Internet Draft
Document: draft-stiemerling-nat-fw-config-00.txt
Expires: May 2002

M. Stiemerling
J. Quittek
NEC Europe Ltd.

November 2001

Simple NAT and Firewall Configuration (SNFC) Protocol Version 1.0

<draft-stiemerling-nat-fw-config-00.txt>

Status of this Memo

Copyright Notice

Abstract

This memo specifies a protocol for configuring Network Address Translators (NATs) and firewalls dynamically to create address bindings and open pinholes. NATs and firewalls are a problem for applications using voice and video streaming, such as IP telephony, because they need to establish voice or video channels dynamically. The Simple NAT and Firewall Control (SNFC) protocol allows clients to send requests for this purpose to serving NATs and/or firewalls. The protocol is designed to provide a very simple and basic solution that can easily be implemented and used.

Table of Contents

1. Introduction

In today's network environments the use of firewalls and Network Address Translators (NATs) is widespread. Firewalls and NATs improve network security and in the times of IPv4 address depletion NATs are keeping the Internet growing. However, NATs and firewalls also are an obstacle to many applications, because in order to traverse them, application specific and session specific configuration is required.

A good example (and the main driver for developing SNFC) is IP telephony. For a connection two voice channels - one for each direction - need to be established. Typically, UDP is used to carry voice data, but for most NATs and firewalls it is a problem to provide the required address mapping and to open corresponding pinholes for UDP traffic without manual configuration. Possible solutions are application level gateways linked to the NAT/firewall or signaling between the application and the NAT/firewall.

Providing a signaling-based solution is the main goal of the midcom working group [2,3]. The group currently discusses requirements for a signaling protocol [4]. The SNFC protocol described in this document

shall contribute to the requirements discussion by demonstrating how a very simple and straight forward approach to such a protocol can

look like. The SNFC protocol is a client/server protocol with the NAT/firewall (middlebox) serving application-level clients (midcom agents) requesting address bindings and firewall configurations. It contains only four different kinds of requests and very simple state machines that are completely specified below.

This memo describes the architectural background for the SNFC protocol and its basic concepts in section 3. Section 4 defines all SNFC messages, and section 5 specifies processing of the messages at a NAT/firewall including the complete specification of state machines. Section 6 explains how a client can open and close a SNFC session, and section 7 explains how it can request address bindings and pinhole configurations. Both sections include example message sequences for these actions. Most of the security issues are discussed in section 8. They are very important, because many network security concepts strongly depend on proper firewall configuration.

2. Terminology

This section defines the terminology that is used throughout this document.

NAT	Network Address Translation, according to [1]: Network Address Translation is a method by which IP addresses are mapped from one address realm to another, providing transparent routing to end hosts
firewall	A general firewall contains two components: a packet filter examining information of Layer 2-4 and an Application Level Gateway (ALG). In this document we restrict use of the term `firewall' refers to packet filters unless metioned explicitly
NAT/firewall	A NAT or a firewall or a combination of both
internal side	The private network of a NAT (cf. [1], Section 2.8) or the protected network of a firewall
external side	The public network of a NAT (cf. [1], Section 2.7) or firewall.
inner or internal IP Address	An IP address which is located at the

outer or external IP Address	An IP address which is located at the external side of a NAT/firewall.
transport parameter set	A set consisting of three items: an IP address, a port number, and an IP protocol type.
inner transport parameter set	Transport parameter set with inner IP address and port number.
outer transport parameter set	transport parameter set with an outer IP address and port number.
inner binding	A binding of an inner transport parameter set of a host (TP0) to and an outer transport parameter set of a NAT/firewall (TP2), as shown in Figure 1.

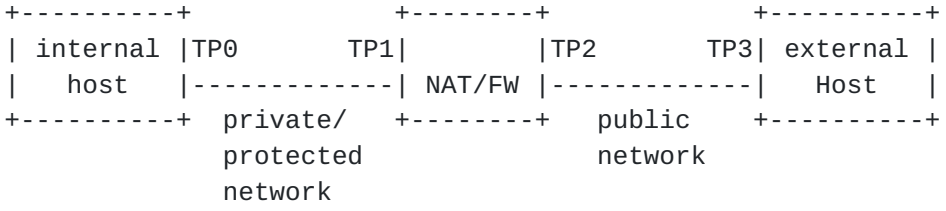

Figure 1: Addresses of internal hosts, NAT/firewall, and external hosts

outer Binding	A binding of an outer transport parameter set of a host (TP3) to and an outer transport parameter set of a NAT/firewall (TP2), as shown in Figure 1.
uni-directional binding	An inner binding or an outer binding
bi-directional binding	A combination of an inner binding and an outer binding
full binding	A bi-directional binding
pin-hole	A configuration of the firewall allowing packets matching a binding to pass the firewall. Like bindings, a pinhole may be uni-directional or bi-directional.

3. SNFC Protocol Overview

The SNFC protocol is intended to comply with the midcom architecture specified in [2]. The protocol allows a client (midcom agent) to establish a SNFC session with a server. An important component of session establishment is the authentication and authorization of the client, because configuring a firewall is a very sensitive issue of security concepts.

For the transmission between client and server TCP is used, this avoids dealing with flow control issues and gives the opportunity of using Transport Layer Security (TLS [5]) mechanism. Instead of TLS, IPSEC [6,7] may be used as well. The protocol uses ASCII encodings, because this simplifies documentation, implementation and debugging. This encoding is not considered to reduce scalability significantly.

Once a session is established, the client can request the establishments of address bindings at a NAT controlled by the server and the opening of pinholes at a firewall controlled by the server. The approach of SNFC is to use the same requests for both purposes. For example a `bind_in' request (described in sections 4 and 5) requests sends an inner transport parameter set to the server and requests the allocation of outer transport parameter set at the NAT and a binding between both sets:

- In case of a pure NAT, the outer transport parameters are allocated and a binding is established providing proper address translation.

- In case of a pure firewall, the allocated outer transport parameter set is identical with the internal one sent with the request, and just a pinhole is configured allowing packets matching the transport parameter set to pass.

- In case of a combined NAT/firewall, the outer transport parameters are allocated, a binding is established providing proper address translation, and a pinhole is configured for allowing the packets matching the binding to pass.

This integration of requests for address binding and for pinhole configuration leads to a very simple protocol with just two requests for binding and pinhole control.

4. SNFC Messages

The message formats described below are defined using the Augmented BNF (ABNF) defined in RFC 2234 [8]. The definitions for `DIGIT', `HEXDIG', `WSP', `CRLF', `CR', `VCHAR' and `LF' are imported from

appendix A of RFC 2234 and not repeated here.

4.1. Common Definitions

The following definitions are used in the subsequent chapters to define the SNFC protocol messages.

```
IPAddress      = IPv4Address / IPv6Address / "0"
IPv4Address    = 1*3DIGIT 3("." 1*3DIGIT)
IPv6Address    = 1*4HEXDIG 7( ":" 1*4HEXDIG)
Port           = 1*DIGIT
MID            = 1*DIGIT                    ; Message ID number
```

The MID is an identifier for the client, to recognize which reply belongs to which request, i.e. the MID in the reply is allways the same as in the request.

```
BID            = 1*DIGIT                    ; Binding ID number
```

The BID is the handle for the Firewall/NAT to identify the correct pin-hole and it may correlate with the corresponding pin-hole number in the Firewall/NAT. A BID of zero means not allocated BID.

```
PT             = "UDP" / "TCP" / "ICMP" / "ANY" ; IP protocol type
Authentication = 1*VCHAR
Challenge      = 1*VCHAR
Message        = 1*VCHAR
Timeout        = 1*DIGIT                    ; timeout in seconds
TPS            = IPAddress WSP Port WSP PT  ; transport param. set
TPS_in         = TPS                        ; internal TPS
TPS_ex         = TPS                        ; external TPS
Version        = "SNFC/1.0"
```

4.2. Message Definitions

The following ABNF definitions define the set of SNFC requests which can be sent from a client to a server.

```
request =  "open"     WSP MID WSP Version WSP Authentication CRLF
request =/ "close"    WSP MID CRLF
request =/ "bind_in"  WSP MID WSP BID WSP TPS WSP Timeout CRLF
request =/ "bind_out" WSP MID WSP BID WSP TPS WSP Timeout CRLF
```

The `open' and the `close' request are exclusively used for session control. The `bind_in' and the `bind_out' request are exclusively used for binding and pinhole control.

4.3. Replies

Every reply message starts with a three digit reply code and ends with `CRLF'. The three digits in a reply code have a special meaning.

The first digit identifies the class of a reply message. The

following classes exist:

1yz	transient positive response
2yz	permanent positive response
3yz	transient negative response
4yz	permanent negative response
5yz	asynchronous notification

The classes 1yz and 3yz are currently not used by SNFC version 1.0. They are defined only for future SNFC extensions.

The second digit encodes the specific category. The following categories exist:

x1z	syntax errors that don't fit any other category
x2z	replies for requests concening session control
x3z	replies for requests concening binding and pinhole control

The third digit gives a finer gradation of meaning in each category specified by the second digit. Below is the ABNF definition of all reply messages and codes:

```
Reply =/ "410" WSP MID CRLF         ; syntax Error
Reply =/ "411" WSP MID CRLF         ; unknown or illegal request
Reply =/ "510" WSP Message CRLF     ; illegal message

Reply =  "220" WSP MID CRLF         ; OK
Reply =/ "420" WSP MID CRLF         ; protocol version mismatch
Reply =/ "421" WSP MID WSP Challenge CRLF  ; authentication failed
Reply =/ "520" WSP Message CRLF     ; session closed

Reply =/ "231" WSP MID WSP BID WSP TPS
                WSP Timeout CRLF    ; OK for uni-direct. binding
Reply =/ "232" WSP MID WSP BID WSP TPS_in WSP TPS_ex
                WSP Timeout CRLF    ; OK for full binding
Reply =/ "233" WSP MID WSP BID CRLF ; binding removed
Reply =/ "430" WSP MID CRLF         ; unknown or illegal BID
Reply =/ "431" WSP MID CRLF         ; binding Refused
Reply =/ "432" WSP MID CRLF         ; illegal IP Address
Reply =/ "433" WSP MID CRLF         ; protocol type not supported
Reply =/ "434" WSP MID CRLF         ; illegal port number
Reply =/ "435" WSP MID CRLF         ; cannot modify binding
Reply =/ "530" WSP BID CRLF         ; binding timed out
```

5. Message Processing in Server

This section describes the processing steps performed by a SNFC server after receiving a message. Common to all incoming messages is that first the syntax is checked. Then we distinguish messages concerning session control ans messages concerning the control of address bindings. For both kinds, we define a state machine and discuss states and transitions.

5.1. Syntax Checking

When the server receives a message, it first tries to recognize a request consisting of the command string and a message identifier (MID). Messages generated by syntax checking are:

- `410' reply
- `411' reply
- `510' reply

If the server is not able to extract both the command string and the message identifier, then the message is discarded. An asynchronous `510' reply may be generated in this case. Otherwise, the command string is checked to be valid, i.e. to be one of the strings `open', `close', bind_in', bind_out', `refresh', or `remove'. If the string is invalid, a `411' reply is sent and processing of the message stops. If a syntax error is detected, a `410' reply is sent and processing of the message stops. Otherwise, the message is further processed as described below.

5.2. Session State Machine

The session state machine has just two states: CLOSED and OPEN. Transisions between these states only appear in conjunction with one or two of the following messages:

- `open' request
- `close' request
- `220' reply
- `420' reply
- `421' reply
- `520' reply

Additionally, a `510' reply may be generated in the CLOSED state as described below.

Figure 2 shows the state machine of a single session with all possible transitions. Please note that a server may serve several clients at a time by running several concurrent sessions.

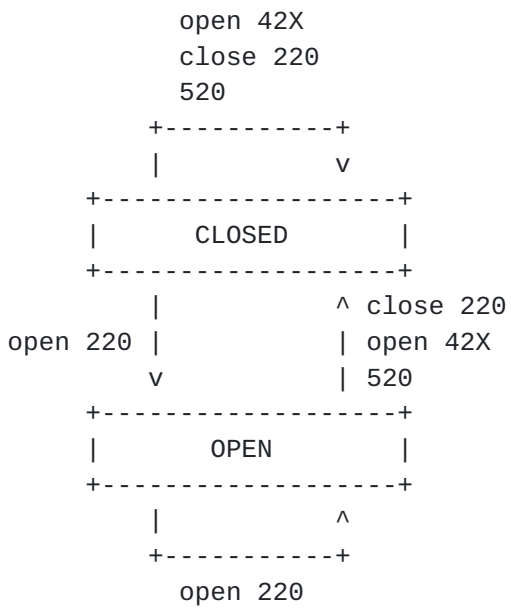

Figure 2: Session state machine

The initial state of all sessions is CLOSED. If a client establishes a connection to the server by successfully creating a socket, then a session in state CLOSED is assigned to this connection. For closed sessions only two requests are accepted: `open' and `close'. All other requests received in this state are discarded. An asynchronous `510' reply may be generated in this case. In the OPEN state, all SNFC messages are accepted. However, only `open' and `close' messages have an impact on the session state.

5.2.1. Transistions from State CLOSED

If an `open' request is received, then first the contained `Version' string is checked. If the version indicated by the string is not compatible to one of the versions supported by the server, then a `420' reply is generated, the connection is closed, and the state machine remains in state CLOSED. Otherwise, the contained `Authentication' string is analysed. If the authentcation check is successful, a `220' reply is generated and the session enters state OPEN. Otherwise, a `421' reply is generated and the session remains in state CLOSED with the connection still established.

At any time the server may generate an asynchronous `520' reply followed by closing the connection. In this case the session will remain in state CLOSED. Particularly, the server may generate a `520' reply, if a connection is established and the time the session remains in state CLOSED exceeds a given timeout value.

5.2.2. Transistions from State OPEN

If a `close' request is received, then the server generates a `220' reply and the session enters state CLOSED with the connection being closed. The same transition occurs if the server decides to close the session. Then it generates a `520' reply, closes the connection, and enters session state CLOSED.

If an `open' request is received, it is processed as in the CLOSED state. First the contained `Version' string is checked. If the version indicated by the string is not compatible to one of the versions supported by the server, then a `420' reply is generated, the connection is closed, and the state machine enters state CLOSED. Otherwise, the contained `Authentication' string is analysed. If the authentcation check is successful, a `220' reply is generated and the session remains in state OPEN. Otherwise, a `421' reply is generated and the session enters state CLOSED with the connection kept established.

At any time the server may generate an asynchronous `520' reply followed by closing the connection. In this case the session will enter state CLOSED.

5.3. Binding State Machine

When the session state machine is in state OPEN, the server accepts further requests regarding bindings. The state machine of a binding contains four states: BID_UNUSED, BIND_IN_ONLY, BIND_OUT_ONLY, FULL_BINDING. Transistion between the states occur in conjunction with the following messages:

- `bind_in' request
- `bind_out' request
- `23X' replies
- `43X' replies
- `530' reply

All of these requests and replies refer to exactly one binding which is indicated by the BID field of the message. The BID uniquely identifies a binding. BIDs are allocated and assigned to bindings by the server. If a request contains a BID which unused because it is not assigned to any binding, then the server will discard the request and generate a `430' reply. BID 0 is an exception, it is reserved for a special purpose and it is never assigned to an existing binding.

For all BIDs other than 0 there exists a state machine as shown in Figure 3. If the server receives a `bind_in` or a `bind_out' request containing BID 0 then a new BID is allocated. Before the request can

be executed, a new binding state machine is instantiated for this BID

with the initial state BID_UNUSED.

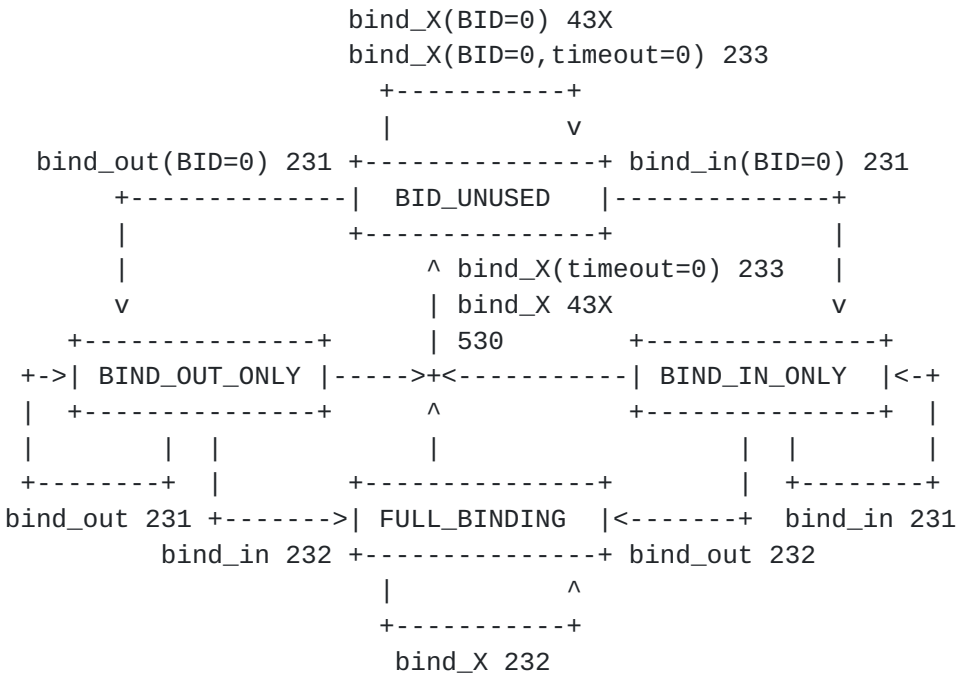

Figure 3: Binding state machine

When a binding state machines makes a transition to the state BID_UNUSED (including transitions from state BID_UNUSED), then the BID is considered to be unused. The client should not use this BID any further, because it may be re-used by the server when allocating a new BID.

5.3.1. Transport Parameter Set Checking

When a `bind_in` or a `bind_out' request is processed, the transport parameter set contained in the message is checked. The checking procedure is common for all states:

- The IP address is checked whether it is an inner IP address in case of a `bind_in' request or an outer address in case of a `bind_out' request, respectively. If the check fails or if the IP address is considered invalid for some other reason, then the server generates a `432' reply.

- The protocol type is checked, whether it is valid and supported. If the check fails, a `433' reply is generated.

- The port number is checked whether it is valid. If the check fails, a `434' reply is generated.

If the checks are successful the server may perform further checks on the combination of the elements of the transport parameter set, for example it may check available resource or it may consult a policy-based access control system checking whether the client is allowed to make the current request. If one of these checks fails, then the server generates a `431' reply. Otherwise the server will continue with establishing the requested binding.

5.3.2. Transitions from State BID_UNUSED

In state BID_UNUSED, only `bind_in` and `bind_out' requests with BID 0 can have an effect on the state machine. The server first performs the parameter checks described above. If one of them fails, the binding state machine remains in state BID_UNUSED.

If the timeout specified by the request message is 0, then the server generates a `233' reply and the binding state machine remains in state BID_UNUSED.

If a `bind_in' request passes all checks described above, then the server allocates an external address provided by the Firewall/NAT and it establishes a uni-directional binding of the requested transport parameter set with the new allocated address. Then the state machine for this BID enters the state BIND_IN_ONLY and the server generates a `231' reply reporting

- the BID allocated for this binding,

- the transport parameter set of the bound external address,

- the timeout in seconds after which the binding will automatically be removed by the server. The timeout chosen by the server is less than or equal to the value specified by the client in the `bind_in' request message.

If a `bind_out' request passes the checks, the server allocates an internal address provided by the Firewall/NAT and it establishes a uni-directional binding of the requested transport parameter set to the new allocated address. Then the state machine for this BID enters the state BIND_OUT_ONLY and the server generates a `231' reply reporting

- the BID allocated for this binding,

- the transport parameter set of the bound internal address,

- the timeout in seconds after which the binding will automatically be removed by the server. The timeout chosen by the server is less than or equal to the value specified by the client in the

`bind_out' request message.

5.3.3. Transitions from BIND_IN_ONLY and BIND_OUT_ONLY

In state BIND_IN_ONLY and BIND_OUT_ONLY a uni-directional address binding is established. This might be sufficient for UDP connections, but not for TCP. In these states the client can request to extend the binding to a bi-directional one by sending a `bind_out' request in state BIND_IN_ONLY or by sending a `bind_in' request in state BIND_OUT_ONLY, respectively. The request must contain the BID of the already existing uni-directional binding.

If the request fails the transport parameter set checking, then the server generates the according `43X' reply and also it removes the already established uni-directional binding. The binding state machine then enters state BID_UNUSED. If the timeout specified by the request message is 0, then the server generates a `233' reply and removes the already established uni-directional binding. The binding state machine then enters state BID_UNUSED.

Otherwise, if the request passes the checks of the transport parameter set, the server creates a bi-directional binding to the already known BID and chooses a timeout less than or equal to the value specified by the request. The server generates a `232' reply reporting the chosen timeout and the internal and the external address allocated at the NAT/firewall. The BID state machine enters state FULL_BINDING.

Other options for the client are requests for

- updating the timeout,

- modifying the binding while keeping the BID and keeping the address allocated by the server,

- removing the binding.

Each of these actions requires the client sending another `bind_in' request in state BIND_IN_ONLY or sending another `bind_out' request in state BIND_OUT_ONLY.

For timeout update and for binding removal the client must use exactly the transport parameter set already contained in the previous requests for this BID. The server must ensure that such unchanged transport parameters pass the transport parameter check.

If the timeout specified in the request message is 0, the server will remove the binding and generate a `233' reply, and the binding state machine enters state BID_UNUSED.

If the timeout specified in the message is larger than 0, the server

must process an update of the binding's timeout without any

interruption of the NAT/firewall operation for this binding. The server chooses a timeout less than or equal to the value in the request message and reports it by generating `231' reply. The binding state machine remains in its state.

If the transport parameters specified in the request message differ from the ones used in the previous message, then they are checked by the server. A `43X' reply is generated on failure of one of these checks. Then the server is supposed to modify the binding. If it is not capable of doing so, it generates a `435' reply, removes the binding and the binding state machine enters state BID_UNUSED.

Otherwise, the server will replace the established binding with a new one. The modified binding will keep the address allocated by the server, but bind it to the transport parameter set contained in the message. The BID remains unchanged. Then the server generates a `231' reply confirming the change and reporting the chosen timeout. The binding state machine remains in its state.

At any time, the server may remove the binding. It must do so at latest if the timeout expires. But also at any earlier time it may do so, for example if the policy for granting binding requests has changed, if a mis-use of the binding was detected, or if the server cannot continue the operation of the binding for technical reasons. In such a case the server generates an asynchronous `530' message indicating the removal of the binding and the binding state machine enters state BID_UNUSED.

5.3.4. Transitions from State FULL_BINDING

In state FULL_BINDING the client can request to

- update the timeout,
- remove the binding,
- modify the binding while keeping the BID and keepig the addresses allocated by the server.

For timeout update and for binding removal the client can use a `bind_in' request or a `bind_out' request. The request must use exactly the transport parameter set already contained in the previous `bind_in' request or `bind_out' request, respectively. The server must ensure that such unchanged transport parameters pass the transport parameter check.

If the timeout specified in the request message is 0, the server will remove the binding and generate a `233' reply, and the binding state machine enters state BID_UNUSED.

If the timeout specified in the message is larger than 0, the server must process an update of the binding's timeout without any interruption of the NAT/firewall operation for this binding. The server chooses a timeout less than or equal to the value in the request message and reports it by generating `231' reply. The binding state machine remains in its state.

If the transport parameter set specified in the request message differ from the ones used in the previous message, then they are checked by the server. A `43X' reply is generated on failure of one of these checks. Then the server is supposed to modify the binding. If it is not capable of doing so, it generates a `435' reply, removes the binding and the binding state machine enters state BID_UNUSED.

Otherwise, the server will replace the established binding with a new one. The modified binding will keep the addresses allocated by the server, but bind it to the transport parameter set contained in the message. The BID remains unchanged. Then the server generates a `232' reply confirming the change and reporting the chosen timeout. The binding state machine remains in state FULL_BINDING.

At any time, the server may remove the binding. It must do so at latest if the timeout expires. But also at any earlier time it may do so, for example if the policy for granting binding requests has changed, if a mis-use of the binding was detected, or if the server cannot continue the operation of the binding for technical reasons. In such a case the server generates an asynchronous `530' message indicating the removal of the binding and the binding state machine enters state BID_UNUSED.

6. Controling SNFC Sessions

After a secure TCP connection has been established between client and server (for example by using TLS [5] or IPSEC [6][7]), the SNFC session requires an initial authentication of the client. This might be technically superfluous, for example if the client already authenticated itself when establishing the connection, but it is an inevitable step of establishing a SNFC session. The client authenticates itself by sending an `open' request containing an authentication string. This might be a shared secret (cookie) in simple authentication systems.

For more secure challenge-reply authentication, the client first sends an `open' request with an arbitrary authentication string. When - as expected - the authentication failed, the server the server returns a challengestring in the following `421' reply. Then the client sends a second `open' request now containing the correct

authentication string derived from the challenge string. This procedure is illustrated by the following Example (a).

For message flow examples in this memo we use the following indication of the direction of a message:

```
C->S: from the client to the server
C<-S: from the server to the client
--: comment line
. . .: some unspecified messages
```

Example (a): successful authentication

```
-- TCP connection establishment and TLS or IPSEC establishment
-- client sends dummy authentication string: 0
C->S: open 1300 SNFC/1.0 0
-- server returns challenge string
C<-S: 421 1300 13e66f34b7416ab9389ccc5b441290aa
-- client sends correct authentication
C->S: open 1301 SNFC/1.0 ab54346de6933ff4556a1b23efd70082
C<-S: 220 1301
-- session now in state OPEN
. . .
-- SNFC message exchange
. . .
-- client closes session
C->S: close 40163
C<-S: 220 Ok
-- session now in state CLOSED
-- connection terminated
```

If the client fails to authenticate itself after a number of invalid `open' requests, the server may disconnect itself from the client. The server in Example (b) disconnects after two invalid `open' requests.

Example (b): failed authentication

```
-- TCP connection establishment and TLS or IPSEC establishment
-- client sends invalid authentication string
C->S: open 55000 SNFC/1.0 34EFA
-- server returns challenge string
C<-S: 421 55000 13e66f34b7416ab9389ccc5b441290aa
-- client sends invalid authentication string
C->S: open 55333 SNFC/1.0 125d5
C->S: 421 55333 13e66f34b7416ab9389ccc5b441290aa
-- server in state CLOSED, client disconnected
```

The client closes a session by sending a `close' request. All established bindings configured by this client will remain established until their timeout expires. Also the server may terminate an open session by sending an asynchronous `520' reply.

7. Controling Bindings and Pinholes

The client can request to establish, entend, modify and remove uni-directional and bi-directional bindings and pinholes. All of these requests, are variants of a `bind_in' or a `bind_out' request. The `bind_in' request name is derived from `establish a binding of the transport parameter set of an external address of the NAT/firewall to the parameter set of an internal address'. A successful `bind_in' request allows a data stream matching the corresponding parameter set to pass the NAT/firewall from the external network to the internal one. The `bind_out' request is defined analogously.

Each binding has a timeout value, which determines when a binding is automatically removed by the SNFC server. The client makes a timeout proposal in his `bind_in' or `bind_out' request and the server may accept this proposal or choose a smaller timeout value. For example the server may be configured to accept all timeout values up to a predefined maximum value. The server informs the client about the chosen timeout in by the next reply.

Control of bindings nad pinholes is illustrated by Example (c) showing the establishment and removal of a uni-directional UDP binding and pinhole.

Example (c): control of uni-directional UDP binding and pinhole

```
   -- server is in state OPEN.
   -- request with BID=0 for binding inner IP address 10.11.1.45
   --   and UDP port 16175 for 180 seconds
   C->S: bind_in 2044 0 10.11.1.45 16175 UDP 180
   -- server allocates external IP address 195.37.70.5 and UDP port
   --   13222 and binds it to the internal transport parameter set
   --   for 180 seconds. BID=1248.
   C<-S: 231 2044 1248 195.37.70.5 13222 UDP 180
   -- binding established and pinhole open
   -- no more messages concerning this BID for 245 seconds
   . . .
   -- binding and pinhole not needed anymore
   -- request to remove by setting timeout to 0
   C->S: bind_in 2067 1248 10.11.1.45 16175 UDP 0
   -- binding and pinhole removed
   C<-S: 233 2067 1248
```

Example (d) shows the message flow of a similar request, but in this case the server is a pure firewall without any NAT function. Therefore the allocated transport parameter set is equal to the one in the request.

Example (d): control of uni-directional UDP pinhole only

```
      -- server is in state OPEN.
      -- request with BID=0 for binding inner IP address 195.37.70.163
```

```
      --   and UDP port 16175 for 180 seconds
      C->S: bind_in 2144 0 195.37.70.163 16175 UDP 180
      -- server does not allocate its own address, but it opens
      -- a pinhole for the requested transport parameter set.
      C<-S: 231 2144 1249 195.37.70.163 16175 UDP 180
      -- pinhole open
      -- no more messages concerning this BID for 245 seconds
      . . .
      -- pinhole not needed anymore
      -- request to remove by setting timeout to 0
      C->S: bind_in 2164 1249 195.37.70.163 16175 UDP 0
      -- pinhole removed
      C<-S: 233 2164 1249
```

The message flow for controling a bi-directional binding and pinhole is illustrated by Example (e). It also shows the extension of the timeout.

Example (e): control of bi-directional TCP binding and pinhole

```
      -- server is in state OPEN.
      -- request with BID=0 for binding inner IP address 10.11.1.50
      --   and TCP port 4524 for 540 seconds
      C->S: bind_in 8888 0 10.11.1.50 4524 TCP 540
      -- server allocates external IP address 195.37.70.5 and TCP port
      --   17250 and binds it to the internal transport parameter set
      --   for 300 seconds. BID=1250.
      C<-S: 231 8888 1250 195.37.70.5 17250 TCP 300
      -- request with BID=1250 for binding outer IP address
      --   134.169.34.13 and TCP port 22343 for 540 seconds
      C->S: bind_out 8889 1250 134.169.34.13 22343 TCP 540
      -- server allocates internal IP address 10.11.1.2 and TCP port
      --   5537 and binds it to the external transport parameter set
      --   for 300 seconds.
      C<-S: 232 8889 1250 10.11.1.2 5537 TCP 195.37.70.5 17250 TCP 300
      -- no more messages concerning this BID for 280 seconds
      . . .
      -- binding and pinhole used and need longer as prior granted
      -- request to extend timeout by 200 seconds
      C->S: bind_in 9023 1250 10.11.1.50 4524 TCP 260
      -- request granted
      C<-S: 232 9023 1250 10.11.1.2 5537 TCP 195.37.70.5 17250 TCP 260
      -- no more messages concerning this BID for 120 seconds
      . . .
      -- binding and pinhole not needed anymore
      -- request to remove by setting timeout to 0
      C->S: bind_out 9077 1250 134.169.34.13 22343 TCP 0
      -- binding and pinhole removed
      C<-S: 233 9077 1250
```

The last Example (f) shows the rejection of a request, because an illegal internal IP address is used.

Example (f): Rejection of illegal internal IP address

```
-- server is in state OPEN
C->S: BIND_IN 458 0 102.12.12.251 1254 UDP 300
C<-S: 432 458
-- no new binding or pinhole established
```

8. Security Considerations

By their nature Firewalls and NATs are a very sensitive points concerning network security. In general it appears to be contradictive to open a port at a firewall for configuring pinholes, because this might make the firewall vulnerable. Therefore, effective means are required for inhibiting mis-use of the SNFC service.

A SNFC server should use a restricted list of clients that are allowed to use the service. Beyond checking the clients IP address and requiring authentication when building up a secure TCP connection with TLS or IPSEC., the server should expect the client to authenticate itself by using a shared secret or based on a public key infrastructure.

The TCP connection also needs to be protected for ensuring integrity of the requests made by the client. Finally, confidentiality of the data exchang between client and server is required to hide information about the participants of communication services that are enabled by SNFC.

9. References

[1] Srisuresh,P., and Holdrege, M., "IP Network Translator (NAT) Terminology and Considerations", RFC 2663, August 1999

[2] Srisuresh, P., Kuthan, J., Rosenberg, J., Molitor, A., Rayhan, A., "Middlebox Communication Architecture and framework", Internet Draft, work in progress, <draft-ietf-midcom-framework-04.txt>, October 2001

[3] Huitema, C., "MIDCOM Scenarios", Internet Draft, work in progress, <draft-ietf-midcom-scenarios-02.txt>, May 2001

[4] Swale, R.P., Mart, P.A., Sijben, P., "Middlebox Control (MIDCOM) Protocol Architecture and Requirements", Internet Draft, work in progress, <draft-ietf-midcom-requirements-02.txt>, July 2001

[5] Dierks, T., Allen, C., "The TLS Protocol Version 1.0", RFC 2246, January 1999

[6] Kent, S., and Atkinson, R., "IP Authentication Header", RFC 2402, November 1998

[7] Kent, S., and Atkinson, R., "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998

[8] Crocker, D., and Overell, P., "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997

10. Authors' Address

Martin Stiemerling
NEC Europe Ltd.
Network Laboratories
Adenauerplatz 6
69115 Heidelberg
Germany

Phone: +49 6221 90511-13
Email: stiemerling@ccrle.nec.de

Juergen Quittek
NEC Europe Ltd.
Network Laboratories
Adenauerplatz 6
69115 Heidelberg
Germany

Phone: +49 6221 90511-15
EMail: quittek@ccrle.nec.de

11. Full Copyright Statement

the copyright notice or references to the Internet Society or other