GFD-R-P.196
FVGA-WG
http://forge.gridforum.org/sf/projects/fvga-wg

Ralph Niederberger, Research Center Jülich
May 15, 2012

Firewall Traversal Protocol (FiTP)

Status of This Memo

Recommendation Proposed (R-P).

Copyright Notice

Abstract

This memo is a draft specification of the Firewall Traversal Protocol (FiTP).

Firewalls control traffic flows between internal and external network segments. Mostly traffic initiated from internal networks to external networks is allowed, but traffic coming from external networks must be explicitly allowed. The rules that specify which packets may traverse a firewall and which not are normally configured by firewall administrators by hand. To speed up such kind of access list changes, it would be desirable to dynamically signal access requests and automatically change those access lists. Though firewalls know about the behavior and requirements of some protocols like FTP, SIP and H.323, a general protocol, which could be used for signaling dynamically required access rules, is not available until now.

This paper proposes a protocol, which would allow such signaling in a secure manner. Firewalls which have installed a corresponding inspection module could be configured automatically, which would ease the configuration of such systems a lot.

The proposed protocol (FiTP) standardizes the control connection between an external application client and an internal authorization server. Firewalls located on this communication path can read this communication. The protocol can be used in two ways. First, a firewall aware of FiTP, could automatically allow connections signaled by authorized users. Secondly, an intermediate solution could be implemented, so that firewalls unaware of FiTP could be configured by the server process, which is the end point of the FiTP control connection. Via this approach a smooth transition would be possible. Installations having old firewall hard- and/or software could use the new protocol already and must not wait until a new FiTP aware system can be installed in the fare future.

Contents

1 Introduction

Today's advantages of communication possibilities between sites worldwide using the global INTERNET imply vice versa severe risks to locally attached systems. Systems which are not secured adequately, could be hacked and misused or even data destroyed.

To protect local systems against security risks from outside, firewalls are used to add an additional layer of defense against attackers. Firewalls control traffic flows between internal and external network segments. A common rule is that traffic originating from an inside network to the outside is allowed, but traffic initiated from outside must be explicitly configured.

Most firewalls are manually configured by firewall administrators. Statefull firewalls implement dynamic rules, triggered by packet inspection. By examining the TCP handshake in the transport layer, a dynamic rule can be added to allow return traffic. By examining application data, a firewall can open additional ports to allow traffic flows for protocols that use parallel flows., such as FTP, H.323 and GridFTP. For example, a firewall may contain a static rule for the control connection of the FTP protocol, and inspect the control messages in transit. Data connections used by FTP are signaled via the control connection, so that the firewall is able to recognize such requests.

Since many Grid applications are using also an unknown number of "n" ad-hoc parallel sessions, where "n" is dependent e.g. on the number of Grid nodes available, data streams needed, or data servers available, it would be desirable to have defined a standardized protocol, which allows to signal the need of those dynamic traffic flows to the firewall systems located on the way between source and destination.

The objective of FiTP is (1) to set up a secure authenticated connection between client and server over which (2) the need of dynamically required data connections can be signaled and (3) the firewalls located on the path between source and destination can change their local access lists accordingly.

It should be mentioned that the FiTP protocol can be used for signaling data connections only which use the same path as the control connection is using, since only firewalls which are on this path can read the control messages sent and are able to configure their access lists accordingly.

If the FiTP protocol extension is not implemented within a firewall, i.e. the firewall does not have implemented code being able to analyze FiTP messages, an intermediate solution could be to install a signaling subroutine on the server side, i.e. an additional subroutine within the FiTP server endpoint software, which allows setting access lists at the firewall according to the access requests by CLI commands or proprietary software. Of course this special solution would imply that the FiTP endpoint knows any firewall on the communication path, which has to be configured and additionally has the rights to do so.

This paper assumes knowledge of the Transmission Control Protocol (TCP), FTP Protocol [FTP-RFC], FTP Security Extensions [FTP-XSEC] and basics of keyed-Hash Message Authentication Code (HMAC) [HMAC]. Some background in X.509 certificates and encryption methods is also helpful. Furthermore the reader should have some knowledge about Grid Applications and their issues with Firewalls [GFD-083] and [GFD-142].

2 Overview

This section provides the terminology, general definitions and background information. Furthermore the FiTP model itself is discussed. The terms defined in this section are only those that have special significance to FiTP. The protocol itself is very similar to the FTP protocol standard and therefore a lot of text could be directly copied from the FTP protocol standard. Also the text structure has been followed the FTP approach where possible.

2.1 Terminology

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 [RFC-2119].

2.2 Firewalls and Grids – Securely managing dynamic INTERNET access

The INTERNET today has become an opaque cloud of communications between constantly changing communication partners. Always changing business relations between national and international corporations makes risk management a permanent and time consuming task. While branch offices need access to most of the internal services, business partner may need access to limited resources only. Customers should be allowed to access special services only, where they can search for products they would like to purchase.

All these communications – however named, e.g. client–server communication, Web-Services, Grid or Cloud – describe a communication relationships between internal and external computer systems often requiring special configurations at firewall systems. These configurations include access for communication sessions (ports) and access to single systems or whole sub networks.

Only few firewall systems have been able to handle applications with dynamically assigned ports in the past. Some implementations exist for applications such as FTP, H.323, and SIP. But currently no general solution is available and explicitly no support for protocols used by Grid applications has been provided until now.

Often within a grid environment each institution or even worse each installation has its own firewall system. All of them have to be traversed by Grid applications. Because of the problems discussed above, project networks are placed in a demilitarized zone in most of the cases. This implies that every computer system used in the project has to be secured carefully. Wrongly configured systems lead to immediate security vulnerabilities.

In many projects supercomputers, PC-clusters and/or special systems are connected via dedicated networks assuming a “Net of Trust”, since configuring access rules for those systems and their grid applications would be much too complicated or insecure. Also access rights granted are not used for long periods often. Sometimes compromises in system security or organizational security policies are accepted only to allow special grid applications, which could not be used otherwise.

The results of those special security requirements have been administrative overhead for deployment and protection of grid environments, wildcard access rights (ports not known, so access granted to whole system), open ports for long time periods, weaker policies or no security policies at all. In general, the security level will be decreased to that of the partner installation.

So providing a standardized and widely-used control protocol allowing any kind of application to request access to internal resources in an easy to use, but secure manner is strongly desirable.

2.3 Definitions

Access-rule

An access-rule prot,sDTH,sport1,sport2,dDTH,dport1,dport2 defines the privilege of a system or subnet sDTH using source ports within the port range [sport1,…,sport2] to transfer data to a system or subnet dDTH at port range [dport1,…,dport2] using the protocol prot (IP, TCP, UDP or IPSEC). The hosts sDTH and dDTH may be different to user_CH and auth-CH. If sDTH and/or dDTH are specifying subnets, then user_CH and/or auth_CH may be outside these subnets respectively. Using prot IP or IPSEC sport1, sport2, dport1 and dport2 are ignored.

Auth-CH

The authentication and authorization control host (auth-CH) "listens" on Port L for a connection from a user host (user_CH). After connection by the user_CH the associated process auth_PI on host auth-CH authenticates the grid user and grants access to local grid resources. It receives standard FiTP commands from the user_PI and sends replies.

Auth_PI

The authentication and authorization protocol interpreter listens at host auth_CH on port L for connections of a user on host user_CH, requesting access grants by FiTP commands for its application data streams.

Control connection

A control connection is the communication path between the user_CH and the auth-CH for the exchange of commands and replies. This connection follows the Telnet protocol but additionally uses the Hashed Message Authentication Code (HMAC) [HMAC] routines even for plaintext transfers. Since HMAC-MD5 has been shown to be vulnerable [HMAC-6151], HMAC-SHA1 will be used within this protocol implementation. For details, how HMAC is used, see below.

Data connection

A full duplex connection (TCP) over which data is transferred. For UDP transfers a virtual full duplex connection is assumed, as being two serial connections between sDTH and dDTH and using ports sport and dport. The data itself transferred between two DTHs is out of the scope of this document and may be a file transfer, message passing, etc.

Data path grant

A data path grant is the access rule which has been included into the firewall configuration for a data connection granted via an FiTP control connection.

Data port

The data transfer process "listens" on the data port for a connection from the active transfer process in order to open the data connection.

dDTH

The data transfer hosts are the source of the data connection (sDTH) and the destination of the connection (dDTH). sDTH establishes the connection to dDTH. If sDTH and/or dDTH are subnets, then the data transfer hosts are located within these subnets.

Encryption

Within this standard protocol encryption and decryption is done using a cryptographic cipher block chaining mode (CBC) in combination with an IDEA block cipher. The encrypted messages are compatible with the encryption format used by the OpenSSL package.

Error recovery

Error recovery allows a user to recover from certain errors such as failure of either host system or transfer process. Within FiTP, error recovery may involve restarting a control connection process at a given checkpoint.

Firewall

A firewall is a logical object (hardware and/or software) within a network infrastructure which prevents communications forbidden by the security policy of an organization from taking place, analogous to the function of firewalls in building construction. Often a firewall is also referred to as a packet filter. The basic task of a firewall is to control traffic between different zones of trust and/or administrative authorities. Typical zones of trust include the Internet (a zone with no trust) and an internal network (a zone with high trust). The ultimate goal is to provide controlled connectivity between zones of different trust levels through the enforcement of a security policy and a connectivity model based on the least privilege principle.

Proper configuration of firewalls demands skill from the administrator. It requires considerable understanding of network protocols and of computer security. Small mistakes can lead to a firewall configuration worthless as a security tool and, in extreme situations, fake security where no security at all is left.

FiTP commands

FiTP commands are a set of commands that comprise the control information flowing from the user_CH to the auth-CH host.

HMAC

The Hash-based Message Authentication Code is a cryptographic hash function using a secret key to calculate a hash representing a unique identifier for any specific message. Any other message will generate a different HMAC. It is not possible to calculate a second string with the same HMAC without using a brute force attack, i.e. testing every possible string. Because of the complexity of this task it is not viable to start a brute force attack. For the FiTP protocol the SHA-256 Base64 cryptographic hash function is used. If the length of a Base64-encoded digest isn't a multiple of 4, simply "=" characters are added. This generates a 4 byte multiples hash.

PI

The protocol interpreter is the software which implements the FiTP protocol in user_PI and auth_PI. The user and auth server sides of the protocol have distinct roles implemented in a user_PI and a server-PI.

Reply

A reply is an acknowledgment (positive or negative) sent from the server to the user process via the control connection in response to FiTP commands. The general form of a reply is a completion code (including error codes) followed by a comma separated text string. The codes are for use by programs and the text is usually intended for human users.

sDTH

The data transfer host establishes the data connection with the "listening" data port. It sets up parameters for transfer and storage, and transfers data on command from its PI. The sDTH can be placed in a “passive" state to listen for, rather than initiate a connection on the data port.

User

A person or a process on behalf of a person wishing to get dynamically opened a connection on a firewall for communication with an application server process.

User_CH

The user control host (user_CH) sends requests for port openings to Port L at the auth-CH. After positive processing of its requests, the user_CH starts the application processes for which the port openings are needed at the firewall.

User_PI

The user protocol interpreter initiates the control connection from its port U to the auth-CH server, initiates FiTP commands, and requests access grants for its application data streams.

2.4 The FiTP model

The main purpose of the FiTP protocol is to inform firewalls located on the communication path of a desired data connection about the request for dynamic access rule configuration for this data traffic. To allow reading those requests on the fly, it is necessary that all requests are sent in clear text. On the other way sending commands in clear text allows easy changing of this commands by a man-in-the-middle. We will describe below how the first one can be assured and the second one be prohibited.

With the above definitions in mind, we can describe the FiTP model as follows:

The user-protocol interpreter (user_PI) initiates the control connection, which follows the telnet protocol, to a predefined special port for FiTP. After this initiation phase the user generates standard FiTP commands and transmits these to the server process via the control connection. Standard replies are sent from the server-PI to the user_PI over the control connection in response to the commands.

Since the FiTP control connection uses cleartext messages, any message sent can be read by intermediate firewalls.

Generally, sending clear text messages over an unsecure connection requires some additional procedures to secure the communication.

First of all a shared session key has to be exchanged. This is done within the initialization phase. It is assumed that the user_PI knows the public key (of a public/private key pair) of the auth_PI. This could have been transmitted by other methods to the user, e.g. via https access to the server side or encrypted emails. How the user_PI gets the public key of th auth_PI is out of the scope of this document.

The user_PI generates an additional random key for this specific session, the so named session key. Then he encrypts this session key with the public key of the auth_PI and sends this as first message to the auth_PI. Additionally the user_PI appends a HMAC of the message at the end of it. This HMAC has to be generated using the session key generated before. The auth_PI decrypts the encrypted session key using his private key, and gets back from the algorithm the shared key generated by the user_PI. He uses this session key to generate the HMAC of the received message. Comparing both HMACs, he can be sure that this message has not been changed. Then he acknowledges the reception of the shared key by sending the acknowledgment also with a HMAC generated by this shared key.

If the user_PI can create the same HMAC with his key, he knows that he is talking to the server. Only the server has the private key belonging to the public key used for encrypting the session key. Only the server knows this secret. If not, something has gone wrong and the session MUST be aborted.

Now both sides have a shared key available with which they further on can encrypt authentication and authorization messages and generate HMACs.

Since the Hashed Message Authentication Code (HMAC) routines are being used even for plaintext transfers, it is guaranteed that no man-in-the-middle can modify messages sent anymore.

To protect against replay attacks, the server appends with his first reply a message sequence ID (MSID) which is a random integer out of the range (0 .. 2**30). Both, the unique session key of client and together with the unique random number make the following message exchanges unique. With every message sent this MSID will be increased by one, so that any message is unique. So a replay attack of the whole session cannot work, since a new session will generated a different key on client side and different random message sequence ID on server side. Furthermore, since every message has its own unique MSID which is increased by 1 every time, client and server can be sure that no messages of any kind can be inserted by a third party, though this would not either help any attacker, since he does not know the shared key and therefore cannot generate valid messages, but also he cannot use a previously sent message again, because this would be out of sequence.

The FiTP commands are grouped into “key exchange”, “authentication”, and “authorization” commands as well as “grant access” commands.

After having exchanged a shared key between user_PI and auth_PI using “key exchange” commands, authentication of both communication partners has to be done. Then the authorization of the user to perform grant access commands, i.e. if the user is allowed to request dynamic connections in general, has to be checked.

The next step is the exchange of grant access requests and responses.

“Grant access” commands allow requests for opening ports to be sent to the server side and being checked there for granting access. For every request by the user_PI the server checks if he can grant this access to this user. So it is possible to provide specific users more privileges than others.

The FiTP protocol is implemented in the same way as the old FTP protocol has been implemented. An application does not know, if a firewall is FTP aware, i.e. if it is able to allow data connections which have been signaled via the FTP control connection. Same applies to the FiTP protocol.

This means, a user has to check if he is allowed to use the FiTP protocol. This is normally true if an internal server provides an FiTP service to the outside world. The system administrator providing an FiTP service has to check locally if his side allows the FiTP protocol. There are two possibilities:

a.) The local firewall supports FiTP, i.e. the firewall software can handle the protocol and the firewall administrators allow that external clients connect to the FiTP server internally.

b.) The local firewall does not support the protocol, but the firewall can be reconfigured by an external process, i.e. the FiTP server. In this situation a special FiTP server implementation has to be used. The communication protocol between the FiTP server and the local firewall is out of the scope of this document.

Dependent on the scenario a.) or b.) above, the server acknowledges the request and if needed, configures the firewall accordingly. (As described this is only required, if the firewall has not already read and interpreted the user request, i.e. this is only required if no FiTP inspection module for this firewall is available.) When the server has acknowledged an access rule, the user can start its data connection.

Several grant access requests can be issued by the user_PI in parallel to allow the signaling of multiple data connections with one FiTP control connection only.

The end of a data connection has to be signaled also, so that the access rules can be deleted from the firewall configuration again. Nevertheless, because of security reasons all access lists granted for an FiTP control connection MUST be deleted when the FiTP control connection has ended.

This implies several requirements on client, firewall and server side.

- The client MUST record any grant request he has sent to the server, to be able to close those requests accordingly.
- The firewall MUST control the connection and act on any grant request accepted by the server side. Furthermore, the firewall MUST close, i.e deny any session, when client and/or server close the session or do not “hold up” the control session via keep alive signaling commands. For this reason, the firewall SHOULD log status information for every FiTP session in progress.
- The server MUST check authentication and authorization of the user and correlate requests with his “user access rights” table. This will allow differentiating between rights of individual users. A system administrator may e.g. be allowed to open a bigger port range than other users or a user may be allowed to open more ports to a host with a certificate than with userid and password only. This server access rights checking is server dependent and not covered within this specification. Both sides of the communication, i.e. client and server have to setup timers, to become aware of outages of the other side. This is needed,

because one side cannot be sure if the other side does not respond because it is very busy or e.g. crashed. If no timers would be used connections could stay open for very long times and would therefore constitute a security risk.

2.5 The FiTP client software library

FiTP uses the Telnet protocol on the control connection. This can be achieved in two ways: first, the user_PI or the server-PI may implement the rules of the Telnet Protocol directly in their own procedures; or, secondly, the user_PI or the server-PI may make use of the existing Telnet module in the system.

Efficiency and independence argue for the first approach. Ease of implementation, sharing code, and modular programming argue for the second approach. In practice, FiTP relies on very little of the Telnet Protocol, so the first approach does not necessarily involve a large amount of code.

Nevertheless it seems preferable to implement the user_PI as a software library, where authentication, key_exchange, and authorization as well as grant access requests are provided as subroutines, so that any grid application can use these subroutines within their program codes. A sample command sequence is provided in chapter “A typical FiTP scenario”. A related program sequence in PERL using an FiTP software library is shown in chapter “A sample FiTP program” below.

2.6 Establishing data connections

The mechanics of transferring data over the granted communication paths is out of scope to this document. Any kind of application can be used, which uses IP, TCP, UDP, or IPSEC transfer protocols. Furthermore the protocol includes IPv4 as well as IPv6 grant requests commands.

Having granted access the communication path can be used. After the data transfer the user MUST signal the end of the data communication to the auth_CH via the control connection so that the firewalls located on the path are aware of deleting access grants. Nevertheless the firewalls MUST close the data path grants just after the control connection has been ended.

2.7 Error recovery and restart

There is no provision for detecting bits lost or scrambled in data transfers; this level of error control has to be handled by TCP or the application itself.

However, a restart procedure is provided to protect users from gross system failures (including failures of a host, an FiTP-process, or the underlying network).

The restart procedure is defined only for the control connection here. Data connection errors have to be handled by the applications themselves.

Error recovery of the control connection may lead to loss of data path grants, which has to be handled by the applications also.

So error recovery is defined here only by means of restarting at predefined control status points.

Error recovery on client and server side is done by sending special restart control packets indicating the kind of error and point of restarting.

In case of any error on the control connection (e.g. time outs, illegal commands, illegal command sequences, commands with wrong HMACs) any side MUST cancel the TCP session, so that all data path grants get destroyed. In case of disaster recovery the user has to restart from the beginning.

3 FiTP command and reply syntax and description

The communication channel from the user_PI to the auth_PI is established as a TCP connection from the user to the standard server port defined below. The user protocol interpreter is responsible for sending FiTP commands and interpreting the replies received; the auth_PI interprets commands, checks authentication and authorization and sends corresponding replies.

The Firewall Traversal protocol commands and replies consist of a four digit number followed by a phase describing text string and command parameters and the HMAC, separated by commas. Every message is terminated by a <CRLF> character, i.e. “\015\012”. A HMAC is not used with path control messages 0xxx and 9xxx as well as 1000 messages, since a shared key has to be exchanged first. Any other message described below is appended by “{HMAC_of_Messages_generated_with_shared_key}” separated by a Colon.

Replies have the same syntax, but have additionally an acknowledgement or negative acknowledgement (ACKN, NACK) as third text string.

The Firewall Traversal protocol commands and replies are structured into “Control Session”, “Key Initiation”, “Key Exchange”, “Authentication”, “Authorization”, and “Grant Access” commands. The format of all commands is as follows:

Number code, code mnemonic, message type, message ID, text [,HMAC]<CRLF>

where code mnemonic can be

PCTL	Protocol Control commands
KEAA	Key Initialization for Authentication and Authorization
KEYE	Key Exchange commands
AUTH	Authentication commands
ADAT	Authorization data commands
GACR	Grant Access Commands and Replies

and message type is

DATA	Information data is sent
ACKN	the previous DATA is acknowleged
NACK	the previous DATA cannot be acknowledged

The message ID is an integer between 1 and 2**30.

The first message, i.e. client to server must be “0000000001”.

Within the response to this initial message the server uses a random number between 1 and 2**30.

Further messages always are computed by adding 1 to the previous message. This guarantees that every message is unique. A replay of a message sequence cannot be used, since the client generates a random key for every new control connection and the server a random message ID for the first response to the client initiation message.

So the Firewall Traversal protocol commands and replies consist of a four digit number followed by a phase describing code mnemonic, a message type, message ID and a text string including command parameters, and the HMAC, separated by commas. Every message is terminated by a <CRLF> character, i.e. “\015\012”.

A HMAC is not used with path control messages 0xxx and 9xxx as well as 1000 messages, since a shared key has to be exchanged first. Any other message described below is appended by “{HMAC_of_Messages_generated_with_shared_key}” separated by a comma.

Replies have the same syntax, but have additionally an acknowledgement or negative acknowledgement (ACKN, NACK) as third text string.

In the rest of chapter 3 the HMAC and the ending <CRLF> are not included in the description, but have always to be added in the protocol messages.

3.1 FiTP commands and replies

3.1.1 Control Session commands and replies

All control sessions commands are structured as follows:

[**4_digit_message_code**],**PCTL,DATA**,[messageID],[textstring]

Responses have the following structure:

[**4_digit_message_code**], **PCTL**,[**ACKN|NACK**],[messageID],[textstring]

Valid commands are:

0000,**PCTL,DATA,**[messageID]

This command is send as the first command after the session has been established, i.e. TCP-Hand-Shake done. It establishes the “FiTP”-Session. Since no messageID has been provided by the server yet, the MSID “0000000001” will be used and ignored by the server.

0000,**PCTL,ACKN,**[messageID]

This is the positive answer of the authentication server protocol interpreter auth_PI to the 0000 control session initiation command of the user_PI. Auth_PI is ready for session. Auth_PI provides with this reply message the initial messageID. This will be increased by 1 with every new messaged sent by client and server, making every message unique.

0001,**PCTL,DATA,**[messageID],**NOOP**

This command may be sent by user_PI process at any time to check if the server is still available. It MUST be used to hold up the control connection when a timeout would appear at the remote side otherwise or SHOULD be used to check if the servers is still available.

0001,**PCTL,ACKN,**[messageID],**NOOP**

This is the acknowledgement to a previous sent 0001 message by the user_PI client process.

0002,**PCTL,DATA,**[messageID],**NOOP**

This command MUST be sent by auth_PI process to check if the client is still available. It MUST be used to hold up the control connection when a timeout would appear at the remote side otherwise.

0002,**PCTL,ACKN,**[messageID],**NOOP**

This is the acknowledgement to a previous sent 0002 message by the auth_PI server process.

0005,**PCTL,NACK,**[messageID],**Awaiting_initiation_cmd_with_code_0000**

This is a negative acknowledgement, since the client has not send a 0000 initiation command.

8000,**PCTL,DATA,**[messageID]

This command MUST be sent by user_PI process for finalizing control connection. This command will be sent by the user_PI after all data connections have ended and the control connection is not needed anymore.

This is the normal way for the user_PI to signal to the auth_PI that the user intends to finalize the control connection.

8000,**PCTL,ACKN,**[messageID]

Positive response of the auth_PI to the finalizing request 8000 by user_PI.

9000,**PCTLPCTL,DATA,**[messageID]

> If user_PI or auth_PI have diagnosed a severe problem they MUST kill the control connection immediately. The PI SHOULD not wait for any response. This message MUST be sent from both sides at any time when identifying any abnormalities.

In the rest of this chapter the text "***stop the session***" is defined as shut down the control session immediately by sending a 9000 message to the communication partner and not waiting for a reply anymore ((i.e. the whole TCP session will be closed). This behavior will signal to the firewall, that any grants requested by this control connection MUST be canceled. If the user needs further access grants, he has to start a new TCP control session from beginning.

0001, 0002 and 9000 messages MAY be sent at any time during the FiTP session. Since these commands are relevant for connection control and ongoing test of availability of the communication partner, each site should be prepared to answers those messages immediately.

3.1.2 Initiation commands and replies for key exchange

All key initiation commands are structured as follows:

[**4_digit_message_code**],**KEAA,DATA**,[messageID],[textstring]

Responses have the following structure:

[**4_digit_message_code**],**KEAA**,[**ACKN|NACK**],[messageID],[textstring]

Valid commands are:

1000,**KEAA,DATA,**[messageID]

> Command send by user_PI for starting session key exchange, authentication, and authorization.

1000,KEAA,ACKN,[messageID]

> This is a positive reply to a 1000 user_PI request. After this response key exchange, authentication, and authorization can start.

1001,KEAA,NACK,[messageID],**Form_error._Awaiting_KEAA_cmd**

The last command sent by user_PI has been illegally formed or is out of sequence. Key exchange and authentication not started yet. The server MUST “***stop the session***” after this response.

1009,**KEAA,NACK,**[messageID]

Auth_PI is not ready for key exchange, authentication, and authorization. The server MUST “***stop the session***” after this response.

3.1.3 Key-negotiation commands and replies

All key negotiation commands are structured as follows:

[**4_digit_message_code**],**KEYE,DATA**,[messageID],[textstring]

Responses have the following structure:

[**4_digit_message_code**],**KEYE**,[**ACKN**|**NACK**],[messageID],[textstring]

Valid commands are:

2000,**KEYE**,**DATA,**[messageID],{key_exchange_data}

2000 messages will be sent by the user_PI for exchanging a session key with auth_PI. The messages sent are dependent on the key exchange method agreed on. The server auth_PI responses with 2000 messages also. User_PI sends already a HMAC. Auth_PI must first decrypt key_exchange data before he can use the now available shared key for HMAC check.

2000,KEYE,ACKN,[messageID],{key_exchange_data}

Positive response to a sequence of one or more user_PI key exchange commands. The key_exchange_data here is the answer to the user_PI. Also this message may be send several times until all data has been transmitted. A 2002,KEYE,ACKN,Done message ends this sequence.

2001,KEYE,NACK,[messageID],**Form_error._ Awaiting_KEYE_cmd**

Negative response to a 2000 request or user_PI message sent is out of sequence. This command may or may not include a HMAC dependent on available shared key. If a key is available auth_PI MUST include the HMAC and user_PI MUST check its correctness. The server MUST "***stop the session***" after this response.

2002,KEYE,DATA,[messageID],**DONE-Shared_Key_Available**

User_PI informs the auth_PI, that key exchange data is complete. So this command ends a sequence of one or more 2000,KEYE user_PI messages. User_PI acknowledges to the auth_PI that a shared key has been negotiated.

2002,KEYE,ACKN,[messageID],**DONE-Shared_Key_Available**

This response is similar to the user_PI 2002 message above. Auth_PI acknowledges to the user_PI that he also agrees on having negotiated a shared key.

2006,KEYE,NACK,[messageID],**Awaiting_KEYE_end_cmd_2002**

This is a negative acknowledgement, since the client has not send a 2002,KEYE,DATA,[messageID],DONE-Shared_Key_Available command. The server MUST "***stop the session***" after this response.

2008,KEYE,NACK,[messageID]

If no shared key has been assigned, but the user_PI tries to start authentication or authorization the Auth_PI will sent an 2008 response to indicate "No shared session key available". The server MUST "***stop the session***" after this response.

2009,KEYE,NACK,[messageID],**No_Shared_Key_Available**

User_PI acknowledges to the auth_PI that no valid shared key has been negotiated. This message can be sent in case of any abnormal situation which may have come up negotiating a shared key. The same message can be sent by the auth_PI to the user_PI in case of no valid session key available. The server MUST "***stop the session***" after this response.

3.1.4 Authentication commands and replies

All authentication commands are structured as follows:

[**4_digit_message_code**],**AUTH,DATA**,[messageID],[textstring]

Responses have the following structure:

[**4_digit_message_code**],**AUTH**,[**ACKN|NACK**],[messageID],[textstring]

Valid commands are:

3000,**AUTH,DATA,**[messageID],**Meth,**{authentication_method}

This command is send by the user_PI to ask for an authentication with method {authentication_meth}. Allowed methods are:

Preshared_keys

Public_key_exchange

Certificate

UID-PWD

3000,AUTH,ACKN,[messageID],**Meth,**{authentication_method}

This command is a positive response to an authentication method request. After this response 301x user_PI commands can be sent and those will be answered by 301x responses.

3001,AUTH,NACK,[messageID],**Form_error._ Awaiting_AUTH_cmd**

This is a negative response. Server awaited a 3000 request. User_PI message sent is out of sequence or illegally formed. The server MUST "***stop the session***" after this response.

3008,AUTH,NACK,[messageID],**Method_not_supported**

This is a negative response to a 3000 request because requested authentication method is not supported. User_PI MAY try another method.

3009,**AUTH,NACK,**[messageID]

Auth_PI is not willing to do authentication currently. The server MUST "***stop the session***" after this response.

3010,**AUTH,DATA,**[messageID],{authentication_data}

This command is used by the user_PI to provide authentication data. Several authentication commands can be sent in sequence to provide relevant information. The text strings {authentication_data} are dependent on the authentication method agreed on. Every "3010, AUTH,DATA={…}" command has to be acknowledged by a "3010,AUTH,ACK,DATA={….}" response.

3010,AUTH,ACKN,[messageID]**,DATA,**{authentication_data}

Positive response to a user_PI authentication command with potential answers to the user_PI.

3011,AUTH,NACK,[messageID],**Form_error._Awaiting_AUTH_cmd**

Negative response to a 3010 request, e.g. user_PI messages sent are out of sequence. The server MUST "***stop the session***" after this response.

3012,AUTH,DATA,[messageID],**DONE-Authentication_Complete**

User_PI informs auth_PI, that authentication data is complete. So this command ends a sequence of 3010 user_PI messages. Since the amount of authentication data is method dependent this allows a sequence of 30xx messages to be sent and terminated.

3012,AUTH,ACKN,[messageID],**DONE-Authentication_Complete**

Auth_PI informs user_PI that authentication data is complete. So this command ends a sequence of 3010 auth_PI acknowledgments. Since the amount of authentication data is method dependent this allows a sequence of 30xx messages to be sent and terminated.

3017,AUTH,NACK,[messageID],**No_Authentication_Method_agreed_on**

User_PI request received for an Authentication, Authorization or Grant Access, but yet no Authentication method has been agreed on. The server MUST "***stop the session***" after this response.

3018,AUTH,NACK,[messageID]

If authentication has not been finished, but the user_PI tries to start authorization the Auth_PI will sent a 3018 response to indicate that the authentication has not been finished yet. The server MUST "***stop the session***" after this response.

3019,**AUTH,NACK,**[messageID]

This is the reply an auth_PI sends back to a user_PI when authentication has failed. User_PI MAY start further authentication requests again with a 3000 message or stop session.

3.1.5 Authorization commands and replies

All authorization commands are structured as follows:

[**4_digit_message_code**],**ADAT,DATA**,[messageID],[textstring]

Responses have the following structure:

[**4_digit_message_code**],**ADAT**,[**ACKN|NACK**],[messageID],[textstring]

Valid commands are:

4000, **ADAT,DATA,**[messageID],**Meth,***{authorization_method}*

With this message the user_PI starts the authorization process. It requests for authorization by method {authorization_method}.

4000, **ADAT,ACKN,**[messageID],**Meth,***{authorization_method}*

This is the response of the auth_PI to a 4000 message request in case of supporting method {authorization_method}.

4001,**ADAT,NACK,**[messageID],**Form_error._Awaiting_ADAT_cmd**

If a 4000 request has been received in an unexpected manner, then a 4001 response will be sent back to the other PI. Also packets not in sequence are answered by this reply. The server MUST “***stop the session***” after this response.

4008,ADAT,NACK,[messageID],**Method_not_supported**

This is a negative response to a 4000 request, because requested authorization Method is not supported. User_PI has to start with a 4000 message again for trying other methods or stop session.

4009,ADAT,NACK,[messageID]

This is the reply an auth_PI sends back to a user_PI when authorization is not possible currently. The server MUST “***stop the session***” after this response.

4010,**ADAT**,**DATA,**[messageID],{authorization_data}

4010 messages will be sent by the user_PI for exchanging authorization data to the auth_PI. The messages sent are dependent on the authorization method. The server auth_PI responses with 4010 messages also.

4010,ADAT,ACKN,[messageID],**DATA,**{authorization_data}

This is a positive response to a user_PI authorization exchange data message with potential answers to the user_PI.

4011,**ADAT**,**NACK**,[messageID],**Form_error._Awaiting_ADAT_cmd**

If a 4010 request has been received in an unexpected manner, then a 4011 response will be sent back from auth_PI to user_PI. Also packets not in sequence are answered by this reply. The server MUST "***stop the session***" after this response.

4012,ADAT,DATA,[messageID],**DONE-Authorization_Complete**

User_PI informs auth_PI about the completion of authorization command requests. So this command ends a sequence of 4010 user_PI messages. Since the amount of authorization data is method dependent this allows a sequence of 4010 messages to be terminated.

4012,ADAT,ACKN,[messageID],**DONE-Authorization_Complete**

Auth_PI informs user_PI that authorization data is complete from his point of view also.

4018,ADAT,NACK,[messageID]

If authorization has not been finished, but the user_PI tries to send Grant access commands already the Auth_PI will sent an 4018 response to indicate "No authorization finished". The server MUST "***stop the session***" after this response.

4019,ADAT,NACK,[messageID]

This is the reply an auth_PI sends back to a user_PI when authorization has failed. User_PI MAY start a new authorization sequence again with another 4000 authorization message or stop session.

It should be mentioned here that a auth_PI may sent back a 4012,ADAT,ACKN command already after a successful 20xx and 30xx sequence. E.g. the auth_PI may decide that after this sequence, authentication has been done and a shared key was exchanged via the authentication process already. The valid authentication may also be sufficient for authorization, e.g. a certificate was presented and this certificate approves for further actions.

3.1.6 Grant access commands and replies

All grant access request and response commands are structured as follows:

[**4_digit_message_code**],**GACR,DATA**,[messageID],[textstring]

Responses have the following structure:

[**4_digit_message_code**],**GACR**,[**ACKN|NACK**],[messageID],[textstring]

Valid commands are:

5000,**GACR,DATA,**[messageID],**Allow,***n,prot,h1,p1,p2,h2,p3,p4*

A user_PI requests access for an application which wants to communicate between ip addresses h1 and h2. The application uses the protocol *prot* and ports on the client side between p1 and p2 and on server side between p3 and p4. h1 and h2 may be single ip addresses or subnetworks. h1 and h2 have to be specified in modified CIDR notation with all quadruples using three digits and the CIDR block prefix using 2 digits (e.g. 192.168.016.000/22, which means all addresses within the class C networks 192.168.16.0, 192.168.17.0, 192.168.18.0 and 192.168.19.0). This allows easy scanning of the positional parameters by firewall hardware. "n" is a decimal number representing the n-th request, also being sent as 5 digit number. Furthermore ports have to be specified as 5 digit numbers (having 65536 a special meaning as "*", i.e. all ports).

By specifying this decimal number (request number) this allows the auth_PI to send responses related to each individual grant request. All communication partners should keep track of all open connections, the user_PI as well as the auth_PI. Also the firewall has to keep status info on all connections active. The request number is needed to signal all partners which request is acknowledged or should be closed. So it is possible to ask several times for the same port openings, e.g. when using port ranges, not knowing which explicit ports will be used by the data transfer applications later on. Think about multiple gridftp transfers in parallel. You always close the request number for the specific data connection, independent of the fact that there may be others using similar or equal port ranges. Think about several work flows in parallel. The firewall has to manage all those similar or equal port requests and MUST not close the port/port range before all requests have been terminated (closed).

5000,**GACR,ACKN,**[messageID],**Allow,***n*,*prot*,*h1,p1,p2,h2,p3,p4*

Positive acknowledgement to the grant request n. For h1 and h2 specification see 5*000 user_PI command above.*

5001,**GACR,NACK,**[messageID],**Form_error._Awaiting_GACR_cmds**

Auth_PI is awaiting a well formed 5000, 5020, 5600 or 5620 message, but got a malformed or quite different message.

5009,**GACR,NACK,**[messageID],**Deny,***n,prot,h1,p1,p2,h2,p3,p4*

The access grant has been denied by the auth_PI

5020,**GACR,DATA,**[messageID],**Close,***n,prot***,***h1,p1,p2,h2,p3,p4*

The access grant will not be needed anymore. It can be discarded. For h1 and h2 specification see 5*000 user_PI command above.*

5020,**GACR,ACKN,**[messageID],**Close,***n,prot***,***h1,p1,p2,h2,p3,p4*

This is a positive feedback from auth_PI. Access grants will be destroyed. For h1 and h2 specification see 5*000 user_PI command above.*

5021,**GACR,NACK,**[messageID],***Form error***

Auth_PI is waiting for a 5000, 5020, 5600 or 5620 message, but got a malformed or quite different message.

5023,**GACR,NACK,**[messageID],**NoConnection,***n*

User_PI has requested to delete an access grant, which is not known to the auth_PI. This may be related to a former timeout for this access grant or misconfiguration.

5600,**GACR,DATA,**[messageID],**Allow6,***n,prot,h6-1,p1,p2,h6-2,p3,p4*

IPv6 version of 5000 message. See “5000, GACR,Allow=...”. h6-1 and h6-2 are the IPv6 addresses of source and destination hosts/nets of the data connections for which the firewall should allow access.

5600,**GACR,ACKN,**[messageID],**Allow6,***n,prot***,***h6-1,p1,p2,h6-2,p3,p4*

IPv6 version of server response 5000 message

5601,**GACR,NACK,**[messageID],**Form_error._Awaiting_GACR_cmds**

IPv6 version of server negative response 5001 message

5609,**GACR,NACK,**[messageID],**Deny6,***n,prot,h6-1,p1,p2,h6-2,p3,p4*

IPv6 version of server deny 5009 message

5620,**GACR,DATA,**[messageID],**Close6,***n,prot***,***h6-1,p1,p2,h6-2,p3,p4*

IPv6 version of client close request 5020 message

5620,**GACR,ACKN,**[messageID],**Close6,***n,prot***,***h6-1,p1,p2,h6-2,p3,p4*

IPv6 version of positive server close response 5020 message

5621,**GACR,NACK,**[messageID],***Form_error***

IPv6 version of server response 5021 message

5623,**GACR,NACK,**[messageID],**NoConnection,***n*

IPv6 version of server response 5023 message

3.2 FiTP replies

3.2.1 Structure of FiTP reply codes

Replies to Firewall Traversal Protocol commands are devised to ensure the synchronization of requests to guarantee that the user process always knows the state of the server. Every command must generate at least one reply. In addition, some commands occur in sequential groups, such as key exchange commands or authentication and authorization commands. The replies show the existence of an intermediate state if all preceding commands have been successful. A failure at any point in the sequence necessitates the repetition of the entire sequence from the beginning.

The details of the command-reply sequence are made explicit in a state diagram below.

An FiTP reply consists of a four digit number (transmitted as four alphanumeric characters) followed by some text. The number is intended for use by automata to determine what state to enter next; except for some commands and replies the text is intended for the human user only. It is intended that the four digits contain enough encoded information that the user-process (the User_PI) will not need to examine the text and may either discard it or pass it on to the user, as appropriate. Nevertheless some special commands and replies contain data, where user_PI and/or auth_PI have to act on this data, e.g. key information, authentication and authorization data.

For security reasons user_PI and auth_PI MUST check that the appended HMAC to a message is valid, so that it is guaranteed that the messages and responses comes from the client and server respectively and are not changed by a man-in-the-middle.

In some cases the user_PI has to correlate responses to earlier sent requests (requesting several grant access rules with a request number included (see the “n” within the 5009, 5020 and 5021 responses).

The four digits of the reply each have a special significance. This is intended to allow a range of very simple to very sophisticated responses by the user-process. The first digit denotes which phase of the control connection has been reached. Digits two and three denote states within those control connection phases. Digit four denotes completion of requests or negative acknowledgements. There are eight values for the first digit of the reply code:

0xyz

The control session has just been started no further commands have been interchanged.

1xyz

Initiating session for Key exchange, authentication and authorization phases is in progress.

2xyz

Key exchange phase has started and is in progress.

3xyz

Authentication phase is in progress.

4xyz

Authorization phase is in progress.

5xyz

Grant access commands and replies are being sent.

8xyz

The control connection is in the closing process.

9xyz

user_PI or auth_PI have diagnosed a severe problem and will kill the control connection immediately.

Digit two is phase dependent. For phase 5, i.e. grant access commands, it differentiates between IPv4 and IPv6 commands having x=0 (for IPv4) or x=6 (for IPv6) respectively.

Digit three is also very phase dependent and corresponds to the different states the communication between client and server has entered.

Digit four within response messages is defined as follows:

xyz0

This indicates a positive response. The auth_PI acknowledges that it has received the request correctly and has changed its state accordingly. For grant access requests it acknowledges that the request has been accepted.

xyz1

The auth_PI has received a message from the user_PI which is illegally formatted, out of sequence or corrupted.

xyz2

The client sends data or the server sends positive responses in intermediate communication states.

xyz{n}

This is a negative response. The previous request cannot be acknowledged. {n} may very from 3 to 8.

xyz9

This is a negative response. The previous request cannot be granted. Dependent on the phase entered this may need starting this phase from the beginning or a grant access request has to be respecified (e.g. access to a special IP address is not allowed, but to another one an access could be granted.)

3.3 Numeric Order List of Command and Reply Codes

0000,**PCTL,DATA,**[messageID]<CRLF>
0000,**PCTL,ACKN,**[messageID]<CRLF>

0001,**PCTL,DATA,**[messageID],**NOOP,**{HMAC}<CRLF>
0001,**PCTL,ACKN,**[messageID],**NOOP**,{HMAC}<CRLF>
0002,**PCTL,DATA,**[messageID],**NOOP**,{HMAC}<CRLF>
0002,**PCTL,ACKN,**[messageID],**NOOP**,{HMAC}<CRLF>
0005,**PCTL,NACK,**[messageID],**Awaiting_initiation_cmd_with_code_0000**<CRLF>

1000,**KEAA,DATA,**[messageID]<CRLF>
1000,KEAA,ACKN,[messageID]<CRLF>
1001,KEAA,NACK,[messageID],**Form_error._Awaiting_KEAA_cmd**<CRLF>
1009,**KEAA,NACK,**[messageID]<CRLF>

2000,**KEYE**,**DATA,**[messageID],{key_exchange_data},{HMAC}<CRLF>
2000,KEYE,ACKN,[messageID],{key_exchange_data},{HMAC}<CRLF>
2001,KEYE,NACK,[messageID],**Form_error ._Awaiting_KEYE_cmd,**{HMAC}<CRLF>
2002,KEYE,DATA,[messageID],**DONE-Shared_Key_Available**,{HMAC}<CRLF>
2002,KEYE,ACKN,[messageID],**DONE-Shared_Key_Available**,{HMAC}<CRLF>
2006,KEYE,NACK,[messageID],**Awaiting_KEYE_end_cmd_2002**
2008,KEYE,NACK,[messageID],{HMAC}<CRLF>
2009,KEYE,NACK,[messageID],**No_Shared_Key_Available**,{HMAC}<CRLF>

3000,**AUTH,DATA,**[messageID],**Meth**,{authentication_method},{HMAC}<CRLF>
3000,AUTH,ACKN,[messageID],**Meth**,{authentication_method},{HMAC}<CRLF>
3001,AUTH,NACK,[messageID],**Form_error._Awaiting_AUTH_cmd**,{HMAC}<CRLF>
3008,AUTH,NACK,[messageID],**Method_not_supported**,{HMAC}<CRLF>
3009,**AUTH,NACK**,[messageID],{HMAC}<CRLF>
3010,**AUTH,DATA,**[messageID],{authentication_data},{HMAC}<CRLF>
3010,AUTH,ACKN,[messageID],**DATA,**{authentication_data},{HMAC}<CRLF>
3011,AUTH,NACK,[messageID],**Form_error._Awaiting_AUTH_cmd**,{HMAC}<CRLF>
3012,AUTH,DATA,[messageID],**DONE-Authentication_Complete**,{HMAC}<CRLF>
3012,AUTH,ACKN,[messageID],**DONE-Authentication_Complete**,{HMAC}<CRLF>

3017,AUTH,NACK,[messageID],**No_Authentication_Method_agreed_on**

3018,AUTH,NACK,[messageID],{HMAC}<CRLF>

3019,**AUTH,NACK**,[messageID],{HMAC}<CRLF>

4000,**ADAT,DATA,**[messageID],**Meth,***{authorization_method}*,{HMAC}<CRLF>

4000,**ADAT,ACKN,**[messageID],**Meth,***{authorization_method}*,{HMAC}<CRLF>

4001,**ADAT,NACK,**[messageID],**Form_error._Awaiting_ADAT_cmd**,{HMAC}<CRLF>

4008,ADAT,NACK,[messageID],**Method_not_supported**,{HMAC}<CRLF>

4009,ADAT,NACK,[messageID],{HMAC}<CRLF>

4010,**ADAT**,**DATA,**[messageID],{authorization_data},{HMAC}<CRLF>

4010,ADAT,ACKN,[messageID],**DATA,**{authorization_data},{HMAC}<CRLF>

4011,**ADAT**,**NACK**,[messageID],**Form_error._Awaiting_ADAT_cmd**,{HMAC}<CRLF>

4012,ADAT,DATA,[messageID],**DONE-Authorization_Complete,**{HMAC}<CRLF>

4012,ADAT,ACKN,[messageID],**DONE-Authorization_Complete**,{HMAC}<CRLF>

4018,ADAT,NACK,[messageID],{HMAC}<CRLF>

4019,ADAT,NACK,[messageID],{HMAC}<CRLF>

5000,**GACR,DATA,**[messageID],**Allow,***n,prot,h1,p1,p2,h2,p3,p4*,{HMAC}<CRLF>

5000,**GACR,ACKN,**[messageID],**Allow,***n*,*prot***,***h1,p1,p2,h2,p3,p4*,{HMAC}<CRLF>

5001,**GACR,NACK,**[messageID],**Form_error._Awaiting_GACR_cmds**,{HMAC}<CRLF>

5009,**GACR,NACK,**[messageID],**Deny,***n, prot,h1,p1,p2,h2,p3,p4,{*HMAC}<CRLF>

5020,**GACR,DATA,**[messageID],**Close,***n,prot***,***h1,p1,p2,h2,p3,p4*,{HMAC}<CRLF>

5020,**GACR,ACKN,** [messageID],**Close,***n,prot***,***h1,p1,p2,h2,p3,p4*,{HMAC}<CRLF>

5021,**GACR,NACK,**[messageID],***Form_error***,{HMAC}<CRLF>

5023,**GACR,NACK,**[messageID],**NoConnection**,n,{HMAC}<CRLF>

5600,**GACR,DATA,**[messageID],**Allow6,***n,prot,h6-1,p1,p2,h6-2,p3,p4*,{HMAC}<CRLF>

5600,**GACR,ACKN,**[messageID],**Allow6,***n,prot***,***h6-1,p1,p2,h6-2,p3,p4*,{HMAC}<CRLF>

5601,**GACR,NACK,**[messageID],**Form_error**,{HMAC}<CRLF>

5609,**GACR,NACK,**[messageID],**Deny6,**n,*prot***,***h6-1,p1,p2,h6-2,p3,p4*,{HMAC}<CRLF>

5620,**GACR,DATA,**[messageID],**Close6,***n,prot***,***h6-1,p1,p2,h6-2,p3,p4*,{HMAC}<CRLF>

5620,**GACR,ACKN,**[messageID],**Close6,***n,prot***,***h6-1,p1,p2,h6-2,p3,p4*,{HMAC}<CRLF>

5621,**GACR,NACK,**[messageID],***Form_error***,{HMAC}<CRLF>

5623,**GACR,NACK,**[messageID],**NoConnection**,n,{HMAC}<CRLF>

8000,**PCTL,DATA**,[messageID],{HMAC}<CRLF>

8000,**PCTL,ACKN**,[messageID],{HMAC}<CRLF>

9000,**PCTL,DATA,**[messageID]<CRLF>

4 Declarative specifications

4.1 Minimum implementation

The minimum implementation of the Firewall Traversal Protocol has to support authentication, key exchange and authorization method “shared_key”. Nevertheless it is recommended to support x.509 certificates (“certificate”).

4.2 Connections

The server protocol interpreter shall "listen" on Port L. The user or user protocol interpreter shall initiate the full-duplex control connection. Server and user processes should follow the conventions of the Telnet protocol. The control connection MUST be closed by the server at the user's request after all transfers and replies are completed. Furthermore the control connection MUST be closed by either client or server, if any unsuspicious behavior occurs.

The data connection will be initiated from host(s) sDTH to the server(s) running at dDTH (host or subnet). The data connection is out of scope of this document allowing any kind of TCP or “virtual” UDP, IP or IPSec communication.

If at any time either the user or server_PI observes that the connection is being closed by the other side, it MUST promptly read any remaining data queued on the connection and issue the close on its own side.

4.3 Commands

The commands are text strings transmitted over the control connections as described in the section on FiTP Commands.

The command syntax has been specified above already.

Upper and lower case alphabetic characters are to be treated identically. This also applies to any symbols representing parameter values such as the key exchange method text string (e.g. certificate).

Thus, any of the following may represent an authentication exchange command:

3030,auth,data,1432446578,meth,certificate

3030,Auth,DATA,1432446578,Meth,certificate

3030,Auth,daTA,1432446578,METh,CertiFicate

3030,AuTH,DaTA,1432446578,MetH,certifiCATE

3030,AUTH,DATA,1432446578,METH,CERTIFICATE

And any of the following may represent a grant access request command:**5000,GACR,DATA, 1432446578,Allow,**

00020,TCP*,123.045.067.089/32*,12345,12359,124.111.222.233/32,05000,05010

5000,gacR,data, 1432446578,allow,

00020,tcp*,123.045.067.089/32*,12345,12359,124.111.222.233/32,05000,05010

5000,GACR,datA, 1432446578,ALLOW,

00020,TCP*,123.045.067.089/32*,12345,12359,124.111.222.233/32,05000,05010

5000,Gacr,Data, 1432446578,Allow,

00020,Tcp*,123.045.067.089/32*,12345,12359,124.111.222.233/32,05000,05010

The argument field consists of a variable length character string.

As described above most of the commands are followed by a colon separated HMAC.

Since the HMAC is a 160 bit string it could contain ASCII control characters also. For this reason, the HMAC is interpreted as 40 times 4 bit strings, which will be transferred into their HEX representation as ASCII characters.

Therefore a HMAC of e.g.

"0100 1100 1011 1111 … 0101 0001 1010"

will be transferred as "4CBF…51A".

The same applies to the key_exchange_data, authentication_data, and authorization_data fields, which will be interpreted as binary info and therefore transferred in the same manner in their HEX representation as ASCII characters.

All commands are ending with the ASCII <CRLF>, "\015\012".

The syntax is specified in 8bit-ASCII below. All characters in the argument field are ASCII characters including any ASCII represented decimal integers. Square brackets denote an optional argument field. If the option is not taken, the appropriate default is implied.

4.4 FiTP command options

The following fields are allowed within FiTP commands:

Meth,<authentication_method> | < authorization_method>

DATA,<authentication_data> | <key_exchange_data> |

<authorization_data>

Allow,<five-digit-integer> <,> <*prot*> <,>

<*ip-cidr*> <,> <*port*> <,> <*port*> <,>

<*ip-cidr*> <,> <*port*> <,> <*port*>

Close, <five-digit-integer> <,> <*prot*> <,>

<*ip-cidr*> <,> <*port*> <,> <*port*> <,>

<ip-cidr> <,> <port> <,> <port>

Deny, <five-digit-integer>

Allow6, <five-digit-integer> <,> <*prot*> <,>

<*ipv6-cidr*> <,> <*port*> <,> <*port*> <,>

<*ipv6-cidr*> <,> <*port*> <,> <*port*>

Close6, <five-digit-integer> <,> <*prot*> <,>

<*ipv6-cidr*> <,> <*port*> <,> <*port*> <,>

<ipv6-cidr> <,> <port> <,> <port>

Deny6, <five-digit-integer>

4.5 FiTP command option syntax

The syntax of the above option fields (using BNF notation where applicable) is:

<messageID> ::= < “0000000001” | “0000000002” | … | “1073741824” >

i.e. any integer 1 through 2**30 specified in 10 digits with leading zeros

<authentication_method> ::= <”CERTIFICATE”> | <”PRESHARED_KEY”> |

<”PUBLIC_KEY_EXCHANGE”> | <”UID-PWD”>

<authorization_method> ::= <”CERTIFICATE”> | <”PRESHARED_KEY”> |
<”PUBLIC_KEY_EXCHANGE”> | <”UID-PWD”>

<authentication_data> ::= <uid_pwd_aa_data> | <preshkey_aa_data> |

<cert_aa_data> | <pubkey_aa_data>

<authorization_data> ::= <uid_pwd_aa_data> | <preshkey_aa_data> |

<cert_aa_data> | <pubkey_aa_data>

<uid_pwd_aa_data> ::= <length_uid><”:”><length_pwd><”:”><uid><pwd>

```
<preskey_aa_data> ::= <length_gid><":"><length_prekey><":"><gid><prekey>
<cert-aa_data> ::= <length_certificate><":"><certificate>
        <pubkey_aa_data> ::= <length_uid>:<uid>
        <length_uid> ::= <integer>
        <length_pwd> ::= <integer>
        <length_gid> ::= <integer>
        <length_prekey> ::= <integer>
        <length_certificate> ::= <integer>
<key_exchange_data> :: <string>
        <uid> ::= <string>
        <pwd> ::= <string>
        <gid> ::= <string>
        <prekey> ::= <string>
        <certificate> ::= <string>
<string> ::= <pr-char> | <pr-char><string>
        <char> ::= any of the 128 ASCII characters except <CR> and <LF>
<pr-char> ::= printable characters, any ASCII code 33 through 126
<five-digit-integer> ::= < "00001" | "00002" | … | "65536" >
               i.e. any integer 1 through 65536 specified in 5 digits
<prot> ::= <"IP"> | <"TCP"> | <"UDP"> | <"IPSEC">
<ip-cidr> ::= <ipaddr><"/"><netprefix>
<ipv6-cidr> ::= <ipv6addr><"/"><v6netprefix>
<ipaddr> ::= <3-digit-qual><.><3-digit-qual><.><3-digit-qual><.><3-digit-qual>
<3-digit-qual> ::= < "000" | "001" | ...  | "254" | "255" >
               i.e. any integer 0 through 255 specified in 3 digits
        <netprefix> ::= < "00" | "01" | … | "32" >
        <ipv6addr> ::= <hex4><":"><hex4><":"><hex4><":"><hex4><":">
                      <hex4><":"><hex4><":"><hex4><":"><hex4>
               i.e. any decimal integer 0 through 32 specified in two digits
        <hex4> ::= <hex><hex><hex><hex>
                                    i.e.  any 4 character hex string
        <hex> ::= <"0"> | <"1"> | <"2"> | <"3"> | <"4"> | <"5"> | <"6"> | <"7"> | <"8"> |
                        <"9"> | <"A"> | <"B"> | <"C"> | <"D"> | <"E"> | <"F">
                 i.e. any hex character 0 through F
        <v6netprefix> ::= < "000" | "001" | … | "128" >
```

<port> ::= <five-digit-integer>   i.e. any decimal integer 1 through 65536

4.6 Sequencing of commands and replies

The communication between user and server is intended to be an alternating dialogue. As such, the user issues a FiTP command and the server responds with a prompt primary reply. The user should wait for this initial primary success or failure response before sending further commands.

4.6.1 Spontaneous Replies

Sometimes "the system" spontaneously has to send a message to a user (usually all users), for example, "System going down in 15 minutes". In those cases User_PI and/or auth_PI may send a command

9000,**PCTL,DATA,**[messageID]<CRLF>

Receiving such a message, both sides MUST immediately close the control connection to be sure that the firewalls on the path are aware of such problems and are able to delete access rights granted for those sessions.

The following section lists all client command codes (in green) and their allowed respective responses by the server (ACKs in blue and/or NACKs in red).

4.6.2 Command Codes and related Replies

In this section, the command-reply sequence is presented. Each command is listed with its possible replies; command groups are listed together. This listing forms the basis for the state diagrams, which will be presented separately.

Connection Establishment

Connection initiation

→	**0000**	
←		**0000**

Connection control (client initiated) (Keep alive testing)

→	**0001**	
←		**0001**

Connection control (server initiated)

→	**0002**	
←		**0002**

Key Exchange, Authentication and Authorization initiation

→	**1000**	
←		**1000**
←		**1001,1009**

Key negotiation

Key negotiation request

→	**2000**	
←		**2000**
←		**2001,2002,2008,2009**

Key negotiation request complete

→	**2002**	
←		**2002**
←		**2001,2008,2009**

Authentication

Authentication data method request

→	3000	
←		3000
←		2006,3001,3008,3009

Authentication data exchange

→	3010	
←		3010
←		3011,3017,3019

Authentication data exchange complete

→	3012	
←		3012
←		3011,3017,3019

Authorization

Authorization Method request

→	4000	
←		4000
←		3018,4001,4008,4009

Exchanging of authentication data

→	4010	
←		4010
←		4011,,4019

Exchange of authentication data complete

→	4012	
←		4012
←		4011,4019

Grant Access Commands and Replies for IPv4

Grant Access (Allow)

→	5000	
←		5000
←		4018,5001,5009

Closing of Access Grants (Close)

→	5020	
←		5020
←		5021,5023

Grant Access Commands and Replies for IPv6

Grant Access (Allow6)

→	**5600**	
←		**5600**
←		**5601,5609**

Closing of Access Grants (Close6)

→	**5620**	
←		**5620**
←		**5621,5623**

Connection Closing

→	**8000**	
←		**8000**

Abnormal Closing of connections

→	**9000**	
←		**9000**

5 A state diagram for the FiTP client

The following diagram shows the different states a client can have. It also shows which server responses lead to which state transitions.

The diagram shows transitions for IPv4 requests and responses only, similar transitions take place in case of IPv6 grant requests and responses.

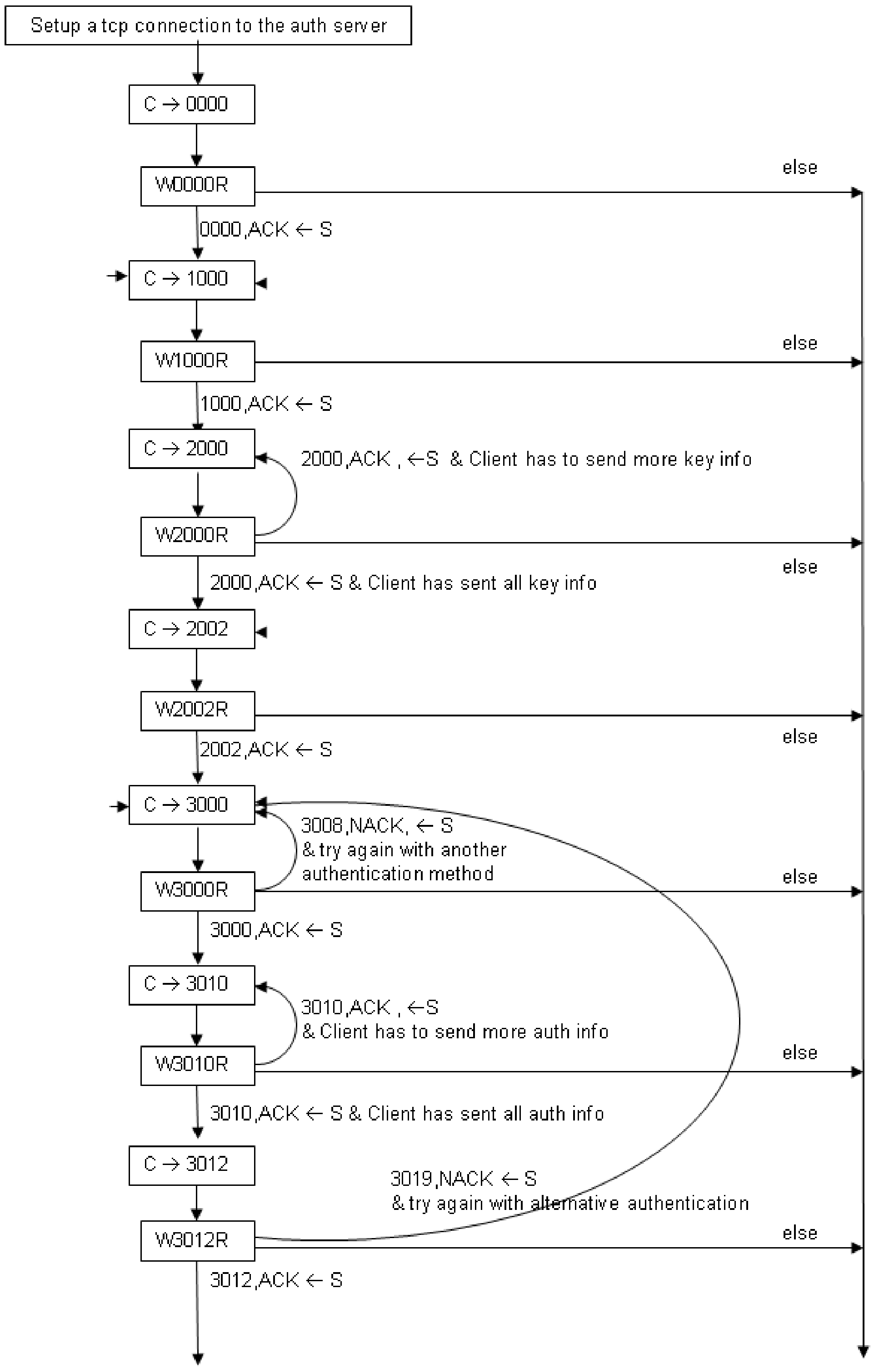
0001 client and 0002 server messages as well as 9000 client and server messages can be sent at any time. So in every state client and server have to act on those messages also. Those state transitions are not depicted to make the state diagram more readable.

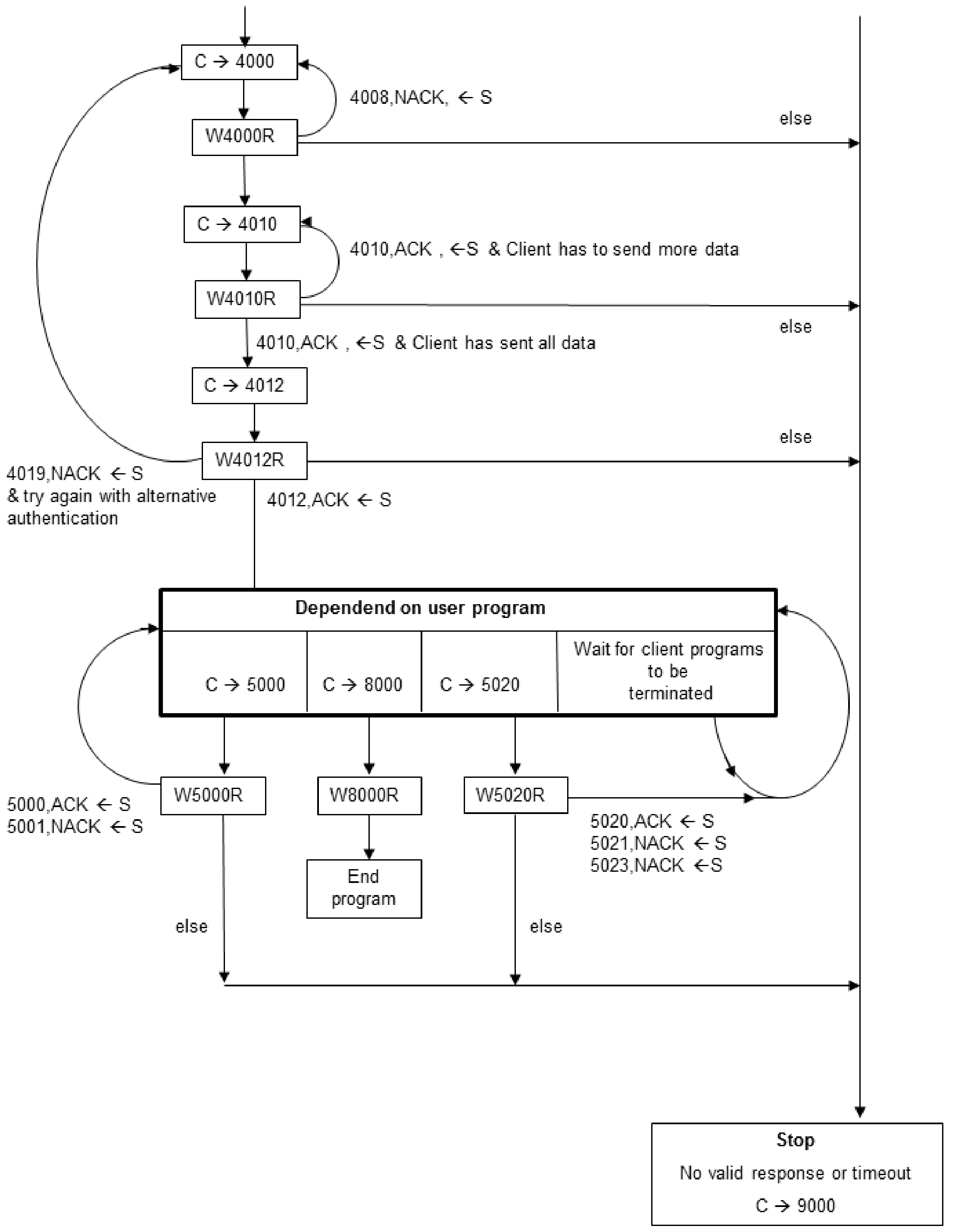
Every 30 seconds a message MUST be sent by the client and a response sent by the server to guarantee that both sides are still available. So Client and server MUST setup a new timer every time a message was sent. If no message exchange is needed, e.g. the client has requested all port openings and is waiting for the end of the data connections the client MUST generate 0001 messages to signal that it is still alive. Also the sever MUST send a 0002 message if the client has not sent any message within the timer interval. If the partner, client or server, did not respond within again 30 seconds with any response, then the communication MUST be closed. A server answers a 0001 message with a 0001 ACK or with the response to the message the client was originally waiting for. The client answers a 0002 message of a server with a 0002 ACK or with a new message, e.g. new grant request or grant delete request.

The following samples show which nomenclatures have been used within the diagram:

C ➔ 0000	Client sends a “0000” message
0000, ACK 🡐 S	Server has sent a “0000” acknowledgement
W2000R	Client is waiting for a “2000” acknowledgment

A further sequence diagram showing the server behavior will not be provided. This diagram would duplicate the client diagram in most parts. Of course the server has to be programmed to allow multiple sessions in parallel, has to have status information for every individual session and should handle those sessions independently. Putting all this parallelisms into one diagram would confuse the reader more than helping understand the details.


Setup a tcp connection to the auth server
C → 0000
W0000R
else
0000,ACK ← S
C → 1000
W1000R
else
1000,ACK ← S
C → 2000
2000,ACK , ←S & Client has to send more key info
W2000R
else
2000,ACK ← S & Client has sent all key info
C → 2002
W2002R
else
2002,ACK ← S
C → 3000
3008,NACK, ← S
& try again with another
authentication method
W3000R
else
3000,ACK ← S
C → 3010
3010,ACK , ←S
& Client has to send more auth info
W3010R
else
3010,ACK ← S & Client has sent all auth info
C → 3012
3019,NACK ← S
& try again with alternative authentication
W3012R
else
3012,ACK ← S


C → 4000
4008,NACK, ← S
W4000R
else
C → 4010
4010,ACK , ←S & Client has to send more data
W4010R
else
4010,ACK , ←S & Client has sent all data
C → 4012
else
W4012R
4019,NACK ← S
& try again with alternative authentication
4012,ACK ← S
Dependend on user program
C → 5000
C → 8000
C → 5020
Wait for client programs to be terminated
5000,ACK ← S
5001,NACK ← S
W5000R
W8000R
W5020R
5020,ACK ← S
5021,NACK ← S
5023,NACK ←S
End program
else
else
Stop
No valid response or timeout
C → 9000

6 A typical FiTP scenario

.

The following section is descriptive only and is not part of the protocol specification itself.

A User at host "U" wanting to use a communication path between host "S" port "sp" and host "D" port "dp" connects to the remote authentication and authorization server "AAS" at port "L": In general, the user transfers his requests to the server "AAS" via the control connection using quadruple [S,sp,D,dp]. Server "AAS" authenticates the user and checks for his authorization. All firewalls on this control path are able to read the requests sent by the client. After they have seen the positive acknowledgement sent back by the server they change configured access lists and dynamically open the required ports. The following message exchanges may be a typical scenario. Here '➔' represents commands from host U to host AAS, and '🡐' represents replies from host AAS to host U.

[FW]

[U] ← --- --- --- --- → [AAS]

### **Three way TCP handshake**

→ TCP,SYNC

← TCP,SYNC,ACK

→ TCP,ACK

### **Control session initiation**

→ TCP: 0000,PCTL,DATA

← TCP: 0000,PCTL,ACKN,{messageID}

### **Initiating KEY exchange, Authentication and Authorization Phase**

→ TCP: 1000,KEAA,DATA,{messageID}

← TCP: 1000,KeAA,ACKN,{messageID}

### **key exchange commands**

→ TCP: 2000,KEYE,DATA,{messageID},{key_exchange_data}

← TCP: 2000,KEYE,ACKN,{messageID},{key_exchange_data}

→ TCP: 2002,KEYE,DATA,{messageID},DONE-Shared_Key_Available

← TCP: 2002,KEYE,ACKN,{messageID},DONE-Shared_Key_Available

### **Asking for preshared_key authentication,**

### **i.e. "I have the shared key, so I am the one you assume."**

→ TCP: 3000,AUTH,DATA,{messageID},Meth,preshared_key

← TCP: 3000,AUTH,ACKN,{messageID},Meth,preshared_key

→ TCP: 3010,AUTH,DATA,{messageID},{authentication_data}

← TCP: 3010,AUTH,ACKN,{messageID},DATA,{authentication_data}

→ TCP: 3012,AUTH,DATA,{messageID},DONE-Authentication_Complete

← TCP: 3012,AUTH,ACKN,{messageID},DONE-Authentication_Complete

###The commands 3010 above may be issued several times

### until all relevant data has been transferred.

### Asking for preshared_key authentication,

### i.e. “I have the shared key, so I am the one you assume.”

→ TCP: 4000,ADAT,DATA,{messageID},Meth,preshared_key

← TCP: 4000,ADAT,ACKN,{messageID},Meth,preshared_key

→ TCP: 4010,ADAT,DATA,{messageID},{authorization_data}

← TCP: 4010,ADAT,ACKN,{messageID},DATA,{authorization_data}

→ TCP: 4012,ADAT,DATA,{messageID},DONE-Authorization_Complete

← TCP: 4012,ADAT,ACKN,{messageID},DONE-Authorization_Complete

### Again commands 4010 may be issued several times

### until all relevant data has been transferred.

### Now starting Grant Access Request Path

### Requesting access for GridFTP [GridFTP1|GridFTP2|GridFTP3]

### control connection

### from host 192.168.15.116 i.e. netprefix 32

### to host 172.100.14.123 (again netprefix 32)

### source port 30123, destination port 2811

→ TCP: 5000,GACR.DATA,{messageID},Allow,
    00001,TCP,192.168.015.116/32,30123,30123,172.100.014.123/20,02811,02811

← TCP: 5000,GACR.ACKN,{messageID},Allow,
    00001,TCP,192.168.015.116/32,30123,30123,172.100.014.123/20,02811,02811

###

### Requesting access rules for 50 data connections for above GridFTP session

### from host 192.168.15.116 i.e. netprefix 32

### to host 172.100.14.123 (again netprefix 32)

### lower source port 30124 and upper source port 30173,

### lower dest port 20001 and upper destination port 20050,

### i.e. we want to connect from ports out of the port range [30124, … , 20173]

### to ports within the range of [20001, … , 20050]

### The request setup below would allow any combination of port combination

### within the specified port ranges,

### e.g. from port 30150 to port 20002 and from port 30125 to port 20014 ###

→ TCP: 5000,GACR.DATA,{messageID},Allow,

00002,TCP,192.168.015.116/32,30124,30173,172.100.014.123/32,20001,20050

← TCP: 5000,GACR.ACKN,{messageID},Allow,

00002,TCP,192.168.015.116/32,30124,30173,172.100.014.123/32,20001,20050

### **Now starting a GridFTP process between 192.168.15.116 and 172.100.14.123**

### **This part is not shown here.**

### **After GridFTP has completed, we have to delete the access grants for Gridftp**

→ TCP: 5020,GACR.DATA,{messageID},Close,

00001,TCP,192.168.015.116/32,30123,30123,172.100.014.123/20,02811,02811

← TCP: 5020,GACR.ACKN,{messageID},Close,

00001,TCP,192.168.015.116/32,30123,30123,172.100.014.123/20,02811,02811

### **Delete grants for GridFTP data connections**

→ TCP: 5020,GACR.DATA,{messageID}Close,

00002,TCP,192.168.015.116/32,30124,30173,172.100.014.123/32,20001,20050

← TCP: 5020,GACR.ACKN,{messageID},Close,

00002,TCP,192.168.015.116/32,30124,30173,172.100.014.123/32,20001,20050

### **Requested access rules deleted. Now closing control connection**

→ TCP: 8000,PCTL,DATA,{messageID}

← TCP: 8000,PCTL,ACKN,{messageID}

### **Closing TCP session**

→ TCP,FIN

← TCP,FIN, ACK

→ TCP,ACK

### **All done.** ###

The following figure below describes the communication between participating communication partners.

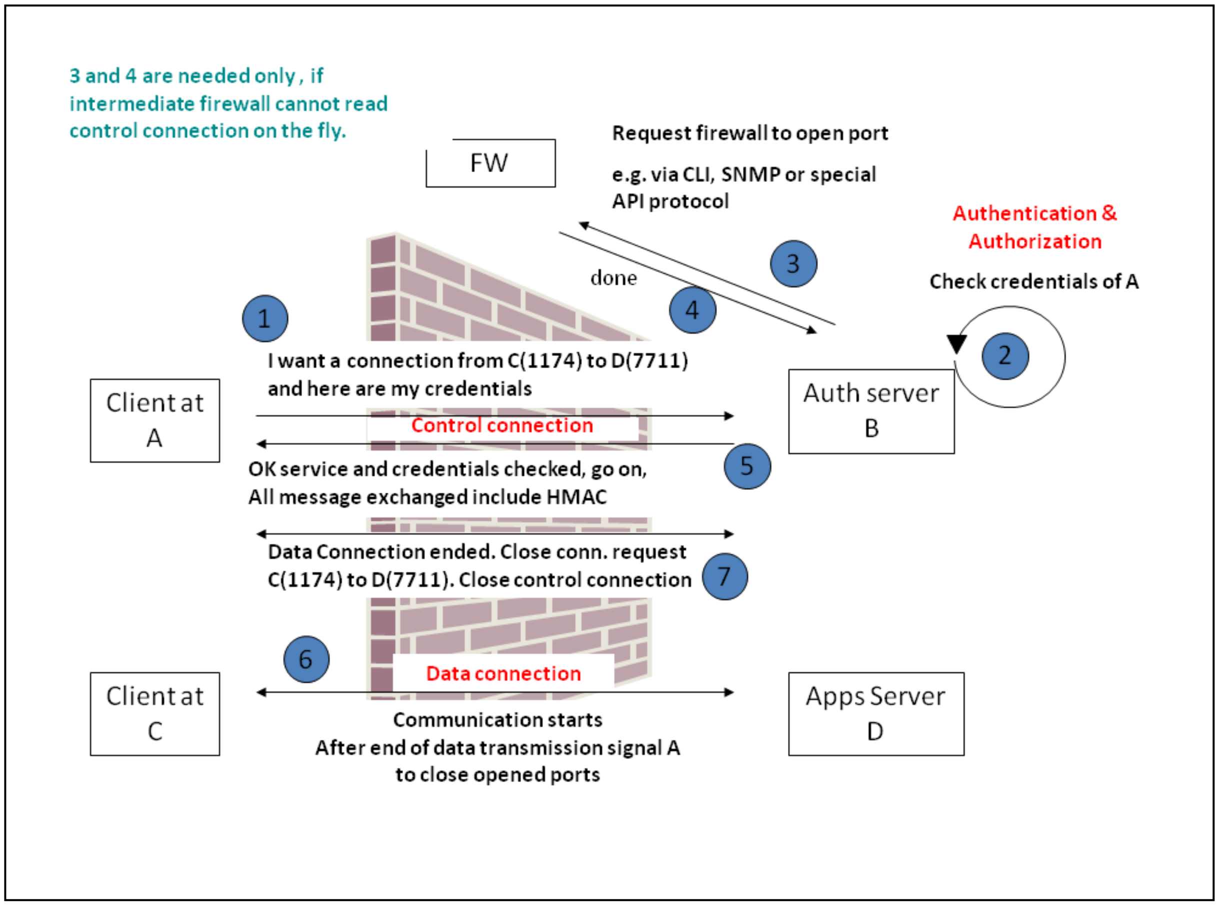
Here, “A” is the client setting up grant requests and “B” is the server to whom “A” sends his requests.

Application client “C” and application server “D” are not relevant for the FiTP protocol itself. The communication between these two partners is the one which client “A” tries to get permitted.

In this sense, client “A” and client “C” may not be located on the same external host. Similar server “B” and server “D” may not run on the same internal host. It has only to be guaranteed by the application user, that the communication between “C” and “D” does not start, before the privileges for access have been granted within the firewall.

Furthermore it should be guaranteed that "A" asks for deletion of these access rights not before communication between "C" and "D" has ended. For this "scheduling" the user is responsible himself.

1.) The client "A" asks "B" for access between "C" and "D"
2.) "B" checks if he can allow communication between "C" and "D"
3.) Signal firewall to open relevant ports
    a. If the firewall is not FiTP-Aware, some special application has to be developed, by which "B" can configure the firewall accordingly. This could be e.g. CLI based or using a firewall specific API.
    b. If the firewall is FiTP aware, nothing has to be done, since it can read the control connection between "A" and "B" and can handle accordingly.
4.) Firewall has configured access rules correctly.
5.) Auth server "B" signals back to "A" that the access rights have been granted.
6.) Data communication can take place
7.) After end of data communication in 6.), Client signals back to server "B" that access rights can be deleted again.


3 and 4 are needed only, if intermediate firewall cannot read control connection on the fly.
FW
Request firewall to open port
e.g. via CLI, SNMP or special API protocol
Authentication & Authorization
Check credentials of A
done
3
4
1
2
I want a connection from C(1174) to D(7711) and here are my credentials
Client at A
Auth server B
Control connection
5
OK service and credentials checked, go on, All message exchanged include HMAC
Data Connection ended. Close conn. request C(1174) to D(7711). Close control connection
7
6
Data connection
Client at C
Apps Server D
Communication starts
After end of data transmission signal A to close opened ports

7 A sample FiTP program in PERL

The following program code is a sample code sequence, which demonstrates a possible use of the FiTP protocol. It is not part of the specification itself, but helps to understand the intended usage of the protocol. Here FiTP is used to signal via the control channel that the client wants to start a GRIDFTP session which uses port 2811 for control channel and ports 20001 to 20050 for data communications.

```
#!/usr/bin/perl -w
use strict;
use Socket;
use FiTP;
############################## Global Variables ##########################
my $VERSION = "FiTP_v_4.0";
my $sessionkey;
my $ret;
my ($remote, $port, $iaddr, $paddr, $proto, $line);

##########################################################################
# THIS SAMPLE CODE IS PROVIDED "AS IS" AND ANY EXPRESS OR IMPLIED
# WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
# OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
# DISCLAIMED. IN NO EVENT SHALL THE CONTRIBUTORS BE LIABLE FOR ANY
# DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
# DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
# SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
# BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
# LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
# NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
# SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
##########################################################################

open(INFH,"<./mykeys");
my @conf=<INFH>;
close(INFH);

############################# Subroutines ###############################
sub call_applications_using_opened_port { print "Starting data connection\n"; return(0); }

############################# Main     ##################################
my $sref=&FiTP::connect("serverhostname","4711"); # TCP connection established ##

$sessionkey=&FiTP::sesskeygen;

if ( ! defined($sessionkey) ) { print "Main: Could not generate a session key\n"; exit 99; }

##########################################################################
######## We assume here, that the public key of the server has been stored in the file
########  fitpserverkeys.pub
```

```
########################################################################
my $srvpubfile = "fitpserverkeys.pub";
$ret=FiTP::initiate($sref,$srvpubfile);

if ( $ret != 0 ) { print "Main: Control path could not be established\n"; exit 99; }

########################################################################
########  We assume here, that we want to use a preshared key for authentication
######## and authorization, which client and server have agreed on before.
########################################################################
our $preshared_key = $conf[0];
$ret=FiTP::start_authentication($sref,$sessionkey,
                                "PRESHARED_KEY",$preshared_key);
if ( $ret != 0 ) { print "Main: Preshared Key Authentication failed\n"; exit 99; }

$ret=FiTP::start_authorization($sref,$sessionkey,
                                "PRESHARED_KEY",$preshared_key);
if ( $ret != 0 ) { print "Main: Preshared Key Authorization failed\n"; exit 99; }

#######################################################################
###   open control port for GridFTP
###   then open port range of 50 ports for GridFTP Data connections
#######################################################################

my @conn1 = ( "00001" , "TCP" , "192.168.015.116/32" , "30123" , "30123" ,
"172.100.014.123/32" , "02811" , "02811" );
my @conn2 = ( "00002" , "TCP" , "192.168.015.116/32" , "30124" , "30173" ,
"172.100.014.123/32" , "20001" , "20050" );

$ret=FiTP::dyna_conn_open4( $sref , $sessionkey , @conn1 );
$ret=FiTP::dyna_conn_open4( $sref , $sessionkey , @conn2 );

if ($ret == 0) { call_applications_using_opened_port(); }
else { print "Main: Dynamic opening not allowed or error when requesting"; exit 99; }

$ret=FiTP::dyna_conn_close4( $sref , $sessionkey , @conn1 );
$ret=FiTP::dyna_conn_close4( $sref , $sessionkey , @conn2 );

$ret=FiTP::closecontrl_session($sref,$sessionkey);
$ret=FiTP::disconnect();

exit
```

8 Connection establishment

The FiTP control connection is established via TCP between the user process port U and the server process port L. This protocol is assigned the service port 4711, that is L=4711. (Instead of 4711 an official port number, preferable within the “well known ports” range below 1024, should be used, agreed on and registered later).

9 Relation between control and data connections

As described before data connections are out of scope of this document. Nevertheless there is at least a logical relationship between control and data connections.

Though a data connection may have been allowed because of a prior control connection, no relationship between both will be stored. There are cases where a firewall cannot know to which access list entry a data connection belongs. There could be a static rule which allows this data connection already. Also there could be other control connections where access grants overlap with the new control connection grants. Furthermore it seems reasonable to close the control connection as soon as possible after the data transmission has been completed. Closing the control connection as early as possible prevents acceptance of further data connections which are not related to this application, but use a source IP, source port, destination IP, destination port quadruple, within the range access has been granted for.

Furthermore most firewalls check access lists for the first packet of a connection only. Any follow up packet is assumed as already checked concerning access lists.

Nevertheless it seems reasonable that established data connections MUST be kept active by the client until the data connection has been finished, because the user_PI does not know how the firewall handles connections.

10 Design alternatives

The FiTP model has been designed to support a variety of security scenarios.

The first scenario assumes that the firewalls, to be dynamically configured, have code integrated already, which is aware of the FiTP protocol, so that the firewalls are able to configure automatically the requested access rules. This scenario also includes asymmetric routing as long as both paths, source to destination and destination to source, are traversing all firewalls, which have to be dynamically configured. The scenario also assumes that the data connections specified by the quadruples source-ip, source-port, dest-ip and dest-port are also traversing these firewalls in both directions.

The second scenario assumes that the firewall is not aware of FiTP and the auth-server is able to request the firewall to dynamically reconfigure its access tables. This could be done via CLI commands or special firewall device dependent software interfaces for configuration. This would imply that every auth-server has access to those configuration routines.

A third approach could be to have a special firewall agent available, which can be contacted from every auth-server. This firewall agent would check if the auth-servers are allowed to request reconfiguration of the firewall and would then be the only one allowed to actively reconfigure the firewall.

Though scenario one is intended to be the primary way for the FiTP protocol usage, the other alternative scenarios will work also. Additionally those scenarios allow configuration of firewalls, which are not located on the control path, but on the data paths only.

Within these different scenarios security policies can be realized manifold.

The firewall can be configured to allow FiTP control connections to a predefined number of auth-servers only. Furthermore it can be configured to limit dynamical opening of ports to specific ip addresses only, dependent on the auth-server, which has authorized these requests (e.g. it could be configured to allow port requests only, denying any request for opening of port ranges). Also a maximum number of parallel data connections allowed could be fixed. There are many other restrictions you could think of.

The same restrictions can be introduced using the alternative scenarios above. There could be a hierarchy of authentication, where the predefined firewall rules are highest rated. The firewall agent could further restrict access dependent on the auth-server who requests data paths. The auth-server itself could further restrict the rules which would be allowed by the firewall agent dependent on the client requesting data connections.

Another way to restrict access via the FiTP protocol may be implemented by looking into the way of authentication and / or authorization, e.g. the auth server as well as the firewalls could allow opening of port ranges only if strong authentication is used. Use of less secure methods would allow opening of single ports only or connections to special servers only. Those kinds of restrictions can be configured into configuration files on the auth server (local policies) or on the firewall (organization wide policies) dependent on the security policy the organization wants to implement.

11 Related Work

The problems firewalls are facing with grid applications have been known for a long time [GFD-083]. To handle communications using variable ports, code has been implemented into firewalls to understand communication behaviors of some of those applications. Everyone is aware of the file transfer protocol (FTP), the session initiation protocol (SIP) and the H.323 protocol. None of those protocols can be simply used in grid applications to signal required port openings.

Grid application programmers normally use their own protocol standards. Unfortunately grid users are a small sub group of the mass of INTERNET users only. Therefore integrating awareness of grid applications into firewalls is cost intensive, but less profitable. Nevertheless some applications have been implemented, mostly for Linux based firewalls, to overcome those grid problems.

Dyna-Fire [DYNA-FIRE] is an extension of the Netfilter iptables software capable of configuring new filtering rules in response to valid requests from users' applications. It uses Port Knocking [Port-Knocking] as signaling protocol, for which a central database provides resource information (containing a list of user based entries for ports to open). Port Knocking is a client-server communication method in which information is encoded in the form of connection attempts to closed ports. The problem with Dyna-Fire is, that the protocol is not transparent, since the clients have to know the IP address of all intermediate firewalls to signal their port requests. Furthermore requirements for every single application and user have to be included into the central resource data base.

Another approach, Cooperative On-Demand Opening [CODO], has been developed which is also an extension to Netfilter iptables. The firewall starts with 3 open ports. Using the first one server declare availability for external communication requests. Port two is used for external clients to signal port requests. The third port provides the same mechanism for internal client. The protocol uses SSL and X.509 certificates. Again the problem is that the protocol is not transparent, since the clients have to know the IP address of all the firewalls on the path.

Generic Connection Brokering [GCB] is a technique where client and server exchange information about which of them should open connections. A server located behind a firewall could open ports to the outside which is less insecure than the other way round. If the client also is located behind a firewall the connection broker can be used to link both connections. In this sense the broker acts as a gateway. To use connection brokerage the applications have to be link to a special GCB library.

Though each UDP packet of a data stream is independent most firewalls interpret consecutive UDP packets with the same quadruple (Source-IP, Source-Port, Dest-IP, Dest-Port) as a steam. UDP Hole Punching [UDP-HP] uses this firewall behavior. An intermediate gateway is used to exchange information (the quadruple) between client and server processes. At most firewalls connections started from internal network segments are allowed. If both, client and server, are synchronized, i.e. use the same quadruple, they can communicate with each other since the firewall interprets packets from one side as answers to packets from the other side. With this technique using a UDP based file transfer, e.g. UDT [UDT] data can be exchanged without any firewall software changes.

Since all this solutions need configuration of servers or firewalls in advance they cannot be used if the port or range of ports is not known prior to application start time.

Also within IETF there have been made a lot of developments dealing with firewalls and NATs.

One protocol which is related to the FiTP protocol is the “Middlebox Communication architecture and framework” [MIDCOM] developed by MIDCOM group of IETF. Here the firewall is freed from any application awareness. MIDCOM agents perform application level gateway functions. They possess application awareness and knowledge of firewall functions. Through the MIDCOM protocol MIDCOM agents signal the firewall to dynamically open application specific ports. So the firewall does not need to know anything about the application. All information is stored and processed by the MIDCOM agent. In this sense a MIDCOM agent is exactly, what is described in chapter 5 using a firewall API to configure a FiTP-unaware firewall. The disadvantage with MIDOM is that a special MIDCOM agent has to be developed for every new application.

The IETF NAT/Firewall NSIS Signaling Layer Protocol (NSLP) [RFC-5973] is a very similar approach to FiTP to signal dynamic port openings to firewalls along the data path. Dynamic configuration includes enabling data flows to traverse these devices without being obstructed, as well as blocking of particular data flows at inbound firewalls. Messages follow the data path through different NSIS forwarders. Every NATFW NSLP-enabled firewall along the data path intercepts this message, processes it, and configures itself accordingly. Thereafter, the actual data flow can traverse all these configured firewalls. NATFW NSLP introduces the split between an NSIS transport layer and an NSIS signaling layer [NSIS-REQ], [NSIS]. The transport of NSLP messages is handled by an NSIS Network Transport Layer Protocol (NTLP, with General Internet Signaling Transport (GIST) [RFC5971] being the implementation of the abstract NTLP). The signaling logic, NATFW is using, is defined in RFC 5973. So using NATSW NSLP requires a second protocol layer to be implemented and used. Furthermore signaling along the path via different NSIS forwarders requires neighboring nodes authentication. This trust relationship, the mechanisms to be used for this and the deployment scenario are out of scope of NATFW NSLP though this is of high importance for any security decision.

12 Summary

The FiTP protocol described below will provide grid applications with a tool for easy opening of firewall ports.

The past has shown that access rules for grid applications will be configured into firewalls for long period though they are used only within short time period. These constantly open ports are a potential security risk which can be minimized when those rules are configured only in time periods where they are really needed.

Usage of the protocol within grid applications minimizes the time period in which traffic can pass the firewall. The other way round it allows opening of ports without manual interaction of a firewall administrator. This leads to fast configuration independent of availability of those administrators. The authenticated and authorized interaction between user applications and authentication/authorization servers provides a secure configuration of the firewalls involved.

It is recommended to use this protocol widely.

13 Security Considerations

This FiTP protocol allows easy dynamic configuration of firewall systems by authorized, but external users. This introduces a potential security risk, which will be analyzed here.

If we think about a normal IPsec communication, which we have allowed to traverse our firewall, security is assumed to be handled by the receiving server process. If we can assume, that the server system has not been hacked, we can postulate, that the authentication and authorization of the remote user has been checked and that he is allowed to use this data connection. If we assume furthermore that the receiving side allows packet forwarding to internal hosts, we have allowed connectivity to any internal host and port via this tunneling technique and only relying on the security checks done by the server system.

The same principle applies to FiTP. We allow control connections to a number of internal servers accessible from remote systems because of access rules within our firewall. Via dynamic requests the external user can ask for access to hosts inside of our organization normally protected by our organizational firewall. We again assume that the internal server has checked authentication and authorization, so that we can postulate that the remote user is allowed to ask for those port openings. The main difference to the model above is, that we now exactly know, where the communication streams are going. Since there is no tunneling of data connections anymore, we are aware of any communication possible. (Of course, the user could again use tunneling techniques on the connections he has got opened.

So the main security difference is the FiTP protocol itself. The main question arises: Can we trust this protocol?

The protocol uses common techniques for secure communication between partners.

- It starts with a key exchange phase, where we agree on a shared key used for the rest of all communications.
- It uses common authentication and authorization techniques to check if the remote user is allowed to request port openings. This authentication and authorization process is encrypted, so that a man-in-the-middle cannot intercept and modify messages exchanged. Exchange of FiTP commands in clear text can only be started after this authentication/authorization.
- And it requests grants, though publicly readable, but protected via the HMAC routines. So any message sent from the corresponding side can be checked by the recipient for any modifications done by man-in-the-middle hackers. If any anomalies arise, both sides of the FiTP control stream must stop the communication. This policy is a strict implementation of the firewall rule: If something is going wrong, do not allow any further access. It is better to allow nothing than allowing everything.

14 Acknowledgements

The document described here has had a lot of iterations until the final version has been materialized. The author would like to thank all persons involved in this process for feedback and comments on the document as well as for proof reading, corrections of spelling, grammar and style. He also would like to acknowledge the presentations from researchers in the OGF FI-RG and FVGA-WG sessions that helped shape this document.

Also he would like to thank the former OGF security area director David Groep, current OGF infrastructure director Richard Hughes-Jones and former OGF infrastructure director Cees de Laat for supporting this work.

A special thank is going to Freek Dykstra, Wolfgang Ziegler and again Richard for proof reading, providing a lot of valuable comments to improving the document, and for correcting grammar, spelling and readability.

15 Author Information

Ralph Niederberger (Editor)
Forschungszentrum Jülich GmbH
P.O.Box
D-52425 Jülich, Germany
r.niederberger@fz-juelich.de

16 Glossary

CIDR	Classless Inter Domain Routing (CIDR) is a method for assigning IP addresses without using the standard IP address classes like Class A, Class B or Class C. In CIDR, depending on the number of hosts present in a network, IP addresses are assigned. In CIDR notation, an IP address is represented as A.B.C.D /n, where "/n" is called the IP prefix or network prefix. The IP prefix identifies the number of significant bits used to identify a network. For example, 192.9.205.22 /18 means, the first 18 bits are used to represent the network and the remaining 14 bits are used to identify hosts.
GridFTP	Special FTP protocol for Grids that allows transferring a file via multiple parallel data sessions.
H.323	H.323 is an umbrella recommendation from the ITU-T that defines the protocols to provide audio-visual communication sessions on any packet-switched network.
HMAC	Keyed-hash message authentication code is a special Message Authentication Code (MAC) used on several protocols like TLS and IPsec and being based on crypthographic hash functions.

IPSec	IP Security, a set of protocols developed by the IETF to support secure exchange of packets at the IP layer. IPsec has been deployed widely to implement Virtual Private Networks (VPNs).
SIP	Session Initiation Protocol. It is an application-layer control protocol that can establish, modify, and terminate multimedia sessions such as Internet telephony calls (VoIP). SIP can also invite participants to already existing sessions, as in multicast conferences. Media can be added to (and removed from) an existing session. SIP transparently supports name mapping and redirection services, which supports personal mobility - users can maintain a single externally visible identifier regardless of their network location. See also RFC 3261, 3262, 3263, 3264, and 3265.
X.509	X.509 is a widely used standard for digital certificates.

17 Intellectual Property Statement

18 Disclaimer

19 Full Copyright Notice

20 References

[TELNET]	Telnet Protocol Specification, J.Postel, J.Reynolds, RFC 854, May 1983
[FTP-RFC]	FTP Protocol, J.Postel, J.Reynolds, RFC 959, http://www.ietf.org/rfc/rfc959.txt, October 1995
[FTP-XSEC]	FTP Security Extensions, M.Horowitz, S.Lunt, RFC 2228, http://www.ietf.org/rfc/rfc2228.txt, October 1997
[GFD-083]	Fiirewall Issues Overview, R.Niederberger (Editor), Open Grid Forum, Oct. 2006, http://www.ogf.org/documents/GFD.83.pdf
[GFD-142]	Requirements on operating Grids in Firewalled Environments, T.Metsch (Editor), Open Grid Forum, Oct. 2008, http://www.ogf.org/documents/GFD.142.pdf
[RFC-2119]	Key words for use in RFCs to Indicate Requirement Levels, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt , S.Bradner
[HMAC-6151]	Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms, S.Turner, L.Chen, RFC 6151, March 2011
[HMAC]	HMAC: Keyed-Hashing for Message Authentication, H.Krawczyk, M.Bellare, R.Canetti, February 1997, RFC 2104, http://www.ietf.org/rfc/rfc2104.txt
[GridFTP-1]	GFD-20: GridFTP: Protocol Extensions to FTP for the Grid, Allock,W. (Editor), Open Grid Forum, April 2003
[GridFTP-2]	GFD-21: GridFTP Protocol Improvements, Mandrichenko,I. (Editor), Open Grid Forum, July 2003
[GridFTP-3]	GFD-47: GridFTP v2 Protocol Description, Mandrichenko,I. (Editor), Open Grid Forum, May 2005
[DYNA-FIRE]	Dyna-Fire, M.L. Green, S.M. Gallo, R. Miller, University at Buffalo, https://forge.ogf.org/sf/wiki/do/viewPage/projects.fi-rg/wiki/DynaFire
[Port-Knocking]	Port Knocking: Network Authentication Across Closed Ports. Krzywinski, M., 2003, SysAdmin Magazine 12: 12-17. http://www.portknocking.org/
[CODO]	CODO: Firewall Traversal by Cooperative On-Demand Opening, S.Son,B.Allcock,M.Livny1, Computer Science Department, University of Wisconsin, Argonne National Laboratory, http://research.cs.wisc.edu/condor/doc/CODO-hpdc.pdf
[GCB]	GCB: Generic Connection Brokering, University of Wisconsin, http://www.cs.wisc.edu/~sschang/firewall/gcb
[UDP-HP]	UDP hole punching, http://en.wikipedia.org/wiki/UDP_hole_punching
[UDT]	UDT: UDP-based Data Transfer for High-Speed Wide Area Networks, Yunhong Gu and Robert L. Grossman, Computer Networks (Elsevier). Volume 51, Issue 7. May 2007.

[MIDCOM]	Middlebox communication architecture and framework, P.Srisuresh, J.Kuthan, J.Rosenberg, A.Molitor, A.Rayhan, RFC 3303, http://www.ietf.org/rfc/rfc3303.txt
[RFC-5971]	GIST: General Internet Signalling Transport, H.Schulzrinne, R.Hancock, October 2010, http://www.ietf.org/rfc/rfc5971.txt
[RFC-5973]	NAT/Firewall NSIS Signaling Layer Protocol (NSLP), M.Stiemerling, H.Tschofenig, C.Aoun, E.Davies, RFC 5973, October 2010, http://datatracker.ietf.org/doc/rfc5973/?include_text=1
[NSIS-REQ]	Requirements for Signaling Protocols, M.Brunner, NEC,April 2004, http://www.ietf.org/rfc/rfc3726.txt
[NSIS]	Next Steps in Signaling (NSIS): Framework, R.Hancock, G.Karagiannis, J.Loughney, S.Van den Bosch, June 2005, http://www.ietf.org/rfc/rfc4080.txt