opening

Many organizations manually open their firewalls for a range of addresses for a trusted or well-protected application. If the application dynamically creates/closes sockets, then simple changes are made so that it uses the predefined range of addresses. Since only a few changes to firewall rules and simple or no changes to the application are required, this mechanism is easy to deploy with the administrator's privilege of the firewall. This mechanism must have little impact on the performance, if any.

Most applications, especially in Grid, VoIP, and P2P, dynamically creates/destroys sockets. The number of addresses being used by an application varies over time. The number of instances of the application may vary in some cases. Therefore, it is generally impossible, given an application, to know a priori what addresses are needed and how long they must be opened. Even if we know the exact range of addresses that must be opened for the application, that range gives us only the upper limit in most cases; and, for most of its lifecycle, the application may use fewer addresses. Therefore, manual opening almost always results in more addresses opened than the application actually uses. These extra openings can be exploited to attack the network. Therefore, no matter how well prepared for attacks the intended application is, this mechanism is vulnerable to attacks to the network. This should not be used unless no other mechanism is available.

2.2 Anonymous approaches

Systems that fall in the anonymous category exploit general practices of organizations and require no interaction with firewalls. Such systems use the fact that most networks allow, for example, outbound connections and Web traffic. We call this class of systems *anonymous* because firewalls and traversal systems do not (or need not) know each other. GCB (Generic Connection Brokering) [7], Gnutella [8], STUN (Simple Traversal of UDP through NATs) [9], TURN (Traversal Using Relay NAT) [10], and the mechanisms that use Web traffic to get through firewalls can be classified as anonymous.

In order to use an anonymous approach, end applications must be changed, and sometimes third-party agents that help the end applications to get through firewall must be installed outside or inside the network. Since end application and agents are not visible from firewalls, [2] this class can be easily deployed. These approaches are considered "cheating," however, and many administrators will try to prevent them. Firewall companies are also developing features

[2] Traversal systems in anonymous approaches generally need to know the fact that one or more firewalls exist in the communication path.

to defeat anonymous approaches. Thus, these systems may become less deployable over time. This class also has the undesirable security property that attackers and/or dissident employees can deceive firewalls by using the tricks that systems in this class invented. Therefore, this class is analyzed as highly deployable at initial time but gradually less deployable over time.

Nevertheless, systems in this class support unique use cases and are useful when communicating parties do not have a control on a firewall in the path. For example, many cable access providers use NAT in front of their entire access network. The end user will probably not have a control relationship with the NAT [9]. Any other class can support this kind of use case. It is unlikely that every firewall may become securely detectable and controllable in the near future. Therefore, we still need to invent systems of this class that invite as few attackers as possible. GCB is a good example that satisfies this property because it uses the fact that outbound connections are inherently more secure than incoming ones and because only (dissident) insiders (but not attackers outside the network) can use the idea to bypass their firewalls.

2.3 Cooperative control

Systems in the on-demand opening class and relay-based systems each are classified as *cooperative* approaches because communications into and/or out of a network are controlled by the cooperation between the firewall of the network and one or more agents. An agent can be in the form of a part of the firewall, a daemon process running apart from applications and the device, a library linked with applications, or a combination of the above.

2.3.1 On-demand opening

The basic idea of the on-demand opening category is that a firewall and agents establish a trust relationship and the device opens pinholes at the agents' request. Figure 1 shows a typical topology of systems in this category. A firewall has one or more trusted agents. An agent dynamically controls the firewall on behalf of applications. Once a pinhole is created at the firewall, the designated applications communicate through it. Agents must understand the applications' intentions to be able to know when pinholes must be created or closed. One common approach is to investigate the applications' payload. Another approach is to have applications explicitly ask for the creation or deletion of pinholes to agents. Of course, an appropriate security mechanism must be used by the agent before it decides to open holes for applications.

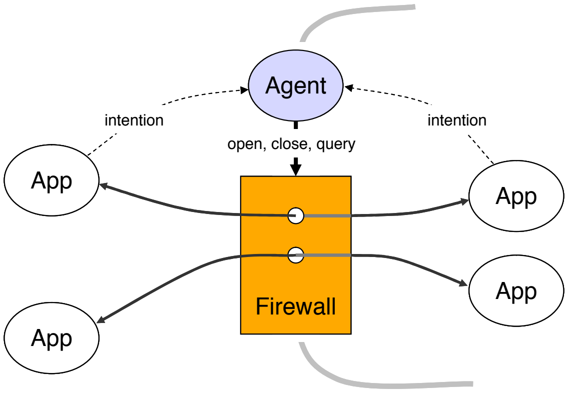

Figure 1: Topology of on-demand opening systems

ALG (Application Level Gateway), DPF (Dynamic Port Forwarding) [7], MIDCOM (Middlebox Communication) [11], RSIP[3] (Realm Specific IP) [12, 13], UPnP (Universal Plug and Play) [14], and CODO may be classified as on-demand opening methodologies.

Since agents effectively control the firewall, systems in this category must be deployed by the administrator rather than the application service providers or end users. This requirement may limit the deployability of the systems in this class. Another barrier to deployment is that the firewall must provide well-defined interfaces so that agents can ask for, by using those interfaces, the creation, deletion, list, and query of pinholes.

The security of the systems in this class is determined mainly by how narrowly pinholes are opened. All previous systems create a pinhole with the address of either the client or the server. Pinholes created by those systems are wider than those created based on both the client's and the server's addresses. Full cone NATs [9] are considered less secure than symmetric NATs for the same reason. On the other hand, CODO allows agents to create pinholes for designated (client, server) pair so that other clients may not talk to the server through the hole. Another important factor is the security model being used by a firewall to trust agents and by the agent to authenticate/authorize application's request for opening. One common security problem of the class is that any client or server can exploit, with address spoofing, the pinholes made for an authentic client and/or server. If this is considered a security problem, then organizations must consider using relay-based mechanisms (explained in the next section).

[3] We assume that RSIP creates pinholes as a part of address leasing.

Once pinholes are created, data communication occurs through those holes. Since agents do not need to participate in the data communication, there may be no extra overhead for data communication, thus making mechanisms in this class efficient.

2.3.2 Relay-based approaches

In a relay-based system, a firewall trusts one or more routers or relaying agents and allows communications from and/or toward them. Data communication as well as connection setup is relayed by those agents. Since firewalls bypass (part of) security checking for packets to/from agents, they must filter packets on behalf of the devices, usually via authentication/authorization. Thus we may assume that firewalls delegate some or all of security checking to those agents. SOCKS [15], tunneling technologies, and overlay routing such as WS-Routing [16] can be classified in this category. Figure 2 shows a typical topology of systems in this category.

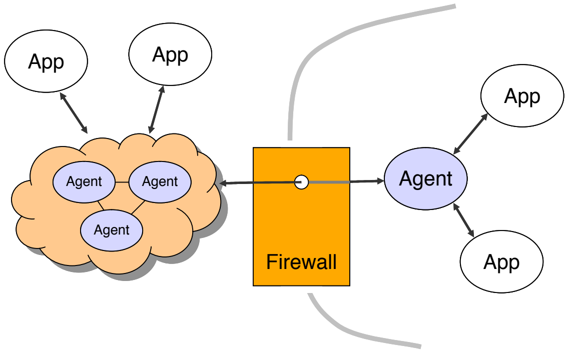

Figure 2: Topology of relay-based systems

We note that an approach is not classified as relay-based simply because it uses relaying agents to traverse firewalls. GCB and TURN, for example, are not classified as relay-based mechanisms, although their agents do relay traffic. They exploit the fact that most organizations allow outbound connections, rather than cooperating with firewalls. The same argument can be applied to overlay routing mechanisms. If a WS-routing system, for example, uses the fact that Web traffic is allowed by most firewalls, then it should be considered as an anonymous approach.

Relay-based approaches can avoid the IP spoofing problems of the on-demand opening class. If a network is configured so that its firewall may not be deceived by IP spoofing with the addresses of the relaying agents of the network[4] and the agents use strong security mechanisms, only intended client, server, or next hop agents can communicate over the firewall. Attackers may be able to send packets (with IP spoofing) to an agent, but they will not be passed to internal nodes or next hop agents because they will fail security checking by the agent.

Like the on-demand opening class, relay-based systems must be deployed by the firewall administrator. The deployment is easier than for the on-demand opening class, however, because the firewall does not have to provide well-defined interfaces for dynamic control of it. Using the ad hoc mechanism of the firewall at hand, administrators need only add a few rules so that communications to/from the agents may be allowed.

With a system in this class, however, not only the connection setup but also all data communications must be relayed via agents. Hence, the performance of these approaches is expected to be slow. Also, there exists the possibility that many features such as reliability and flow control, which underlying network already provides, must be implemented again at higher layers. These duplicated functionalities may make those systems even slower, as well as requiring significant duplicated effort.

2.4 Summary

This section categorizes firewall traversal systems and analyzes each category. Manual opening should not be used, if possible, because the approach is insecure and hence nullifies the effect of having firewalls. Anonymous approaches also are insecure and degrade deployability; however, they support unique environments that no other systems can support. On-demand opening can be implemented very securely and efficiently; however, it supports only a limited environment that allows dynamic control of firewalls. Relay-based opening supports more environments than does on-demand opening and can be implemented most securely; however, it tends to be heavyweight and slow.

The most important message of this section is that no single system can satisfy everybody with diverse environments and different criteria.

3. Cooperative On-Demand Opening

CODO is a firewall traversal system in the on-demand opening class. It is intended to be a major component of the integrated system that will allow peers to choose one mechanism, through a negotiation,

[4] For example, if agents are attached to the firewall via a separate network interface, the network guarantees that no node either inside or outside the network can send packets with agents' addresses to the firewall.

from a list of mechanisms possibly in different categories.

CODO can control outbound as well as inbound connections. In CODO, connections into or out of a network are enabled through the cooperation of a firewall, a FA (firewall agent), and a CL (client library) linked with applications. Figure 3 shows a typical CODO topology.

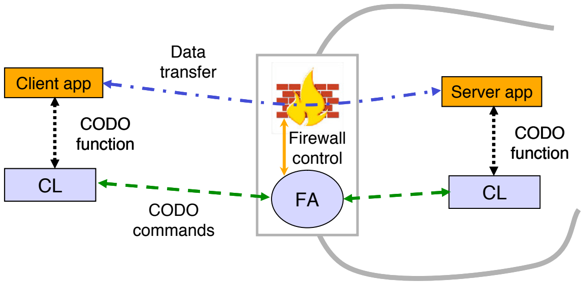

Figure 3: CODO topology

The FA installed on the headnode of a network dynamically creates and closes pinholes at the firewall for CODO applications that want to communicate with the world. Through secure channels, the CL interacts with the FA by exchanging CODO commands on behalf of the application. The application uses CODO services by calling well-defined CODO functions that the CL provides. Since the CL provides well-defined APIs, applications can easily be made CODO-aware.

When a server running behind a firewall creates a socket, the information about the server socket is registered to the FA of the network. If a client wants to connect to the server, its CL contacts the FA and asks for permission to come in the network. The FA then creates a pinhole so that a connection can be made from the client to the server. When the pinhole is not needed anymore, either the client's or the server's CL notifies the FA so that it may close the hole. If the client is also behind a CODO-enabled firewall that does not allow outbound connections, then its CL contacts the FA of the client network to get permission to go out, before it contacts the server's FA.

CODO is one of the most secure systems in the on-demand opening category because pinholes are made just for the (client, server) address pair and they are kept open just for as long as necessary. How CODO achieves this goal is explained in the following sections.

3.1 Connection procedure

We describe here how a connection is established by using CODO.

3.1.1 Server binding

In order to be able to accept connections from outside, server sockets behind a firewall must be locally bound, registered to the FA of its network, and officially bound.

Local binding is nothing new. Just as with regular binding, a socket is bound to an address. Through the local binding, an (IP, port) pair, called the local address, is assigned to the socket.

Since inbound connections are arranged by the FA of the network, enough information about a server socket must be kept in the FA. This is done by a process called *registration*. After a server socket is bound to a local address, the server's CL sends a registration request with the local address and the type of the socket. After authentication/authorization and the official bind (explained shortly), the FA records the information sent by the CL and other information that it collects from the official binding process.

Official binding is the process of assigning the official address to a server socket. To explain what an official address is, we need to explain *address leasing*, first. This is a major idea of NATs. Because a private address is not routable in the public network and may be used by multiple endpoints, a NAT device rents a public address (or a set of public addresses) to a socket (a host) and dynamically translates between the socket's real address and the leased public address as packets flow. As a result, a socket behind a NAT has two addresses: one for intratraffic and the other for traffic outside the network. Note that we call the real address the local address in this paper. Since the Berkeley socket API allows only one address per socket, the local address is known to the owner application that creates and binds the socket, while the leased address is known to the peer.

If a server binds a socket to a dynamic address, it has to pass the address of the socket to a client through an out-of-band mechanism. Therefore, what address is known to the owner is important for the connection setup. We define the address of a socket that is known to the owner application as the *official address* of the socket.

When an FA receives a registration request with a private local address, it finds a public address and rents the address to the server socket. This leased address becomes the official address of the socket. Of course, if the local address is public, then the local address becomes the official address without address leasing. As a successful response to the registration request, the FA sends the official address to the CL of the server application.

When an application asks the address that a CODO socket is bound to, the CL returns with the official address instead of the local (real) address. Thus, the local address of a CODO socket is hidden from the application. CODO requires that applications not call the binding function with a private IP address. If an application calls the function with a private IP address, then it already knows that the socket is bound to that IP. Therefore, the IP part of the official address is automatically decided by the application, and CODO does not have any freedom to assign a public address. In this case, CODO assumes that the application does not want to accept connections from clients outside the network. Yet, applications can call the binding function with a specific port number because CODO assigns an official address with the same port number, if requested by the application.

3.1.2 Connection arrangement

This section explains how CODO connections are actually made. As an example, we use a TCP connection from a client to a server. The client’s network does not allow outbound connections and the server’s network does not allow inbound connections from the client’s network. The setting is shown in Figure 4. Note that this is the most complex and hardest situation. It is obvious which may be bypassed for simpler settings. (UDP connections will be explained in Section 3.1.3.)

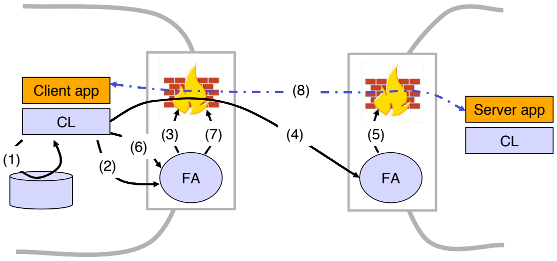

Figure 4: Firewall-firewall connection

We assume that the FA of the server’s network has the information about the server socket through the binding process described in Section 3.1.1. We also assume that the client knows the official address of the server socket via an out-of-band mechanism. For example, RTP (Real-time Transport Protocol) [17] addresses for media stream can be passed, using SDP (Session Description Protocol) [18] messages, as a part of a SIP (Session Initiation Protocol) [19] session establishment. A connection is established through the following steps:

(1) The client’s CL discovers the connection information such as whether a direct connection is possible to the server, what FA must be contacted if direct connection is impossible, and whether outbound connections are allowed.

(2) If outbound is not allowed, the client’s CL contacts its FA and asks for permission to contact the server's FA.

(3) The FA of the client network creates a pinhole for the address pair (client, server's FA).

(4) The client’s CL contacts the server's FA and asks for inbound connection to the server. The CL sends client’s address with the request.

(5) The server’s FA creates a pinhole for (client, server) address pair and replies with the address that the client can directly send packets to. One little complication occurs when the client is behind a NAT. In this case, the client’s CL reserves a NAT binding and asks for an opening for the address pair (NAT binding, server).

(6) The client’s CL contacts its FA again and asks for outbound permission toward the address that the server's FA replied.

(7) The client’s FA creates a pinhole for the address pair in the request.

(8) The client and the server communicate through pinholes created.

How the CL discovers the connection information in Step 1 is an important issue. [5] In the current implementation of CODO, each node has a table that must be manually configured. Obviously this solution is not scalable, although the table should not be very big because we can have one entry per a network. A registry-based mechanism would be a better solution. However, the connection information must be determined based not only on the server’s address but also on the client’s address because, given a server, different FAs and policies may have to be used for different clients in different networks. Therefore, the registry may become very big and tend to change relatively often. Managing the registry to a reasonable size and distributing its entries among nodes is an open question and requires further research.

Connection establishment within a private network also needs help from CODO agents. A client within the same private network as a server cannot make a direct connection with the official address of the server. In this case, the client’s CL asks the FA of the network for the server’s local address and then makes a direct connection to it. No pinholes are made for intraconnections.

3.1.3 UDP connections

Although there is no connection setup in UDP communications, we can loosely define a UDP session

[5] Actually this is not a CODO-specific issue. Any firewall traversal mechanism must answer this question.

as a set of UDP messages that are allowed or rejected as a whole by stateful firewalls. More precisely, it is a series of UDP messages from a client to a server such that each message passes a firewall before a predefined timeout from the previous one.

In order to support UDP sessions over firewalls, a CL maintains a mapping table. Each entry in the table maps the official address, *addr1*, of a peer to a real address, *addr2*. An entry (addr1, addr2) means that UDP messages addressed to addr1 by the application must be sent to addr2. When the application wants to send a UDP message to an address, the CL searches the mapping table for the address. If an entry is found, then it refreshes the timeout value of the entry and sends the packet to the mapped address. Otherwise, a procedure similar to the TCP cases in Section 3.1.2 is performed. If the procedure succeeds, then the sender's CL sends the message to the address that the receiver's FA returned and then creates an entry with the official and the real addresses. Entries not referred to for a while are deleted.

3.1.4 Promiscuous mode

CODO opens pinholes in a very secure way such that packets only between a designated sender and receiver can get through firewalls. To do this, it requires a fair number of interactions between CLs and FAs. Another drawback of CODO is that it requires that both the client and the server be able to speak the CODO protocol. This requirement may result in server's being unable to communicate with legacy clients that cannot become CODO-aware.

In order to address this issue, an FA can operate in a less secure *promiscuous* mode. In this mode, the FA creates a pinhole as part of the server binding process. Since the client's address is not available at the binding time, the pinhole is made solely based on the server's address, allowing any client to get. Therefore, clients need not use the CODO mechanism to connect to the server. Clients that do use CODO mechanisms still can connect to a server by contacting an FA in the promiscuous mode. CODO in the promiscuous mode is similar to DPF, RSIP, and UPnP.

3.2 Fault tolerance

Successful connection depends on the reliability of FAs. Nevertheless, applications should continue to work with a limited ability in the event of FA failure. During FA downtime, CLs operate as if those FAs do not exist. For example, if a server's CL cannot contact the local FA at binding time, it degrades the socket as a regular one and bypasses all CODO mechanisms. If a client's CL cannot contact the server's FA, it tries a direct connection to the server.

If an FA recovers from its failure, applications that were affected by the failure should switch to full functional mode. To achieve this goal, we design FAs to maintain soft states so that they can recover their states by receiving socket information from server CLs. The CL periodically tries to contact the failed local agent. If successful, it sends the current state of sockets to recover the FA's state and upgrades the sockets to CODO sockets by doing whatever it would have done if the FA had not failed.

Pinholes created by an FA survive at the firewall over the failure and recovery of the FA. Some pinholes may become unnecessary during the downtime of the FA because, for example, a socket for which a pinhole was created is closed while it is down. These leftover openings should be deleted for security. If firewalls supported timeout on pinholes, then the process would be easier because leftover pinholes would be deleted after timeout. Unfortunately, the firewalls we targeted do not support this feature.

For garbage collection of leftover openings, each FA records a snapshot of holes that it is in charge of in a persistent place and, when it recovers from a failure, it deletes all pinholes recorded. Certainly this blind flush will delete necessary rules as well. The necessary rules are recreated as a part of state recovery when server CLs send socket information they own.

3.3 Implementation

CL is implemented as a library and as a layer between the application and the kernel, as depicted in Figure 5. Applications use CODO socket calls to create a CODO socket, bind it to an address, connect to a server, accept a connection from a client, and so forth. The CL provides some file system calls so that applications may duplicate socket descriptors, make a socket non-block, and multiplex multiple file descriptors, including CODO sockets. The CL also has a few functions for process control, such as CODO_fork and CODO_execve. These are mainly for inheriting open sockets to child processes. All CODO calls have the same APIs as their regular counterpart. This strategy is intended to facilitate application programming and enable dynamically linked applications to use CODO without rewriting.

FA is implemented as a daemon running on a firewall machine. It uses Linux netfilter API to create, delete, and list pinholes. Although CODO currently supports only firewalls based on the Linux netfilter, it interacts with firewalls through an abstraction layer that defines necessary firewall functions to dynamically control it. Therefore, any firewall with those functions can be easily supported.

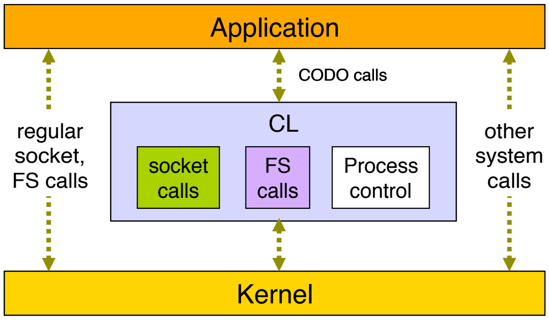

Figure 5: CL implementation

For authentication and secure communication between CLs and FAs, we designed a variant of SSL/TLS [20]. One reason for our modification was so that bidirectional authentications can be done with fewer message exchanges than SSL/TLS. More important, we wanted to enable CODO to return to the application a secure channel. Hence, we needed a security system that can do authentication and integrity/encryption for UDP communications in the face of possible reordering and losing of messages.

4. Deployment

CODO is *selfish deployable*. Each organization can independently decide whether to use CODO mechanisms. Moreover, each organization can determine the security level to apply for authentication and authorization and can decide the level of control (e.g., whether outbound connections are controlled or allowed). No global or bilateral agreement among communicating organizations is required.

CODO is also *application-by-application* deployable. Organizations can deploy it only to necessary and proven applications. The selfish and application-by-application deployable nature of CODO makes it easy to deploy gradually.

5. Security considerations

To analyze how CODO affects the security of organizations using firewalls, we need to consider for what purposes such organizations use firewalls and which of these are impacted by CODO.

- IPv4 address shortage and easy network management

One of the most important reasons for using NAT is that IPv4 addresses are being sold out. Another reason may be easy address planning and network management. Obviously CODO has no adversary effect on this goal. In the light of this goal, connections being blocked are a side-effect. CODO removes this undesirable side-effect.

- Address concealment

A private network is considered secure at some level because the addresses of hosts are hidden from outside. In CODO, private addresses are still hidden from the public network.

- Inbound connections

Organizations block inbound connections for many reasons. We believe that the followings are major ones.

1. Unprepared servers can be protected from attacks.

   Unlike clients, servers are generally wide open to any client. Many legacy servers are not well prepared for attacks. By blocking inbound connections, those servers are protected from attacks launched outside the network.

   CODO makes servers accessible from outside and may introduce security problems. However, two contending interests operate here. One is to protect nodes from attacks by closing the door, and the other is to collaborate with the world by opening the door. If an organization can totally sacrifice one for the other, the solution is easy. In most cases, however, one goal must be achieved with the other sacrificed as little as possible.

   CODO is a near-optimum solution to this problem. CODO opens a firewall only when there is an authenticated client and server pair. CODO also opens holes as narrow as possible.

2. Management can keep employees from running undesirable services.

   By blocking inbound connections to arbitrary addresses, an organization can keep members from running undesirable or insecure services. Since CODO uses a strong security mechanism, only authorized services can register server sockets to their FAs and therefore accept connections from outside of their networks.

- Outbound connections

Outbound connections are blocked, although not as often as inbound cases are, for similar reasons. Since CODO allows outbound connections only from authenticated clients, organizations can achieve the same level of control and security.

- Third-party security

Firewalls provide chokepoints of traffic so that other protection systems such as anti-virus, logging, and IDS (Intrusion Detection System) can easily be placed. CODO does not change the flow of traffic and, thus, has no effect on this goal.

6. Performance measurement

We used two private networks behind Linux NAT boxes. One network has 24 nodes connected to the NAT and each other via 100M Ethernet. The other network has two nodes connected via 10M Ethernet. The two networks are connected via department network. Neither inbound nor outbound connections are allowed in the networks.

Using a test suite that we wrote, we measured times for socket binding, private-private connection setup and data transfer, and intranetwork connection and data communication. For data transfer, we measured the total amount of time that 100 messages of 1,000 bytes long are sent by a client and then echoed back by a server. Table 1 shows the results for TCP and Table 2 for UDP. In order to indicate the overhead of CODO, the table also has numbers for regular sockets with manual openings. For private-private measurements, we used a client in the smaller network and a server in the other. For intranetwork communication, we used a client and a server both in the bigger network. Each test was performed multiple times over several days to reduce the effect of network fluctuation. The numbers in the tables are all in units of microseconds and include times taken for authentication and encryption/integrity. We used X.509 (RSA) public key for authentication and session keys establishment. SHA-1 and 3DES were used for integrity and encryption, respectively.

Table 1: TCP results

	Bind	Inter		Intra	
		Conn	Data	Conn	Data
CODO	48720	63433	1957	12426	541
Reg.	16	833	2015	243	503

Table 2: UDP results

	Bind	Inter		Intra	
		Conn	Data	Conn	Data
CODO	48854	63972	2030	12371	608
Reg.	8	8	2025	14	500

The tables show that CODO overhead is fairly large for both binding and connection setup. However, almost no overhead for data communication was observed. Considering the security mechanisms used and the number of interactions between CODO agents, the overhead for binding and connection is not surprising. We profiled CODO operations by running instrumented CODO codes. The result showed that about 87% of the binding time was spent on security handshakes between the CL and FA, and 60% of the connection time was spent on security handshakes.

The testing environment is very restrictive. In real settings, one might not use some of the CODO mechanisms. For example, most networks allow outbound connections. In such cases, private-private connections may become a little less efficient than intraconnections. If a network trusts insiders, then one can skip security mechanisms with the local agent.

7. Related research

Many firewall traversal systems have been developed. GCB, Gnutella, STUN, and TURN can be classified as anonymous systems. All of them basically use the fact that most firewalls allow outbound connections. GCB is considered the most versatile anonymous approach so far. Gnutella does not support firewall-to-firewall connections. STUN supports only UDP over full cone firewalls. TURN does not support the situation in which a server needs to accept connections from multiple clients. ALG, DPF, MIDCOM, RSIP, and UPnP are on-demand opening systems. These systems are as secure and efficient as CODO in the promiscuous mode. SOCKS, VPN, and overlay routing systems can be classified as relay-based systems.

More fundamental approaches to solve connectivity problems have been proposed, too. TRIAD [21] and IPNL (a NAT-extended Internet architecture) [22] propose a new layer between TCP/UDP and IP. They provide elegant solutions to NAT traversal, but they cannot be used for firewall traversal.

IPv6 [23] is beginning to be widely deployed. It provides enough address space and enables easy network management. Thus, it solves most problems that NAT tries to solve. However, it is still questionable whether IPv6 can replace NAT completely. Furthermore, firewalls will certainly exist after the full deployment of IPv6.

8. Conclusion

This paper categorizes firewall traversal mechanisms into four classes and analyzes systems in each class based on essential features. The categorization may not be perfect, and new approaches in the future may require that new classes be added. However, the categorization does catch important characteristics of (at least) existing systems and analyzes important implications. Organizations may use the results to choose a traversal system and configure their network to use the system securely and efficiently. More important, the analysis underscores the need for integrated systems.

This paper also introduces CODO, a firewall traversal system. CODO supports firewall traversal most securely than do other systems in the on-demand opening class. In the promiscuous mode, CODO also provides the same security and efficiency as the other systems in the on-demand opening class.

The security implications are discussed in terms of the goals of using firewalls that may be broken by CODO. We believe that this methodology can be a good reference for the security analysis of any firewall traversal system.

Acknowledgements

This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38.

References

[1] K. Egevang, P. Francis, "The IP Network Address Translator (NAT)," *IETF RFC1631* May 1994.

[2] I. Foster, C Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Intl. Journal of Supercomputing* Applications 2001.

[3] E. Adar and B. A. Huberman. "Free Riding on Gnutella." *First Monday*, 5(10), 2000.

[4] M. S. Blumenthal and D. D. Clark. "Rethinking the design of the Internet: The End-to-End Argument vs. the Brave New World," A*CM Transactions on Internet Technology,* 1(1), 2001.

[5] J. Salter, D. Reed, D. D. Clark. "End-to-End Arguments in System Design," *ACM Trans. Comput. Sys.*, 2(4), 1984, pp. 277–288, 1984.

[6] D. D. Clark, "The Design Philosophy of the DARPA Internet Protocols". *Proc. of the ACM SIGCOMM'88, in: ACM Computer Communication Reviews,* 18(4), 1988, pp. 106–114.

[7] S. Son,M. Livny, "Recovering Internet Symmetry in Distributed Computing." *Proceedings of the 3rd International Symposium on Cluster Computing and the Grid*, Tokyo, Japan, May 2003.

[8] "The Gnutella Protocol Specification v0.4 Rev. 1.2," http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

[9] J. Rosenberg, J. Weinberger, C. Huitema, R. Mahy, "STUN – Simple Traversal of User Data Gram (UDP) Through Network Address Translators (NATs)," *IETF RFC 3489*, March 2003.

[10] J. Rosenberg, R. Mahy, C. Huitema, "Traversal Using Relay NAT (TURN)," *Internet-Draft*, July 2004.

[11] P. Srisuresh et al., "Middlebox Communication Architecture and Framework," *IETF RFC 3303*, Aug. 2002.

[12] M. S. Borella, G. E. Montenegro, "RSIP: Address Sharing with End-to-End Security," *Special Workshop on Intelligence* at the Network Edge, San Francisco, 2000.

[13] M. Borella, J. Lo, D. Grabelsky, G. Montenegro, "Realm Specific IP: Framework," *IETF RFC 3102*, July 2000.

[14] Universal Plug and Play. http://www.upnp.org

[15] M. Leech, M.Ganis, Y. Lee, R. Kuris, D. Koblas, L. Jones, "SOCKS Protocol Version 5," *IETF RFC 1928,* March 1996.

[16] H. F. Nielsen, S. Tahtte, WS-Routing, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-routing.asp

[17] H. Schulzrinne, R .Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," *IETF RFC 3550,* July 2003.

[18] M. Handley, V. Jacobson, "SDP: Session Description Protocol," *IETF RFC 2327,* April 1998.

[19] M. Handley, H. Schulzvinne, E. Schooler, J. Rosenberg, "SIP: Session Initiation Protocol," *IETF RFC 3261,* June 2002.

[20] T. Dierks, C. Allen, "The TLS Protocol," *IETF RFC 2246*, Jan. 1999.

[21] D. R. Cheriton, M. Gritter, "TRIAD: A New Next Generation Internet Architecture," March 2000. http://www-dsg.stanford.edu/triad/triad.ps.gz.

[22] P. Francis, R. Gummadi, "IPNL: A NAT-Extended Internet Architecture," *SIGCOMM'01*, Aug. 27, 2001.

[23] S. Deering, R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," *IETF RFC 2460*, Dec. 1988.

THIS PAGE NOT TO BE PRINTED WITH THE ARTICLE

The submitted manuscript has been created by the University of Chicago as Operator of Argonne National Laboratory ("Argonne") under Contract No. W-31-109-ENG-38 with the U.S. Department of Energy. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.