Network Working Group
Internet-Draft
Expires: March 11, 2007

M. Shore
D. McGrew
Cisco Systems
September 7, 2006

An Authorized IP Firewall Control Application
draft-shore-afwc-00.txt

Status of this Memo

Copyright Notice

Abstract

The Authorized Firewall Control Protocol provides an interface that allows network entities to request firewall and NAT services and resources. It is an instance of a protocol that provides authorizations and other security servcies, and interworks with other such instances. AFWC uses its authorization facilities to provide network administrators more control over network border admissions decisions than is provided by other firewall pinholing protocols.

Table of Contents

1. Introduction

This specification defines a protocol for establishing and managing firewall pinholes, and a formal model for evaluating pinhole requests against a pre-established set of authorizations. The design relies on a lower layer in the system to provide cryptographic protections, and to provide the authorizations of the other endpoint.

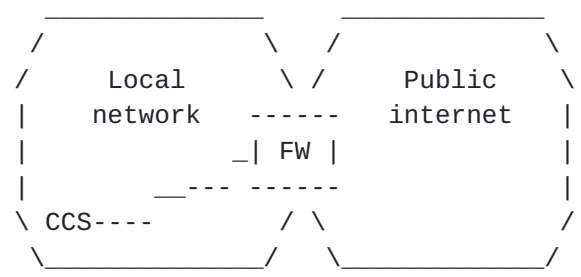
In this document we refer to a "firewall control application." The application actually supports requests to both firewalls and NATs, and while we do not consider NAT to be a type of firewall or to be a security device in general, we recognize that there are substantial semantic, topological, and protocol similarities between asking for a firewall pinhole and asking for a NAT table mapping. For the sake of convenience we will refer to a firewall control application throughout this document, acknowledging that in reality the application includes NAT, a non-firewall function.

2. Network scenarios

Participants in an AFWC dialogue include an initiator, a responder, and an authenticationserver. In the firewall control application, the "initiator" is the entity requesting a firewall pinhole or service, or NAT table mapping. This might be, for example, an IP telephony call control server, or a network gaming endpoint. The "responder" is the firewall or NAT, and the authentication server is the server providing authentication and authorization services to the initiator and responder.

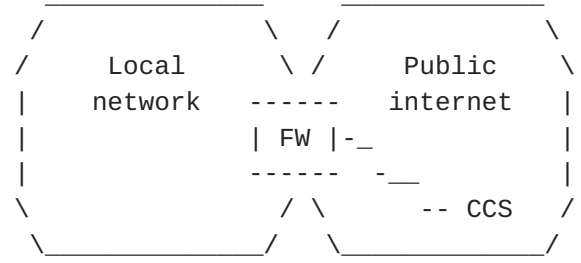
In this document we will use "initiator" and "responder" to describe protocol participants.

Firewall control is typically seen as between elements in the same network, allowing implicit authorization based on topological considerations such as sharing of address space. For example,



In this figure, a VoIP call control server ("CCS") establishes a request connection to a firewall ("FW").

However, it may be the case in some instances or for some applications that an external entity, outside of the local network or address space, may wish to request pinholes or other resources from the firewall.



In this scenario the trust relationships will need to be made explicit, and there may be substantial changes to the security considerations.

3. Transport

AFWC relies on a cryptographic layer for transport and to provide authorizations of the other endpoint. The following cryptographic services MUST be provided by the crypto layer:

- o entity authentication of the peer
- o confidentiality of all messages sent to and from the peer
- o message authentication on all messages sent to and from the peer
- o protection from replayed messages
- o protection from reflected messages.

In addition, the lower layer MUST establish the authorizations of the peer in a secure and reliable manner and pass these authorizations to this protocol.

The messages are framed using the following format. The REQUEST (Figure 3) and RESPONSE (Figure 4) elements convey arbitrary data in their Body fields. This data is meant to convey a request or a response. The Access Type Identifier (ATI) field (Section 5) indicates the type of access that is being requested. The Body field is interpreted according to the value of the ATI field. The Request Identifier field is used to match replies to requests. That field can be set to any value by the sender of a REQUEST; these values SHOULD be distinct. The receiver of a REQUEST element MUST set that field of the corresponding RESPONSE element to the same value.

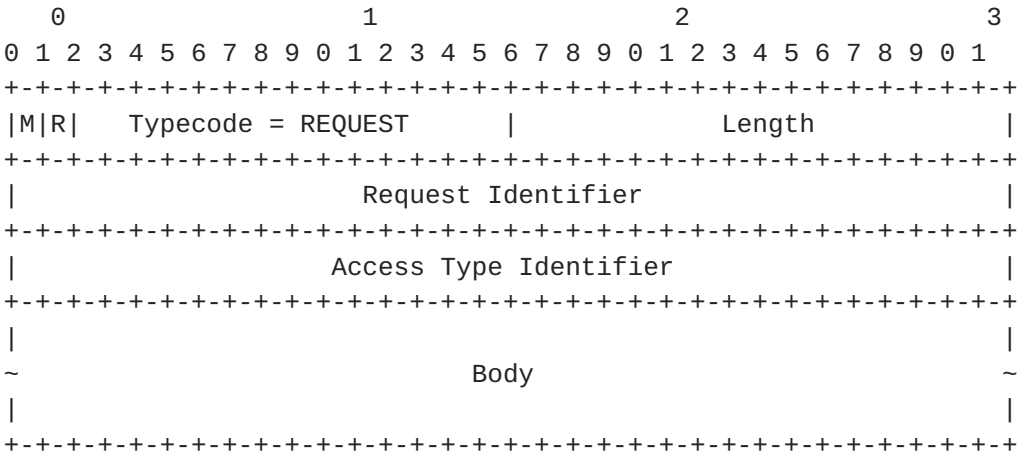


Figure 3

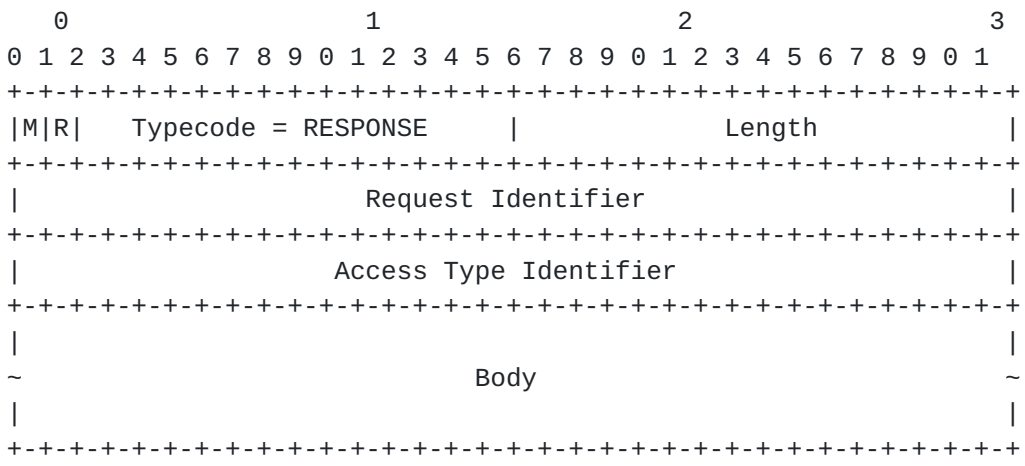

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
|M|R| Typecode = RESPONSE | Length |
Request Identifier
Access Type Identifier
Body

Figure 4

4. Pinholes

4.1. Firewall pinholes

In this specification, a pinhole is a traffic flow that the firewall permits, which can be very narrowly scoped. For example, a pinhole can be created that allows VoIP traffic to flow between a phone behind the firewall and a phone outside the firewall, but which does not allow traffic from any other hosts outside the firewall to reach the phone behind the firewall. The definition of a traffic flow used in this specification appears below.

Each pinhole is associated with a PINHOLE_ID value, which can be used to uniquely identify the pinhole, and a timer value that indicates when the pinhole is to expire.

A pinhole corresponds to a "permit" statement, in that it only allows traffic through the firewall, and cannot cause traffic to be blocked. There is no equivalent "deny" statement in this specification. This design decision makes the application simpler and should make implementations easier to manage. (n.b. this should not in any way be construed as suggesting that an application cannot request the closure of a pinhole it created -- see below)

An initiator MAY request a pinhole for which the traffic flow is already allowed through the responder (firewall), either in part or in whole. In this case, if the responder rejects the request, it MUST NOT be construed to indicate that the flow has been blocked.

4.2. NAT Table mappings

A NAT table mapping is the data structure in a NAT providing the mapping between an external {address, port, protocol} tuple and an internal, private {address, port, protocol} tuple. In some types of NAT there may be additional data associated with the mapping. For example, in a symmetric NAT each mapping is also associated with one external peer. Those additional data are opaque to the endpoint and are internal to the NAT.

A NAT table mapping is represented by a PINHOLE_ID value, as well.

5. Access Type

The IP Firewall Control application is identified by an Access Type Code of ACCESS_TYPE_FW_CTL. The cryptographic and authorization layers use this code in order to deliver the IP Firewall Control messages to the appropriate application on each participating device. The Access Type Code identifies an authorization application in the same way that a well-known TCP port identifies a service on a host.

6. Message Exchanges

This section defines the IP Firewall application protocol. We use the notation that T* denotes zero or more instances of the term T, T+ denotes one or more instances of the term T, and T? denotes zero or one instances of the term T. In this section, I is the initiator, a device that is making an access control request to the firewall, and R is the responder, which is the firewall itself. 'I' refers to the initiator (the entity requesting pinholes) and 'R' refers to the responder, which in this case is the firewall.

The crypto layer and transport layer operate below the application layer, and provide the essential security services and reliable data transport. The application layer contains the "access control dialog". It contains four messages: Start, Offer, Request, and Response. These messages are described below along with the data formats and the semantics used in the IP Firewall Control application.

6.1. Open Pinhole Exchange

An initiator begins an Open Pinhole exchange in order to cause a responder to allow a particular traffic flow. The flow is described by one or more IPV4_SELECTOR or IPV6_SELECTOR TLVs, and is associated with a particular PINHOLE_ID.

Message	Flow	Format
Start	I -> R	<empty>
Offer	R -> I	(PINHOLE_ID (IPV4_SELECTOR | IPV6_SELECTOR)*) | INFO
Request	I -> R	PINHOLE_ID (IPV4_SELECTOR | IPV6_SELECTOR)? | NAT_TUPLE? | INFO
Response	R -> I	PINHOLE_ID INFO? | NAT_INFO?

6.1.1. Start message

The Start message is sent by the initiator to the responder to initiate the authorization exchange. The message body is empty.

6.1.2. Offer Message

A responder constructs the Offer message as follows. First, it MUST

check the authorizations of the initiator to make sure that it is authorized to act as a controller (see the section on Authorizations). If it is not, then the Offer message MUST contain an INFO element with Status Code ERROR and Info Code ACCESS_NOT_ALLOWED, and the session must be terminated. Otherwise, the responder encodes into the Offer a PINHOLE_ID that is not currently in use, which will be associated with the pinhole created by the session, if it completes successfully. The responder MAY send one or more IPV4_SELECTOR or IPV6_SELECTOR TLV element(s) describing the traffic that it is willing to allow. The use of these elements provides basic capability discovery and topology discovery . The responder indicates to the authorization system the maximum time, expressed as a number of seconds, that it is willing to allow a pinhole to remain open. This value is used by the authorization system, and is also conveyed to the initiator.

A responder that also does NAT, or a stand-alone NAT, MUST include an INFO element that has a Status Code of CAPABILITIES with the NAT flag set.

The responder MAY include INFO elements that indicate other conditions, though the controller MAY ignore them.

An initiator processes an Offer message as follows. First, it MUST check the authorizations of the responder to make sure that it is authorized to act as a firewall or NAT (see the section on Authorizations). If it is not, then the Offer message MUST contain an INFO element with Status Code ERROR and Info Code ACCESS_NOT_ALLOWED, and the session must be terminated. If any IPV4_SELECTOR or IPV6_SELECTOR element(s) appear in the message, the initiator SHOULD use the information to guide the construction of the Request.

6.1.3. Request message

The initiator builds the firewall request by including the PINHOLE_ID element sent in the Offer and zero or more IPV4_SELECTOR or IPV6_SELECTOR elements that describe the traffic that the initiator is requesting to be allowed to traverse the firewall. The initiator indicates to the authorization system the minimum time that it would like for the pinhole to remain open; this value MUST be no greater than the duration indicated with the Offer message. The initiator MAY include INFO elements that indicate other conditions, though the responder MAY ignore them.

The initiator builds the NAT request by including the PINHOLE_ID element sent in the Offer and one or more NAT_TUPLE elements that describe the internal address for which the initiator is requesting a

mapping. The presence of a NAT_TUPLE element implies that a NAT table mapping is being requested, and, by implication, that a firewall pinhole request is being requested. There may be cases in which there are no SELECTOR elements but one or more NAT_TUPLE elements; in that case the request will be treated as being for both a NAT table mapping and a firewall pinhole for each NAT_TUPLE element. Otherwise the construction of the request message is the same as it is for a firewall pinhole request.

A responder processes a Request message as follows. First, it MUST check the authorizations of the initiator to make sure that it is authorized to open the pinhole that it has requested (see the section on Authorizations). If it is not, then the Offer message MUST contain an INFO element with Status Code ERROR and Info Code ACCESS_NOT_ALLOWED, and the session must be terminated. Otherwise, the responder constructs a Response message. This message serves as an acknowledgement.

NAT processing of the Request message is the same as firewall processing of the Request message.

6.1.4. Response message

The Response message contains the PINHOLE_ID that was included in the Offer and the Request. In the firewall case, if nothing goes wrong, then this message contains an INFO element with a Status Code of NOTIFY and an Info Code of OK. If there are any errors or warnings, then the INFO element must be set appropriately. If the duration requested by the initiator is greater than the maximum that the responder is willing to allow, then the responder SHOULD install the pinhole with the maximum duration to which it consents. In this case, the firewall SHOULD send an INFO element with Status Code WARNING and Info Code of DURATION_TOO_LONG. The responder MUST implement the pinhole before sending the Response. The number of seconds before the pinhole expires is provided to the authorization system, which forwards it to the controller.

If the request was for a NAT table mapping, the Response message MUST also contain a NAT_INFO TLV. The NAT_INFO TLV is used to communicate the external address back to the controller.

An initiator processes a Response as follows. It uses the PINHOLE_ID to associate the reply with the request that it made earlier. If an INFO element with Status Code NOTIFY and Info Code OK appears in the message, and no element with Status Code ERROR appears in the message, then the session has concluded successfully. Otherwise, the controller MUST NOT assume that the pinhole that it has requested has been implemented by the responder. If an INFO element containing

DURATION_TOO_LONG appears, then the initiator SHOULD be prepared to make another Open Pinhole request before the pinhole times out and is removed by the responder. The duration conveyed by the authorization system indicates the number of seconds that the responder has committed to keep the pinhole open.

6.2. Close Pinhole Exchange

An initiator initiates a Close Pinhole exchange to close a responder to a traffic flow that had been previously allowed via an Open Pinhole exchange. A Close Pinhole exchange causes the responder to reverse the firewall policy changes that were made in the previous exchange. The messages for this exchange are outlined in the following table.

Message	Flow	Format
Start	I -> R	<empty>
Offer	R -> I	(PINHOLE_ID (IPV4_SELECTOR | IPV6_SELECTOR)*) | INFO
Request	I -> R	CLOSE_PINHOLE PINHOLE_ID+
Response	R -> I	CLOSE_PINHOLE (PINHOLE_ID | INFO)+

The Start message is empty, as in the Open Pinhole exchange.

The responder constructs the Offer exactly as done for the Open Pinhole exchange (as it must, since at this point in the protocol it has no idea whether or not the initiator will be requesting to open or close a pinhole).

The initiator constructs the Request as follows. The PINHOLE_ID sent by the responder is ignored. The message MUST start with a CLOSE_PINHOLE TLV, which indicates that the initiator is requesting that one or more previously created pinholes are to be closed. The PINHOLE_ID element associated with the pinhole(s) that the initiator wishes to close are included in the message. At least one PINHOLE_ID element MUST appear in the message.

The responder constructs the Reply as follows. The message MUST start with a CLOSE_PINHOLE TLV, which acknowledges that the exchange will close previously opened pinholes. For each PINHOLE_ID that appears in the Request, the responder includes a PINHOLE_ID in the Response, followed by a INFO TLV element. The INFO element contains

a Status Code of NOTIFY and an Info Code of OK if the pinhole was closed successfully.

The Close Pinhole Exchange is the same for NATs as it is for firewalls.

7. Data formats

The IP Firewall application defines the following TLV types:

7.1. IPV4_SELECTOR

IPV4_SELECTOR encodes a traffic selector, which defines a particular IP Version Four packet flow. The firewall uses these values as the basis for packet matches. The Value field of an IPV4_SELECTOR TLV element consists of the ipv4_selector_t structure shown below, in network byte order:

```
typedef struct {
  uint32_t src_addr;
  uint32_t src_mask;
  uint32_t dst_addr;
  uint32_t dst_mask;
  uint16_t src_port_lo;
  uint16_t src_port_hi;
  uint16_t dst_port_lo;
  uint16_t dst_port_hi;
  uint16_t protocol;
  uint32_t spi;
  uint16_t reserved;
} ipv4_selector_t;
```

with the fields as follows:

src_addr: source address in the IP header

src_mask: a value to mask with the src_addr field

dst_addr: destination address in the IP header

dst_mask: a value to mask with the dst_addr field

src_port_lo: ports may be specified as a range of values; this is the low value for the source port in the IP header

src_port_hi: high value for the range of source ports in the IP header

dst_port_lo: low value for the range of destination ports in the IP header

dst_port_hi: high value for the range of destination ports in the IP header

protocol: the protocol number carried in the IP header

spi: IPSec SPI.

The value 0x00 is used in the protocol field to denote a match to any protocol.

7.2. IPV6_SELECTOR

IPV6_SELECTOR encodes a traffic selector, which defines a particular IP Version Six packet flow. The Value field of an IPV6_SELECTOR TLV element consists of the ipv6_selector_t structure shown below, in network byte order:

```
typedef struct {
  uint128_t src_addr;
  uint128_t src_mask;
  uint128_t dst_addr;
  uint128_t dst_mask;
  uint32_t flow_label;
  uint16_t src_port_lo;
  uint16_t src_port_hi;
  uint16_t dst_port_lo;
  uint16_t dst_port_hi;
  uint16_t protocol;
  uint16_t reserved;
} ipv6_selector_t;
```

with the fields as described above.

7.3. NAT_TUPLE

The NAT_TUPLE TLV contains the description of an {address, port, protocol} tuple for which a NAT table mapping is being requested. In other words, the NAT_TUPLE describes an internal address.

```
typedef struct {
       uint32_t addr;
       uint16_t port;
       uint16_t protocol;
       uint32_t hint_addr;
       uint16_t hint_port;
       uint16_t hint_protocol;
```

```
} nat_tuple_t;
```

The addr field represents an IPv4 address. Because this is for NAT, we assume that we will not need to support NAT functions for IPv6, although this may be revisited if necessary.

When the initiator is making the request on behalf of another party (for example, a call control server requesting a media pinhole for a VoIP endpoint) the hint_addr, hint_port, and hint_protocol fields SHOULD be used to assist a NAT device in resolving ambiguous requests. An example of ambiguity would be those cases when the NATted address spaces attached to two different interfaces on the same NAT use the same or overlapping addresses.

An example of its use would be for, say, a VoIP call control server to use the hint fields to send the {address, port, protocol} tuple of the endpoint from which it received signaling to the NAT. The NAT would search its mapping tables on all interfaces for a match. If a match is found the interface with which the matching mapping is associated would be the interface to which the request is applied.

If the hint fields are not used they MUST be null.

7.4. INFO

```
typedef struct {
    uint32_t status_code;
    uint32_t info_code;
} info_t;
```

The Status Code describes the status state machine of the sender of the INFO element. If any condition occurred that will prevent the successful completion of the exchange, then this field will have the value ERROR. This value indicates to the recipient that it MUST NOT expect the sender to participate in the exchange any further.

The Status Codes are:

Message	Meaning
OKAY	No error, no message
ERROR	An error was encountered
CAPABILITIES	This element contains a capabilities description

The Info Code provides detailed information, but does not convey any information about the base authorization exchange.

The Info Code values that can be used by the IP Firewall Control application are as follows:

Value	Meaning
OK	No problems occurred
ACCESS_NOT_ALLOWED	Access is denied due to lack of authorization
DURATION_TOO_LONG	Duration requested is too long
BAD_PARAMETER	A bad parameter appeared in a request
TRY_AGAIN	Request cannot be completed, but try again
RESOURCE_NOT_AVAILABLE	A resource needed for the request is not available
NAT	This device provides NAT functions

7.5. NAT_INFO

The NAT_INFO TLV carries the response to the NAT request (implicit in the inclusion of a NAT_TUPLE TLV in the Request message). It includes both the internal and external addresses.

```
typedef struct {
       uint32_t i_addr;
       uint32_t e_addr;
       uint32_t i_port;
       uint32_t e_port;
       uint16_t protocol;
} nat_info_t;
```

where the fields are as follows:

i_addr: Internal IPv4 address

e_addr: External IPv4 address

i_port: Internal port

e_port: External port

protocol Protocol (TCP, UDP, SCTP, etc.)

7.6. ICMP_MESSAGE

```
typedef struct  {
       u_char icmp_type;
       u_char icmp_code;
}  icmp_t;
```

ICMP_MESSAGE carries a description of a filter rule for ICMP messages, based on ICMP message types and codes.

The values are as follows:

```
   ICMP_ECHOREPLY                      0
   ICMP_UNREACH                        3
       ICMP_UNREACH_NET                0
       ICMP_UNREACH_HOST               1
       ICMP_UNREACH_PROTOCOL           2
       ICMP_UNREACH_PORT               3
       ICMP_UNREACH_NEEDFRAG           4
       ICMP_UNREACH_SRCFAIL            5
       ICMP_UNREACH_NET_UNKNOWN        6
       ICMP_UNREACH_HOST_UNKNOWN       7
       ICMP_UNREACH_ISOLATED           8
       ICMP_UNREACH_NET_PROHIB         9
       ICMP_UNREACH_HOST_PROHIB        10
       ICMP_UNREACH_TOSNET             11
       ICMP_UNREACH_TOSHOST            12
       ICMP_UNREACH_FILTER_PROHIB      13
       ICMP_UNREACH_HOST_PRECEDENCE    14
       ICMP_UNREACH_PRECEDENCE_CUTOFF  15
   ICMP_SOURCEQUENCH                   4
   ICMP_REDIRECT                       5
       ICMP_REDIRECT_NET               0
       ICMP_REDIRECT_HOST              1
       ICMP_REDIRECT_TOSNET            2
       ICMP_REDIRECT_TOSHOST           3
   ICMP_ECHO                           8
   ICMP_ROUTERADVERT                   9
   ICMP_ROUTERSOLICIT                  10
   ICMP_TIMXCEED                       11
       ICMP_TIMXCEED_INTRANS           0
       ICMP_TIMXCEED_REASS             1
   ICMP_PARAMPROB                      12
       ICMP_PARAMPROB_ERRATPTR         0
       ICMP_PARAMPROB_OPTABSENT        1
       ICMP_PARAMPROB_LENGTH           2
   ICMP_TSTAMP                         13
   ICMP_TSTAMPREPLY                    14
   ICMP_IREQ                           15
   ICMP_IREQREPLY                      16
   ICMP_MASKREQ                        17
   ICMP_MASKREPLY                      18
```

7.7. PINHOLE_ID

```
   typedef struct  {
          uint32_t id;
          uint32_t context;
```

```
}  pinhole_id_t;
```

The PINHOLE_ID is an identifier generated by the responder and associated with a particular pinhole. The id field of a PINHOLE_ID TLV is 32 bits long, and MAY be treated as an unsigned integer in network byte order (e.g. for display to the user). The method by which the firewall generates these identifiers is intentionally left unspecified, in order to provide maximum flexibility for implementations. Of course, each PINHOLE_ID value associated with an existing pinhole MUST be unique. Each PINHOLE_ID offered to a controller in an Offer message SHOULD be unique, though it is acceptable for a controller to offer the same value twice as long as it detects this condition and does not attempt to install multiple pinholes with the same PINHOLE_ID values.

The context field carries responder-specific information to distinguish context. Examples of contexts include interface identifiers and identifiers for virtualized firewalls or NATs.

7.8. APPLICATION_ID

```
typedef struct {
       uint16_t id_no;
       char     version[14];
} application_id_t;
```

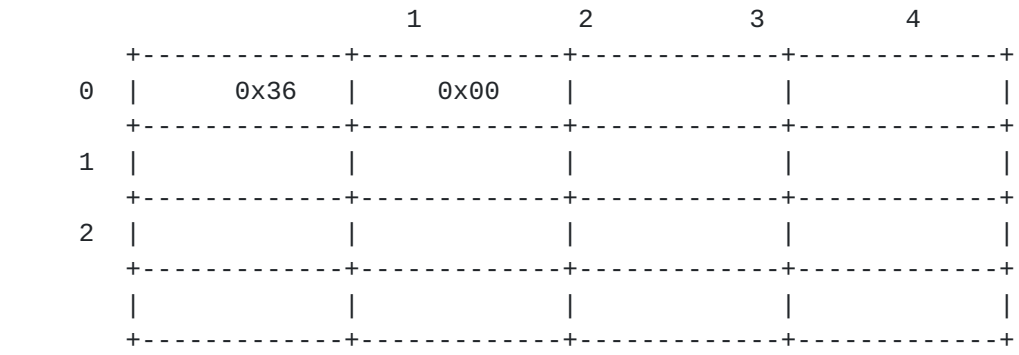
The APPLICATION_ID is a mechanism for identifying the application for which the resources (firewall pinholes, NAT table mappings) are being requested. It is expected that the information will be used as input to a policy-based decision whether or not to grant the request.

The id_no member represents the application itself. It MUST use the well-known port number, allocated by the IANA, for the application. In the case where there is no well- known port, such as a Cisco-proprietary protocol for which no well-known port has been requested, a number outside the IANA registered port range MUST be allocated.

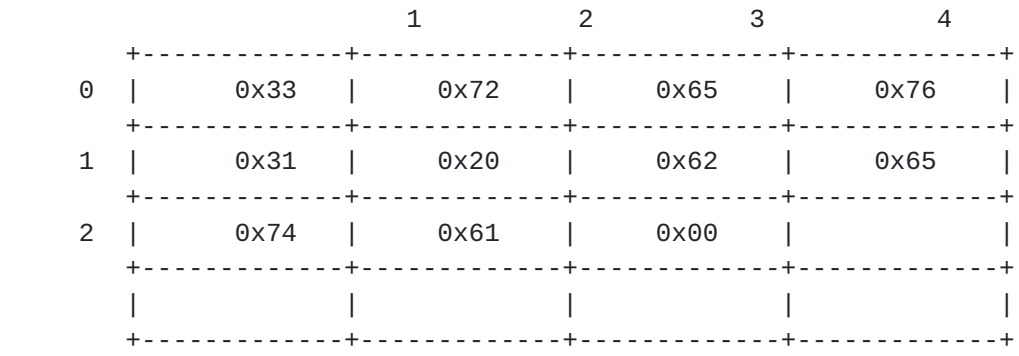
There are cases in which a protocol uses a dynamically allocated port number -- for example, non-tunneled H.245 or RTP. In those cases the application_id SHOULD be that of the parent protocol (say, H.225 in the case of H.245 or SIP or RTSP in the case of RTP).

The version field carries a null-terminated ASCII representation of the version number. If the version string is 14 octets long no null termination is necessary; if the version string is more than 14

octets in length it MUST be truncated to 14 octets. For example, version "6" would appear:



while version "3rev1 beta" would be represented as:



8. Traffic Flows

A flow is defined as a set of IP packets passing through a particular point in the network that match one or more IPV4_SELECTOR or IPV6_SELECTOR elements. A selector A matches a packet P when the following seven conditions hold:

1. (A.src_addr AND A.src_mask) equals (P.src_addr AND A.src_mask)

2. (A.dst_addr AND A.dst_mask) equals (P.dst_addr AND A.dst_mask)

3. A.src_port_lo <= P.src_port

4. A.src_port_hi >= P.src_port

5. A.dst_port_lo <= P.dst_port

6. A.dst_port_hi >= P.dst_port

7. A.protocol equals P.protocol, or A.protocol equals zero.

When the src_addr or dst_addr of a selector is zero, the selector will match any source address or destination address, respectively. Similarly, a selector with a protocol value of zero will match any protocol. In the future, additional selector TLVs may be defined in order to express additional information (such as TCP or IP options or MPLS labels) or to express a particular sort of flow more compactly (such as the UDP port pairs often used in RTP). However, any additional selector TLVs will describe what the traffic flow of interest is, and will not describe what should be done with it. If, for example, a controller needs to express that a particular flow should have QoS applied to it, or should be rate-limited, or should be monitored or audited, then the specification of the responder behavior with respect to the traffic flow MUST be expressed using TLV elements that are separate from the selector elements.

9. Responder Discovery

In some cases, the initiator may need to open a pinhole, but not know the responder to which the Open Pinhole exchange should be addressed. The authorization interception feature can be used to find the appropriate firewall, in some cases.

A network device that implements the authorization system has the capability of intercepting packets that it is forwarding, and acting on those packets if appropriate, and forwarding them otherwise. A firewall implementing the Authorized IP Firewall Control application MAY intercept Start messages for that application. An initiator MAY send an Authorized IP Firewall Control message addressed to an end host behind a responder onto which a pinhole should be installed.

Note that in order to discovery multiple nested firewalls, a firewall implementing Start message interception SHOULD forward the Start message on towards its destination.

This firewall discovery method will work only when the responder is on both the path from the initiator to the end host and the path(s) from the end host to the other devices to which it intends to communicate over the pinhole. When a network is multi-homed, this may not be the case.

10. Authorizations

We define the following TLV formats, for native authorization:

The FW_CTL TLV format has a zero-length Value field. It grants permission to its holder to control responders to allow the flow of traffic described by the TLV elements that follow it. If there are no flow-description elements that follow it, then a FW_CTL element does not actually convey any authorizations. For example, the statement

```
FW_CTL IPV4_SELECTOR1 IPV4_SELECTOR2
```

grants permission to control any responder to permit the flow of traffic described by the two selector elements that follow it.

The FW TLV format has a zero-length Value field. It grants permission to its holder to act as a responder for the traffic described by the TLV elements that follow it. If there are no flow-description elements that follow a FW element, then the element conveys no authorizations. For example, the statement

```
FW IPV4_SELECTOR1 IPV4_SELECTOR2 IPV4_SELECTOR3
```

grants permission to act as a responder and control the flow of traffic described by the three selector elements that follow it.

A single authorization element MAY contain both a FW_CTL element and a FW element. Formally, the authorizations have the format

```
((FW IPV4_SELECTOR+)|(FW_CTL IPV4_SELECTOR+))+
```

using the notation described above.

We say that selector A contains selector B, or B is contained by A, whenever every packet that matches selector B also matches selector A. This situation occurs if and only if all of the following conditions hold:

1. (A.src_addr AND A.src_mask) equals (B.src_addr AND A.src_mask)

2. (A.dst_addr AND A.dst_mask) equals (B.dst_addr AND A.dst_mask)

3. A.src_port_lo <= B.src_port_lo

4. A.src_port_hi >= B.src_port_hi

5. A.dst_port_lo <= B.dst_port_lo

6. A.dst_port_hi >= B.dst_port_hi

7. A.protocol equals B.protocol, or A.protocol equals zero.

Note that if A contains B, it is not necessarily true that B contains A. However, if A contains B and B contains C, then it follows that A contains C.

When checking authorizations against requests, a responder MUST

- verify that the selector describing the authorizations granted by the server to the initiator (I_AUTHS) are contained in the server's authorizations.

- verify that the selector describing the initiator's request is contained in the selector describing the authorizations granted by the server to the initiator.

A future version of this specification may contain other authorization elements.

11. NAT discussion

It may be the case that the firewall being controlled is co-resident with a NAT or that there is a NAT between the controller and the firewall. It may also be the case that we are controlling a stand-alone NAT. This raises some issues, some of which are addressed in this document and some of which are for future study.

11.1. Stand-alone NAT

This is straightforward -- we send requests to the NAT and the NAT returns the external address, allowing us to use the external address in protocols that make use of embedded addresses, such as VoIP protocols or streaming media protocols.

The behavior of the Authorized IP firewall control protocol is the same whether it is being used to control NATs or firewalls, with the difference lying in the data elements.

11.2. The NAT is co-resident with the firewall

In this scenario the firewall should be treated as a stand-alone NAT. That is to say, the data elements will include the NAT_TUPLE in the Request and the NAT_INFO in the Response. The reason for this is that it is assumed that an application would not want an external address if it did not also want a firewall pinhole, and it would want both resources to have the same lifetime.

When the Request arrives at the NAT/firewall device, the device MUST process the NAT request first, and the results of the NAT processing (that is to say, the NAT table mapping) passed to the firewall function as part of the pinhole descriptor. If the NAT table mapping is not acquired first and used in the filter description, the filter rule that will be installed not be correct for processing inbound (external->internal) traffic.

The firewall pinhole for outbound traffic will contain:

Source address: untranslated internal address

Source port: untranslated internal port

Destination address: untranslated peer address

Destination port: untranslated peer port

The firewall pinhole for inbound traffic will contain:

Source address: untranslated peer address

Source port: untranslated peer port:

Destination address: translated (external) address

Destination port: translated (external) port

11.3. There is a NAT between the controller and the firewall

The issue here is that NAT is transparent to the endpoint. In the typical case an endpoint does not know whether or not there is a NAT along the path between it and its peer. Even if communication fails the endpoint cannot know whether the failure was caused by a NAT interaction or some other network failure. When a NAT is present an endpoint will not know the external address, which is the correct one, to send in a pinhole request. A pinhole descriptor will contain the endpoint's local address and port (the address on the network interface card and the port number assigned by the local TCP or UDP stack). In the case where a NAT is present between an endpoint and the firewall on which it's requesting a pinhole, the address and port local to the endpoint are not the address and port visible outside the NAT, and the pinhole descriptor will not reflect that. The pinhole will be for the "wrong" internal address.

12. NAT call flow examples

Below we show some sample SIP call flows, using the Authorized IP FW Control application to acquire NAT table mappings (and open pinholes if appropriate) to provide traversal capabilities. In all cases the SIP messages are shown in black and the AFWC messages are shown in red. These call flows assume the use of a 4-message exchange, which may be reduced to two messages in future versions of this document. In these examples, the endpoints or their call control servers always function as initiators and the NATs always function as responders.

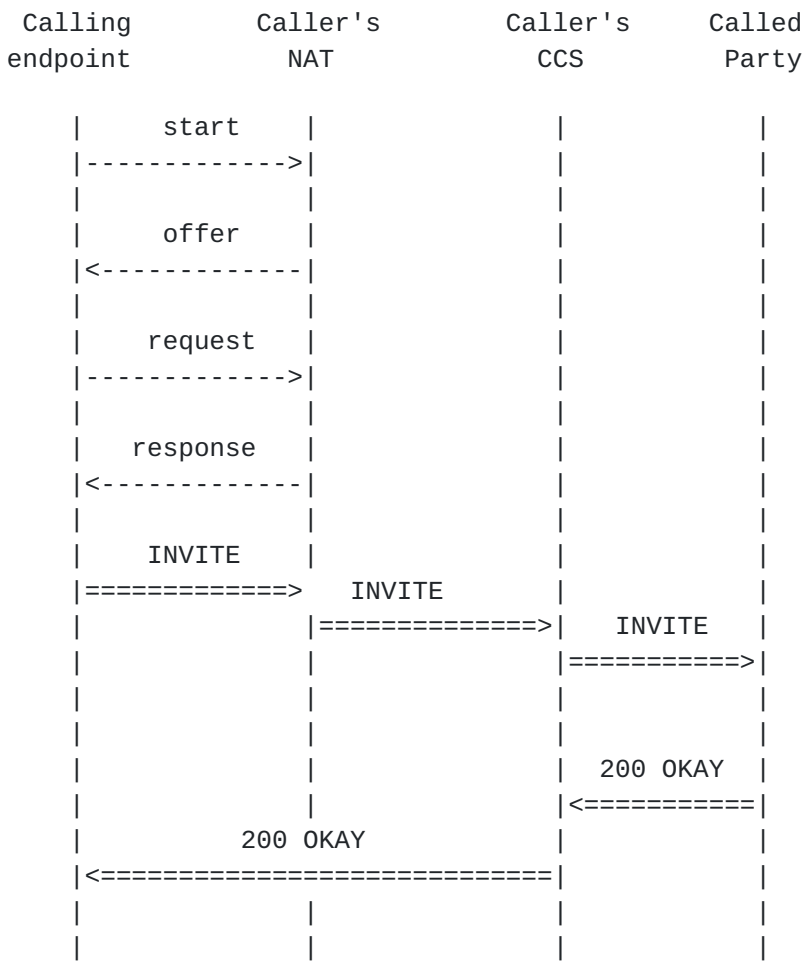
Because SIP (and, for that matter, H.323, RTSP, and other session-oriented protocols) embed addresses in signaling messages it is necessary to acquire NAT table mappings before sending an INVITE (calling party) or a 200 OK (called party). In firewall-only applications the request may be made before or after the signaling message is sent, however that entails risking that the application signaling progresses even if firewall resources are not available.

In these examples and in actual use, the endpoint (or its agent, such as a call control server) will generally request NAT table entries only for the address/port tuples on which it will be listening for incoming media. The case in which requests must be made for mappings for outbound media is that in which the NAT does not automatically allocate mappings for new outbound data streams [ETH] i.e. the NAT is entirely controlled and only creates mappings on explicit request.

In the exceptional case that the NAT is symmetric, both internal and external addresses will need to be installed to describe a mapping.

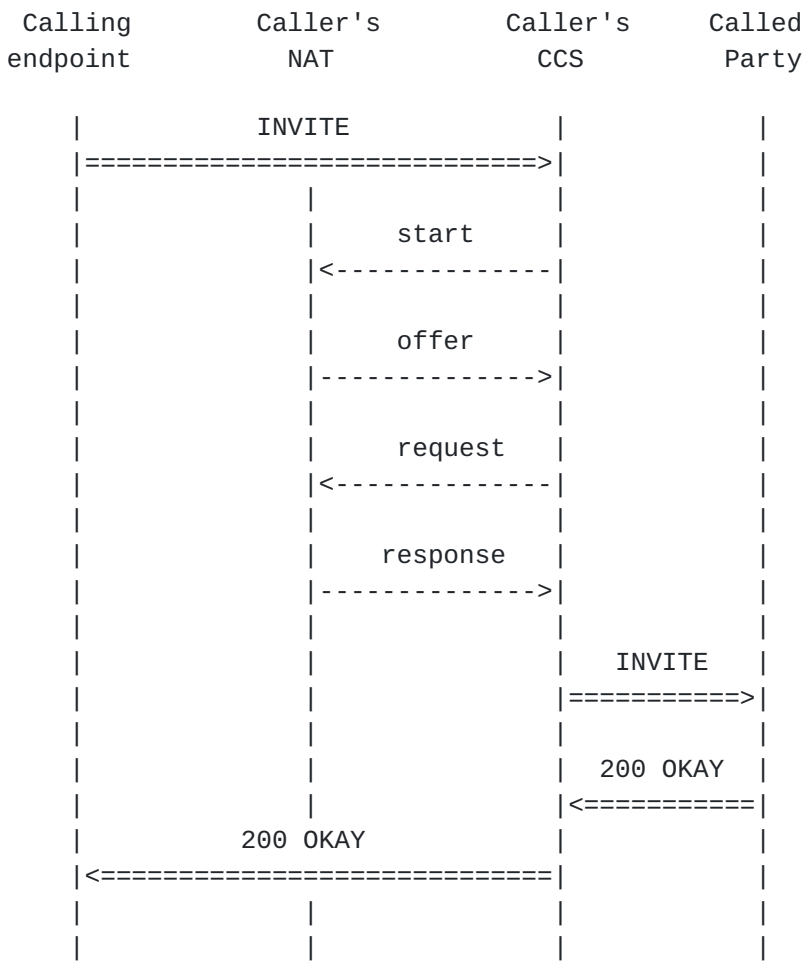
12.1. Calling endpoint is NATted, controls firewall

In this example the calling party is NATted but is routing signaling through a call control server. It is sending AFWC messages to the firewall itself. For the sake of simplicity the call flow shows the signaling being sent directly to the called party, however in actual use it is likely that the signaling would be sent to the called party's call control server.



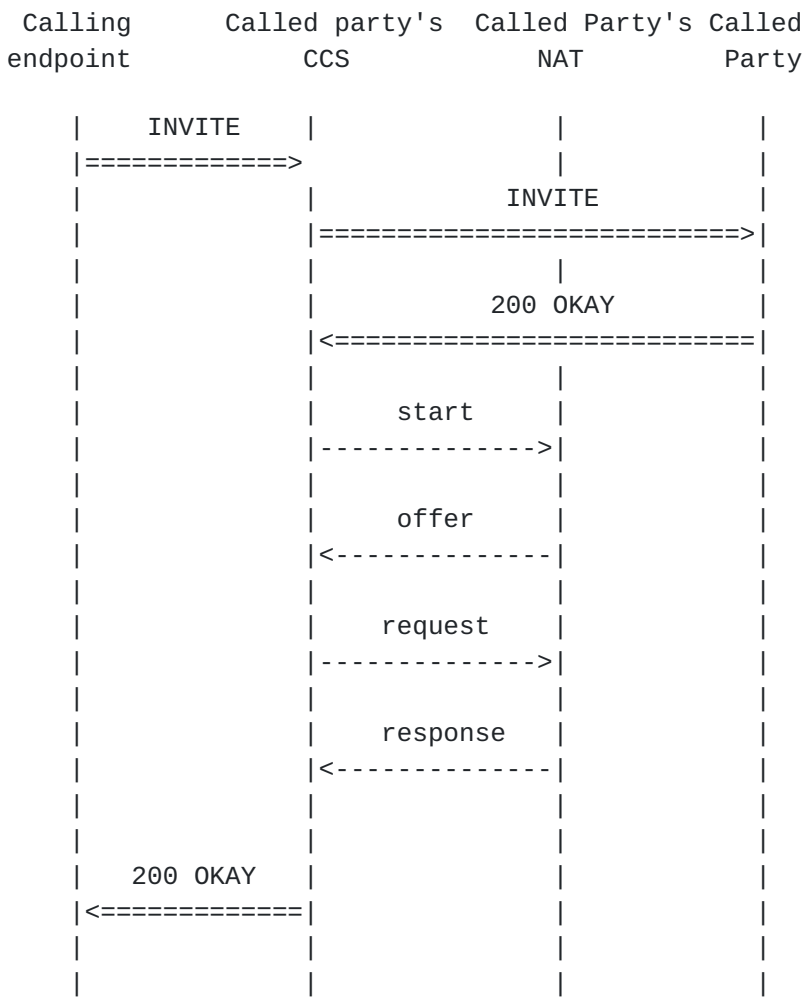
12.2. Calling endpoint is NATted, CCS controls firewall

In this example the calling endpoint is NATted, as well, however the call control server is sending AFWC requests to the firewall/NAT. Note that in this example the call control server is topologically "outside" the firewall, which has implications for assumptions about trust relationships and suggests that the notion of a "trusted interface" cannot be relied upon in this case.



12.3. Called endpoint NATted

In this case the called party is NATted but its call control server is accessible. The INVITE is sent to the called party's call control server, which forwards it to the called party. Note that the call control server cannot use AFWC to acquire a NAT table mapping until it has received the 200 response from the called endpoint; this is because it needs to send the address/port tuple on which the endpoint expects to receive media as part of the AFWC request.


Calling endpoint
Called party's CCS
Called Party's NAT
Called Party
INVITE
INVITE
200 OKAY
start
offer
request
response
200 OKAY

13. Failover

This application does not define its own failover system, and instead recommends that firewalls use the failover mechanisms that they have in place. A firewall may not be able to retain pinholes across reboots. However, if the firewall implements a checkpointing or standby feature, the pinholes SHOULD be included in the state that is checkpointed or duplicated on the standby system. The AFWC protocol allows for a hot-standby system by allowing one device to respond on behalf of another (assuming that the standby device is properly authenticated and authorized). This feature should allow existing standby systems to implement the AFWC without any additional changes.

It may be desirable to introduce a secure mechanism by which a controller can discover if a firewall has reloaded. One way to do this would be to use a 128-bit EPOCH value, which the firewall selects randomly at boot time. By including an EPOCH element in the Offer, the firewall could securely convey the current epoch to the controller. By retaining and comparing epoch values, a controller can detect if a firewall has reloaded.

14. Security Considerations

Appendix A. Acknowledgements

The authors would like to thank Raghu Gyambavantha, Eric Wang, and Jan Vilhuber for their comments and suggestions.

Authors' Addresses

Melinda Shore
Cisco Systems
809 Hayts Road
Ithaca, New York 14850
USA

Email: mshore@cisco.com

David A. McGrew
Cisco Systems
510 McCarthy Blvd
Milpitas, California 95035
USA

Email: mcgrew@cisco.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.