AFT Working Group
Internet Draft
Expires in six months

D. Chouinard
Intel Corporation
November 20, 1997

SOCKS V5 UDP and Multicast Extensions to Facilitate Multicast Firewall Traversal

draft-ietf-aft-mcast-fw-traversal-01.txt

Status of this Memo

Abstract

This proposal creates a mechanism for managing the ingress or egress of IP multicast through a firewall. It does this by defining extensions to the existing SOCKS V5 protocol [RFC-1928], which provides a framework for doing user-level, authenticated firewall traversal of unicast TCP and UDP traffic. However, because the current UDP support in SOCKS V5 has scalability problems as well as other deficiencies -- and these need to be addressed before multicast support can be achieved -- the extensions are defined in two parts: Base-level UDP extensions, and Multicast UDP extensions.

Using the SOCKS framework for managing multicast flows in/out of an organization, offers numerous security advantages over what is possible with a conventional firewall approach. These are spelled out in the draft.

Table of Contents

What s Changed

This revision contains the following changes from the version 00, dated July 23, 1997.

@ The title was modified to include "to Facilitate Multicast Firewall Traversal."

@ The document name was changed to draft-ietf-aft-mcast-fw-traversal-01.txt to better reflect the purpose of the document.

@ The abstract was shortened.

@ The Introduction section was added to provide background on the security issues with multicast and firewalls, and how SOCKS can solve them.

@ Clarification was added indicating that addressing information must be included in UDP BIND messages for an existing association.

@ Clarification was added on how non-proxy-based SOCKS servers respond to UDP BIND commands.

@ The section on Multicast Assumptions and Requirements (Section 5.1) was supplemented with a discussion about non-proxy-based multicast-capable SOCKS servers.

1. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC-2119].

2. Introduction

The main impetus for writing this draft is to provide a solution to getting multicast safely through firewalls. This section presents the case why an augmented SOCKS protocol provides such a solution.

Following are some of the security concerns that come to mind when considering allowing multicast through a firewall:

1. Prevent users from inadvertently sending stuff out (i.e., limit who can send content out, and with what scope)

2. Limit who can bring multicast content into an organization

3. Limit what groups can be joined

4. Provide protection to internal users who have joined an external multicast group (consistent with the protection defined by the unicast security policy)

5. Provide a detailed log of all multicast session activity (who was listening/sending and for how long)

6. Limit how much bandwidth is consumed by flows (which could by user dependent) to prevent over-utilization of an organization s network.

Nearly all of these concerns require that the firewall know the identity of the user and what they are attempting to do. While the same argument could be made for unicast protocols, it is actually a more pervasive argument for multicast. The reasons are:

- Most multicast applications have no formal control protocol that a firewall can monitor in order to dynamically modify filtering rules for UDP flows

- The only real "control protocol" a firewall can always monitor is the IGMP messages that carry multicast routing protocol information (DVMRP, PIM, etc.). However, since these messages are router-to-router, they do not contain anything about the original initiator of the message. What s more, only the first IGMP join for a given group will even

traverse the firewall; subsequent joins for the same group will be assimilated into the routing tree formed inside the firewall.

The end result is that without some light-weight out-of-band control protocol that authenticates a multicast client to the

firewall and makes requests of the firewall, it is not possible to address most of the security concerns mentioned above.

Given that such a protocol is needed, ideally it should be implementable in such a way that it is transparent to, and independent of, the multicast application running on the client. Because SOCKS is implemented at the session layer, and provides the same kinds of capabilities for unicast applications, it is ideally situated to provide these services for multicast applications; indeed, even more so for multicast applications because of the inherent challenges with IP multicast!

Using a multicast SOCKS solution offers the following advantages over a conventional firewall that provides class D address filtering as well as IGMP monitoring and filtering:

@ The Multicast SOCKS Server (MSS) knows the identity of the user who has asked to a join a group (due to user authentication). In a conventional approach, simply monitoring the IGMP only reveals that somebody issued a join of a particular group.

@ The MSS not only knows the group address that a client wishes to join, it also knows the port. This enables the MSS to filter traffic specific to that group address and port, and block traffic to the group address that is for other ports (where unsuspecting unicast services may be listening on). This enables better protection of clients that have joined an external group. In a conventional approach where the firewall monitors IGMP, only the group address is specified; not the port.

@ A proxy-based SOCKS server can convert multicast to unicast, and in doing so, can provide enhanced security (authentication, integrity, and privacy).

The remainder of the document discusses how the aforementioned extensions are manifested into the SOCKS V5 protocol.

3. Use of Feature Discovery

Using the SOCKS feature discovery extension documented in [Van97], a SOCKS client MAY query a SOCKS server to see if it supports the UDP and/or Multicast extensions. In Feature Description List (FDL) syntax, the tag-subtag used by the SOCKS server to indicate this support is COMMAND-ENHANCED_UDP_MODE (04 04 in hexadecimal). Note, ENHANCED_UDP_MODE is a new command which is described in Section 4.1.1.

The values are:

Value	Description
X'01'	Base-level UDP extensions (covered in Section 4)

X'02' Multicast UDP extensions (covered in Section 5)

SOCKS servers that support *only* the Base-level UDP extensions do so only for unicast addresses.

SOCKS servers that support *only* the Multicast UDP extensions still support the base-level UDP extensions, but only in the context of providing the services described in Section 5.

SOCKS servers that support both extensions (i.e., for unicast and multicast) must explicitly indicate both values in the tag-subtag for enhanced UDP mode (e.g., 04 04 02 01 02 in hexadecimal).

4. Base-level UDP Extensions

The base-level UDP extensions augment the current SOCKS UDP support by increasing efficiency and scalability, as well as providing the foundation for the multicast extensions (discussed in Section 5).

The base-level UDP extensions provide the following new functionality:

* Ability to control multiple UDP sockets over one control connection

* Reduced overhead on UDP datagram handling (by reducing the size of the SOCKS UDP datagram header).

* Ability for SOCKS client to send and *receive* TCP-encapsulated UDP.

* Correct handling of fragmentation.

In order to provide these extensions, new commands are added, as well as changes to some of the existing SOCKS V5 packet structures and procedures. These changes are done in a way to maintain compatibility between a SOCKS client or server that does not support the extensions.

The following is a summary of the changes:

* The packet structure for both the UDP control channel packets and the reply packet changes after the SOCKS client and server enter "UDP enhanced-mode" (described below).

* All UDP-related commands are done on the UDP control channel and, among other new fields, include an Association IDentifier (AID) that uniquely identifies a particular UDP session (association) between an address/port on the SOCKS client and an address/port on a remote host.

* The following new UDP commands are added:

  * UDP Bind: Establishes a UDP association for sending and/or receiving (replaces the UDP Associate command as well as the INTERFACE DATA subcommand specified in [SOCKS5])
  * UDP Release: Releases a previously established association.

* Support for TCP-encapsulation of UDP data is supported by first establishing a UDP send and/or receive association for TCP-encapsulation, and then sending a new packet type (SOCKS-UDP-DATA-REQUEST) specifically for encapsulating UDP over TCP.

* The TCP control connection is not terminated when a command for a particular socket cannot be satisfied (since multiple UDP sessions can be controlled over the connection)

* The SOCKS UDP data header is reduced in size by using the association identifier instead of the actual destination address. Further, a Datagram Identifier field is added to this header to address problems with fragmentation.

The details of these extensions are spelled out in the following sections (in a manner consistent with that used in RFC 1928).

4.1. SOCKS Requests

The SOCKS Request message has not changed, however, a new command (ENHANCED_UDP_MODE) is added.

* CMD
  Existing commands:
  * CONNECT X'01' (used only for TCP)
  * BIND X'02' (used only for TCP)
  * UDP ASSOCIATE X'03' (not supported once Enhanced UDP Mode is achieved)

  New command:
  * ENHANCED_UDP_MODE X'04'

4.1.1. ENHANCED_UDP_MODE

4.1.1.1. Request

Providing the SOCKS server advertises support for the base-level UDP extensions (or even the multicast extensions), the client may send this command to the server, indicating its desire to use the base-level UDP extensions. The message is populated as follows:

VER: X'05'
CMD: ENHANCED_UDP_MODE (X'04')

All other fields are unused and SHOULD be 0s.

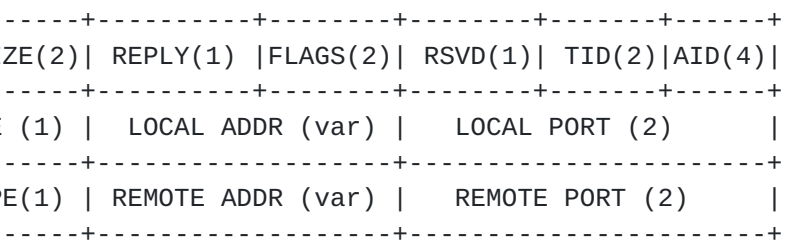
4.1.1.2. Reply

The reply message is the same as that used in SOCKS V5, however the version number will contain the version of the UDP extension instead of the SOCKS version number. The current version is X'01'.

Upon receiving a successful reply, the TCP control channel has achieved "Enhanced UDP Mode," and subsequently, can only be used for the UDP commands defined in Section 4.3.

4.2. SOCKS Replies

Once "Enhanced UDP Mode" has been established, the format of all subsequent SOCKS Reply messages is as follows:



The fields in the Reply packet are as follows:

* PKT TYPE: SOCKS-UDP-CONTROL-REPLY (X'02')
* SIZE: Total size (in octets) of this message, in network order.
* REPLY:

Existing return codes (SOCKS V5):

  * X'00' succeeded
  * X'01' general SOCKS server failure
  * X'02' connection not allowed by ruleset
  * X'03' Network unreachable
  * X'04' Host unreachable
  * X'05' Connection refused
  * X'06' TTL expired
  * X'07' Command not supported
  * X'08' Address type not supported

New return codes:

  * X'10' Bad Association ID
  * X'11' Multicast Receive not allowed by ruleset
  * X'12' Multicast Send not allowed by ruleset
  * X'13' Bad Association context
  * X'14' Bad multicast address
  * X'15' TTL value not allowed by ruleset

* FLAGS: Command-dependent flag
* RSVD: Reserved.
* TID: A 2-byte opaque Transaction Identifier that is set by the client in the request and is returned by the server in a

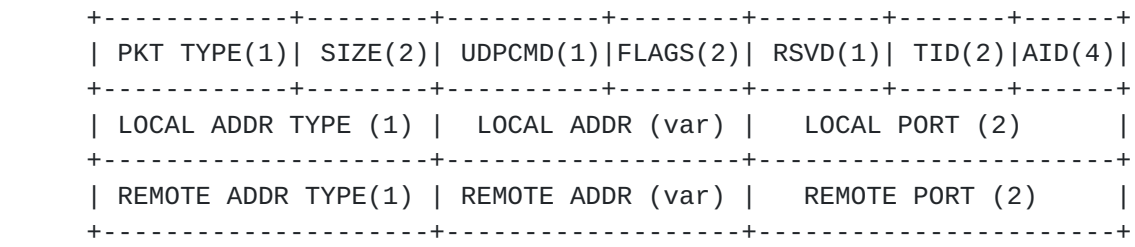
response to a particular request. The server does not use or interpret this value in any way; it merely returns it in the response to a particular request.

* AID: A 4-byte opaque Association Identifier that identifies a particular association.

* LOC ADDR TYPE/ADDR/PORT: Addressing information on the local side of the SOCKS server; that is, on the same side as the SOCKS client. The ADDR TYPEs and the format of the ADDR are the same as those in SOCKS V5.
* REM ADDR TYPE/ADDR/PORT: Addressing information on the remote side of the SOCKS server. The ADDR TYPEs and the format of the ADDR are the same as those in SOCKS V5.

4.3. UDP Control Channel

Once "Enhanced UDP Mode" has been established, the format of all subsequent SOCKS UDP Control messages is as follows:



The fields in the CONTROL CHANNEL packet are as follows:

* PKT TYPE: SOCKS-UDP-CONTROL-REQUEST (X'01')
* SIZE: Total size (in octets) of this message, in network order.
* UDP CMD: UDP command
  * UDP BIND: X'05' (new)
  * UDP RELEASE: X'06' (new)

  (Note: additional commands are defined in the Multicast extensions section of this document)
* FLAGS: A command-dependent bit-mask. These values are bit-wise ORed to form the actual flag.
  Valid flags are:
  * RECEIVE-FROM: X'0001'
  * SEND-TO: X 0002
  * UDP-UNICAST: X 0004
  * TCP-ENCAPSULATED: X 0008   Indicates that datagrams will be encapsulated in TCP using the procedures in Section 4.4.

  (Note: additional commands are defined in the Multicast extensions section of this document)
* RSVD: Reserved.
* TID: 2-byte opaque Transaction Identifier - client sets in request, server returns in response. Enables client to couple replies to requests.
* AID: 4-byte opaque Association Identifier
* LOC ADDR TYPE/ADDR/PORT: Local (source) address information on the SOCKS client. The ADDR TYPEs and the format of the ADDR are the same as those in SOCKS V5.

* REM ADDR TYPE/ADDR/PORT: Remote address information (for the ultimate destination). The ADDR TYPEs and the format of the ADDR are the same as those in SOCKS V5.

The use of these fields is command dependent and is further described below.

4.3.1. UDP BIND

4.3.1.1. Request

The SOCKS client sends the UDP BIND command each time a new binding (or association) is made between the SOCKS client and a UDP source or destination, enabling the SOCKS Server to establish a "UDP Relay" for the association.

The client MUST populate the following fields:

PKT TYPE: SOCKS-UDP-CONTROL-REQUEST (X'01')
SIZE: Total size (in octets) of this message, in network order.
UDP CMD: UDP BIND (X'05')
FLAGS: A bit-wise OR of the various flags defined in Section 4.3.

Examples:
RECEIVE-FROM | SEND-TO | UDP-UNICAST - client wants to send and receive using the unicast UDP method.

RECEIVE-FROM | UDP-UNICAST - client wants to only receive using the unicast UDP method.

RECEIVE-FROM | SEND-TO | TCP-ENCAPSULATED-UDP - client wants to send and receive using the TCP-encapsulated method. The client must do a UDP BIND with the SEND and TCP-ENCAPSULATED-UDP flags set, and get a successful response before following the procedures in Section 4.4.

Other combinations are also possible, however, UDP-UNICAST are TCP-ENCAPSULATED-UDP are mutually exclusive.

RSVD: Unused (X 00).
TID: Contains a unique value (chosen by the client) that enables the client to identify the reply to this particular request.
AID: Contains an existing association ID if a send or receive association already exists on this same socket to the destination address or X'00000000'.
LOCAL ADDR TYPE/ADDR/PORT: Contains the address and port that the client will use to send the UDP datagrams on for this association.
REMOTE ADDR TYPE/ADDR/PORT: Contains the remote (destination) address and port that the client wishes to send to or receive from.

It is permissible for a SOCKS client to establish a send or receive association to the same remote address in one UDP BIND operation, or by sending a another UDP BIND. Subsequent UDP BINDs SHOULD include the existing AID rather than establishing an entirely new AID for

the reverse direction, though the addressing information MUST still be included.

SOCKS client implementations are free to choose how many and which UDP sockets they control over a single TCP control connection. For example, a SOCKS client may control all UDP sockets for the entire client over a single control connection. Alternatively, it could group only those UDP sockets being used by a given process over a single control connection. It could even degenerate into the SOCKS V5 model of creating a control channel for each UDP socket, though this practice is not recommended.

4.3.1.2. Reply Processing

The reply to a UDP BIND will contain the following information:

PKT TYPE: SOCKS-UDP-CONTROL-REPLY (X'02')
SIZE: Total size (in octets) of this message, in network order.
REPLY: succeeded (X'00') or specific failure. Note: if the SOCKS server cannot entirely satisfy the request, it MUST fail the request, and send back an appropriate failure code.
FLAGS: Contains a bit-wise OR of the parts of the request that the SOCKS server could satisfy (even if the entire request is failed due to parts that could not be satisfied).
RSVD: Unused (X 00).
TID: Will contain the same value as that used in the corresponding request.

If the REPLY is a failure, then the remaining fields are not populated and SHOULD be 0s. If REPLY is succeeded, then the remaining fields MUST be populated as follows:

AID: The AID will contain a unique identifier (the SOCKS server determines the extent of the uniqueness) that the client must use in subsequent UDP commands referring to this association. The AID must also be used in the SOCKS header on the UDP datagrams (discussed below).
LOCAL ADDR TYPE/ADDR/PORT: The Address on the SOCKS server that the SOCKS client must send to for UDP datagrams to be forwarded. This is also the address that the SOCKS server will send from when forwarding datagrams to the SOCKS client. Note: non-proxy-based SOCKS servers would fill this field in with the destination address specified by the client in the REMOTE ADDR fields in the UDP BIND.
REMOTE ADDR TYPE/ADDR/PORT: The address information, specific to the interface, that the SOCKS server will send (or receive) on, on behalf of the SOCKS client for this particular association. On a multi-homed SOCKS server, this address

is usually not reachable by the SOCKS client, but is reachable by the host on the remote side of the SOCKS server. Note: non-proxy-based SOCKS servers would fill this field in with the client s local address specified by the client in the LOCAL ADDR fields in the UDP BIND.

If a UDP Control Channel is closed, the SOCKS Server MUST release all associations established on that control channel. Thus, implementations should not necessarily close the control connection if a UDP BIND fails (as was common practice in SOCKS V5 when a UDP ASSOCIATE failed), unless there are no other UDP associations active or the implementation desires this behavior.

4.3.2. UDP RELEASE

4.3.2.1. Request

The SOCKS client sends the UDP RELEASE command when a previously established association is to be terminated. The UDP RELEASE must contain the AID for the particular UDP association to be terminated. It MUST also contain the appropriate flag(s) for which the association is to be released.

PKT TYPE: SOCKS-UDP-CONTROL-REQUEST (X'01')
SIZE: Total size (in octets) of this message, in network order.
UDP CMD: UDP RELEASE (X'06')
FLAGS: Contains a bit-wise OR of one or more of the following:
RECEIVE-FROM (X 0001)
SEND-TO (X 0002)
Note: it is permissible for an implementation to send X 03 (an OR of RECEIVE-FROM and SEND-TO) even if it had previously only established a send or receive association, but not both.
RSVD: Unused (X 00).
TID: contains a unique value (chosen by the client) that enables the client to identify the reply to this particular request.
AID: Association Identifier that identifies the UDP association to be released.

4.3.2.2. Reply Processing

The SOCKS Server will send a REPLY to a UDP RELEASE with the following fields populated:

PKT TYPE: SOCKS-UDP-CONTROL-REPLY (X'02')
SIZE: Total size (in octets) of this message, in network order.
REPLY: succeeded (X'00') or specific failure
FLAGS: Same as those specified in the request.
RSVD: Unused (X 00).
TID: Will contain the same value as that used in the corresponding request.
AID: Association Identifier contained in the UDP RELEASE

4.4. Procedure for TCP-Encapsulation

Once a client has established an association ID for TCP-encapsulated data transmission, then it must send the encapsulated data over the UDP control channel using the following packet structure:

```
+-------------+----------+---------+-----------------+
|PKT TYPE (1) | SIZE (2) | AID (4) | DATA (variable) |
+-------------+----------+---------+-----------------+
```

PKT TYPE: SOCKS-UDP-DATA-REQUEST (X'03')
SIZE: Total size (in octets) of this message, in network order.
AID: Association ID
DATA: Application data. Note: The size of the application data is SIZE   7.

There is no reply to a SOCKS-UDP-DATA-REQUEST. The SOCKS server either silently relays the request or discards the packet if it cannot or will not relay it.

The programming interface MUST report an available buffer space for UDP datagrams that is no more than 65,529 decimal (2^16   7) since the two-byte size field in the header includes the 7-byte header itself.

It is permissible for a client to establish and control multiple UDP associations (TCP-encapsulated or otherwise) over a single control channel. Thus, SOCKS clients and servers that support TCP-encapsulation MUST support receiving SOCKS-UDP-DATA-REQUEST and SOCKS-UDP-CONTROL-REQUEST packets on the same control connection.

4.5. Procedure for UDP-based Clients

In a manner similar to SOCKS V5, a UDP-based client operating in enhanced UDP mode MUST send its datagrams to the UDP relay server at the UDP port indicated by LOCAL ADDR and PORT in the reply to the UDP BIND request. If the selected authentication method provides encapsulation for the purposes of authenticity, integrity, and/or confidentiality, the datagram MUST be encapsulated using the appropriate encapsulation.

Each UDP datagram carries a UDP request header with it (note this request header differs significantly from that used in SOCKS V5).

```
+---------+----------+----------+---------+-----------------+
| RSV (1) | FRAG (1) | DGID (2) | AID (4) | DATA (variable) |
+---------+----------+----------+---------+-----------------+
```

The fields in the UDP Data header are:

RSV: Reserved X'00'
FRAG: Current fragment number
   00 = standalone (not fragmented)
   1-127 = fragment # specific to this DGID
   MSB set (128-255) = End of fragment

DGID: Datagram Identifier opaque Identifier that uniquely
identifies the fragments of a specific datagram
AID: Association Identifier
DATA: user data

As in SOCKS V5, when a UDP relay server decides to relay a UDP datagram, it does so silently, without any notification to the requesting client. Similarly, it will drop datagrams it cannot or will not relay.

Furthermore, when a UDP relay server receives a reply datagram from a remote host, it MUST encapsulate that datagram using the above UDP request header and any authentication-method-dependent encapsulation.

The SOCKS server must ensure that only datagrams originated from the expected remote host (i.e., the one designated in the REMOTE ADDR information in the UDP BIND) are relayed to the SOCKS client. Any datagrams arriving from a source IP address other than the one recorded for the particular association MUST be dropped.

The programming interface on the SOCKS client MUST report an available buffer space for UDP datagrams that is 8-Bytes smaller that the actual space provided by the operating system. This allows for the SOCKS UDP data header.

Fragmentation is optional and is handled the same as that documented in [RFC-1928], with the following exception:

* When fragmentation is supported and a datagram is fragmented, the DGID field MUST be populated with a unique value that remains constant for all fragments of the particular datagram on the particular association. Note: DGID MUST be unique to the degree that no other SOCKS data header with the same AID uses the same value until the last fragment of a given datagram is transmitted. Implementations are encouraged to preserve the uniqueness longer by using a monotonically increasing counter for each datagram that is fragmented. The receiving end of the fragments can use the DGID to ensure that all fragments of the *same* datagram are received properly before passing the datagram up to the application layer.

5. Multicast Extensions

If during feature discovery (described in Section 3), a SOCKS server advertises support for the Multicast UDP extensions, then the SOCKS client and server must follow the guidelines set forth in this section as well as support the base-level UDP extensions specified in Section 4 (for multicast usage). The procedure for the SOCKS client to indicate to the SOCKS server that it wants to use the multicast extensions is the same as that described in Section 4.1.1 and is summarized below.

1. SOCKS client does feature discovery to the SOCKS server and learns whether or not Multicast UDP extensions are supported
2. If so, the SOCKS client may send ENHANCED_UDP_MODE command

3. After a successful reply, the SOCKS client may use the multicast extensions.

5.1. Assumptions and Requirements

The following list provides some assumptions and requirements about the deployment and usage of SOCKS in a multicast environment. The actual protocol details in the next section are in large part derived from these assumptions.

* A Multicast-capable SOCKS Server (MSS) is a SOCKS server that supports the multicast extensions defined below. It MAY also support all of the SOCKS V5 capabilities defined in [RFC-1928], including TCP services. However, a MSS could be dedicated for multicast usage only.
* A Multicast-capable SOCKS Client (MSC) configuration will specify the MSS address, as well as the range of class D IP addresses and TTL for which it must use the MSS.
* The MSS is not required to have the capabilities of a multicast router (mrouter) built into it. A MSS could coexist with a mrouter, collectively forming a barrier between an internal and external multicast network (e.g., Mbone) through which all multicast traffic must pass in order to enter or leave the internal network. See Figure 1 below for an example of such an arrangement.

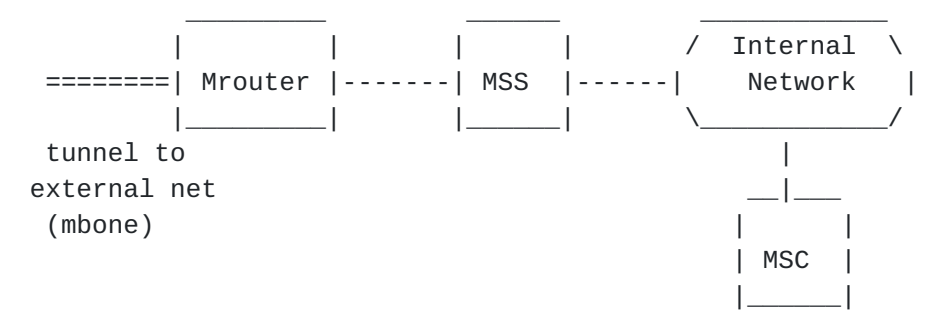


Figure 1: Example Usage of a SOCKS Multicast Server

Other possibilities exist where perhaps the MSC exists on the "external" network. However, for the purpose of the discussion in Section 5, the MSC is assumed to be on the "internal" network, and internal and external are relative to Figure 1.

* The MSS SHOULD support two different modes of operation for multicast:
  * Multicast-to-Unicast Mode (MU-mode) - the MSS receives packets from an external multicast group and unicasts them

to internal MSCs who have established a receive association to that group. Similarly, any packets received from an internal MSC are multicast externally, as well as unicast

to the other MSCs that have a receive association to the particular multicast group address.

* Multicast-to-Multicast Mode (MM-mode) - the MSS receives packets from an external multicast group and multicasts them to internal MSCs on the same multicast group address. Similarly, packets received by the MSS from an internal MSC on a particular multicast address are multicast externally on the same address.

  MM-mode provides less control than MU-mode in that non-SMCs could listen to a group address once it's allowed in. However, MM-mode significantly increases efficiency and scalability since it needs only to multicast a given datagram once rather than unicasting a copy to each SMC with an association to the multicast group.

  When operating in MM-mode, no UDP data headers will be used on datagrams since authentication-method-dependent encapsulation is not currently possible for a group; hence, handling fragmentation is not an issue. Note: in order to support encapsulation, an authentication-method-dependent procedure is needed to securely distribute shared security parameters to each of the members of a particular group. This mechanism is currently not defined, and is for further study.

How the mode is configured on the MSS, and whether it applies globally, on a user basis, or a multicast group basis, is implementation dependent.

* In both modes, the MSS MUST NOT forward packets from internal hosts that have not authenticated themselves to the MSS and received permission to receive from or send to a multicast group.

* MSSs SHOULD support filtering/blocking of externally originated multicast datagrams based on a configurable list of source addresses. This allows an MSS to be aware of other MSSs that share the same border between the two networks (e.g., corporate network and the Internet), enabling it to filter potential duplicate packets that originated from one of the other MSSs.

* The implementation of a MSS may or may not be based on a proxy architecture. Typically, SOCKS servers have been implemented as a proxy; in such case, the UDP streams are received by a process on the MSS and then retransmitted (in either MU-MODE or MM-MODE fashion). Alternatively, an MSS

could be implemented in such a way that the UDP datagrams are not proxied at the application layer, but routed at the network layer. In this case, SOCKS acts as a control protocol to a mrouter-like device in order to dynamically modify the multicast UDP filter rules.

5.2. UDP Control Commands and Flags

In order to support multicast, the following functionality is needed between the MSC and MSS:

* Start/stop receiving datagrams from a multicast group
* Start/stop allowing the sending of datagrams to a multicast group
* Change the multicast scope (TTL) that the MSS uses when sending (relaying) the datagrams to a multicast group

To provide these capabilities, the following additions are made to the usage of the base-level UDP extensions:

* Flags are added to the UDP BIND command and reply (UDP-MULTICAST, MM-MODE, MU-MODE), which allow the MSC to establish a multicast-receive or multicast-send association (or both) using a single association ID (AID), as well as to request and learn the multicast mode from the MSS.
* A new UDP command (SET-MCAST-TTL) has been added to control the changing of the multicast scope.

The values for the new command and the command flags are defined as follows:

UDP Command:
* UDP BIND:
  * command flags
    * UDP-MULTICAST: X 0010 (16 Decimal) - acts as a modifier to the other flags defined in defined in Section 4.3.
    * Multicast-to-Unicast Mode (MU-MODE): X'0020' (32 Decimal)
    * Multicast-to-multicast Mode (MM-MODE): X'0040' (64 Decimal)
* SET TTL: X'07'

The details of the new multicast commands/flags follow below.

5.2.1. Receiving From a Multicast Group

5.2.1.1. Request

A UDP BIND command containing both the RECEIVE-FROM and UDP-MULTICAST FLAGs set, indicates the MSC's desire to receive packets from a particular multicast group. If the MSC has already established an association to this multicast group on the same socket (via the SEND-TO/UDP-MULTICAST flags), then the MSC SHOULD include the existing AID. Note: while it is possible for the SMC to

create an entirely new AID, it is potentially more wasteful on the MSS, and is not recommended.

As in the base-level UDP extensions, a client implementation MAY establish both a send and receive association in one or two UDP BIND operations. If the MSC groups the send and receive request together and the MSS can not satisfy the entire request, the reply to the UDP BIND will fail with an appropriate failure and the FLAGS SHOULD be set with what the MSC is permitted to do (relative to what was requested).

The MSC MUST populate the following fields:

PKT TYPE: SOCKS-UDP-CONTROL-REQUEST (X'01')
SIZE: Total size (in octets) of this message, in network order.
UDP CMD: UDP BIND (X'05')
FLAGS: Bitwise OR of RECEIVE-FROM and UDP-MULTICAST. May optionally include MM-MODE and/or MU-MODE to indicate the MSC s preference of the mode. If both flags are included or excluded, the client has no preference. The MSS is not obligated to honor the MSC s preference, but SHOULD if possible (i.e., it doesn t violate security policy). Additionally, the use of TCP-ENCAPSULATED-UDP is allowed with RECEIVE-FROM and UDP-MULTICAST, and is mutually exclusive with the MM-MODE flag.
RSVD: Unused (X 00).
TID: Unique Transaction ID generated by the MSC.
AID: Contains an existing association ID if a send-association already exists on this same socket to the same multicast group or X'00000000'.
LOCAL ADDR TYPE/ADDR/PORT: contains the unicast address and port (on the MSC) that the MSS will use if MU-mode is employed.
REMOTE ADDR TYPE/ADDR/PORT: contains the multicast group address that the MSC desires to join.

5.2.1.2. Reply Processing

The successful response to this command will contain the following fields populated by the MSS:

PKT TYPE: SOCKS-UDP-CONTROL-REPLY (X'02')
SIZE: Total size (in octets) of this message, in network order.
REPLY: one of:
* X 00 Succeeded
* X'10' Bad Association ID
* X'11' Multicast Receive not allowed by ruleset
* X'12' Multicast Send not allowed by ruleset
* X'14' Bad multicast address

Note: if the SOCKS server cannot entirely satisfy the request, it MUST fail the request, and send back an appropriate failure code.

FLAGS: Contains a bit-wise OR of the parts of the request that

the SOCKS server could satisfy (even if the entire request is failed due to parts that could not be satisfied) including selection of the multicast mode.

RSVD: Unused (X 00).

TID: Contains the MSC-generated transaction ID in the corresponding request.
AID: Unique Identifier for this association.
LOCAL ADDR TYPE/ADDR/PORT: If MU-Mode is used, these fields contain the unicast address and port on the MSS that the MSC will receive datagrams from.
REMOTE ADDR TYPE/ADDR/PORT: Contains the multicast group address that the MSC requested to join.

Upon receiving a successful reply, a multicast receive association has been established between the MSC and MSS, and the MSC may begin receiving any traffic generated to the group address. However, before the MSC can send to the group, it must establish a "send association" by following the procedure in Section 5.2.2.

5.2.2. Sending to a Multicast Group

5.2.2.1. Request

A UDP BIND command containing both the SEND-TO and UDP-MULTICAST FLAGs set indicates the MSC's desire to send to a particular multicast group. This command must be issued by the MSC regardless if the MSC has already established a receive-association (by using UDP BIND with the RECEIVE-FROM/UDP-MULTICAST flags). If the MSC has already established a receive association to this multicast group on the same socket, then the MSC SHOULD include the existing AID. Note: while it is possible for the SMC to create an entirely new AID, it is potentially more wasteful on the MSS, and is not recommended.

The MSC MUST populate the following fields:

PKT TYPE: SOCKS-UDP-CONTROL-REQUEST (X'01')
SIZE: Total size (in octets) of this message, in network order.
UDP CMD: UDP BIND (X'05')
FLAGS: Bitwise OR of SEND-TO and UDP-MULTICAST. May optionally include MM-MODE and/or MU-MODE to indicate the MSC s preference of the mode. If both flags are included or excluded, the client has no preference. The MSS is not obligated to honor the MSC s preference, but SHOULD if possible (i.e., it doesn t violate security policy). Additionally, the use of TCP-ENCAPSULATED-UDP is allowed with SEND-TO and UDP-MULTICAST, and is mutually exclusive with the MM-MODE flag.
RSVD: Contains the TTL value (1-255) the MSS should use during transmission to the multicast group.
TID: Unique Transaction ID generated by the MSC.
AID: Contains an existing association ID if a receive-association already exists on this same socket to the same

multicast group or X'00000000'.

LOCAL ADDR TYPE/ADDR/PORT: Contains the unicast address and port (on the MSC) that the MSS will receive from if MU-mode is employed.

REMOTE ADDR TYPE/ADDR/PORT: Contains the multicast group address that the MSC desires to send to.

Upon receiving a successful reply, a multicast send-association has been established between the MSC and MSS.

5.2.2.2. Reply Processing

The response to this command will contain the following fields populated:

PKT TYPE: SOCKS-UDP-CONTROL-REPLY (X'02')
SIZE: Total size (in octets) of this message, in network order.
REPLY: one of:
* X'10' Bad Association ID
* X'11' Multicast Receive not allowed by ruleset
* X'12' Multicast Send not allowed by ruleset
* X'14' Bad multicast address

FLAGS: if REPLY is succeeded, then contains a bit mask
containing the mode selected by the MSS
Multicast-to-Unicast Mode (MU-MODE): X'0020'
or
Multicast-to-multicast Mode (MM-MODE): X'0040'

RSVD: Unused (X 00).
TID: Contains the MSC-generated transaction ID in the corresponding request.
AID: Unique Identifier for this association.
LOCAL ADDR TYPE/ADDR/PORT: If REPLY is succeeded and FLAGS is MU-Mode, these fields contain the unicast address and port on the MSS that the MSC MUST send datagrams to for them to be relayed to the multicast group.
REMOTE ADDR TYPE/ADDR/PORT: If REPLY is succeeded and FLAGS is MM-Mode, these fields contain the multicast address and port that the MSC MUST send datagrams to for them to be relayed to the multicast group specified in the request. Typically this would be the same as that specified in the request.

5.2.3. Releasing a Multicast Association

Releasing a multicast send or receive association (or both) is identical to the procedure described for the base-level UDP extensions in Section 4.3.2.

5.2.4. Setting the TTL

5.2.4.1. Request

The SOCKS MSC uses the SET TTL command to instruct the MSS of the TTL (Time-to-Live) value that it SHOULD use when sending datagrams

to a particular multicast group. This command is only valid for an established multicast send association.

PKT TYPE: SOCKS-UDP-CONTROL-REQUEST (X'01')

SIZE: Total size (in octets) of this message, in network order.
UDP CMD: SET TTL (X'07')
FLAGS: unused (X'0000')
RSVD: Contains the TTL value (1-255) the MSS should use during transmission to the multicast group.
TID: Unique Transaction ID generated by the MSC.
AID: Contains an existing association ID.

5.2.4.2. Reply Processing

The reply to a SET TTL will be populated as follows:

PKT TYPE: SOCKS-UDP-CONTROL-REPLY (X'02')
SIZE: Total size (in octets) of this message, in network order.
REPLY: one of:
succeeded (X'00')
Bad Association ID (X'10')
Bad Association context (X'13') - No send association is established
TTL value not allowed by ruleset (X'15') TTL value is too high
TID: Contains the MSC-generated transaction ID in the corresponding request.
FLAGS: Unused (X'0000')
AID: Unique Identifier for this association.

6. Security Considerations

This document describes extensions to [RFC-1928] which itself is a security protocol for the session-layer traversal of IP network firewalls. As in [RFC-1928], the security of firewall traversal is highly dependent on the authentication and encapsulation methods provided by a particular implementation, and selected during the negotiation between the SOCKS client and SOCKS server.

Multicast-specific considerations include:

* MM-Mode vs. MU-Mode: MU-Mode is more secure than MM-Mode for two reasons:
  1.) Packets are specifically unicast from the SOCKS server to the SOCKS client, and vice-versa, making them harder (than MM-Mode) to intercept for eavesdroppers;
  2.) Authentication-method-dependent encapsulation is supported in MU-Mode, but not MM-Mode. Thus, in MU-Mode, it is possible for the SOCKS server to use packet-level authentication in determining whether to forward a packet, whereas in MM-Mode, it must resort to inspecting the source IP address.
  For many environments, the added performance and scalability

offered by MM-Mode may outweigh the additional security offered by MU-Mode. Administrators should carefully understand the tradeoffs.

* SOCKS client implementations SHOULD optionally support getting user confirmation upon attempting to send multicast

traffic through a SOCKS server. This helps prevent the unintentional multicasting of data beyond an organizational boundary.

* SOCKS Server implementations MAY perform content filtering in cases where, based on addressing or other means, the server knows what the content should be. For example, since the class D addresses in the range of 224.2.*.* are reserved for conferencing, a SOCKS server could ensure the UDP traffic is encapsulated in RTP/RTCP headers.

7. Acknowledgments

This document benefited from the thoughtful insights and comments from Marc Vanheyningen, Wei Lu, Jamie Jason, and Kira Attwood.

8. References

[RFC-2119] Bradner, S, "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, Harvard University, March 1997.

[RFC-1928] Leech, M., et. al., "SOCKS Protocol Version 5", RFC 1928, March 1996.

[SOCKS5] Leech, M., et. al., "SOCKS Protocol Version 5", March 1997, work in progress.

[Van97] VanHeyningen, M., "Feature Discovery: A Generic Mechanism for SOCKS Version 5", July 1997, work in progress.

9. Disclaimer

The views and specification herein are those of the author and are not necessarily those of his employer. The author and his employer specifically disclaim responsibility for any problems arising from correct or incorrect implementation or use of this specification.

10. Authors' Address

Dave Chouinard
Intel Corporation
MS JF3-206
2111 NE 25th Ave.
Hillsboro, OR, USA 97124
+1(503)264-7481
dave_chouinard@mail.intel.com