GWD-R-P
FVGA-WG
http://forge.gridforum.org/sf/projects/fvga-wg
Ralph Niederberger (Editor), Research Center Jülich
August 23, 2011

Firewall Traversal Protocol (FiTP)

Status of This Memo

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

Copyright Notice

Trademark

Abstract

Firewalls control traffic flows between internal and external communication partners. Mostly traffic from inside to outside is allowed, but traffic coming from outside must be explicitly configured. The rules which packets may traverse the firewall and which not are normally configured manually by firewall administrators. 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 some protocols are inspectable by firewalls already 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 standard 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) can be used in two ways. First of all, 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, before installing a system which is FiTP enabled.

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. Firewalls control traffic flows between internal and external communication partners. Mostly traffic from inside to outside is allowed, but traffic coming from outside must be explicitly configured.

Normally configuring firewalls is manually done by firewall administrators. To speed up such kind of access list changes, some protocols, which use parallel sessions belonging to the same application, are known to firewall system software, so that dynamic connection attempts can be allowed though no explicit access rule had been available. One example is the well known FTP protocol, where allowed control connections have been specified within the access list of the firewall. Access rules for the data connection are built dynamically on the fly by inspecting the control connection. Data connections to be used will be 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 on the number of Grid nodes available, data streams needed, data servers available, etc, 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, which allows setting access lists according to the access requests by CLI commands or proprietary software at the firewall.

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 helpful also. Furthermore the reader should have some knowledge about Grid Applications and their issues with Firewalls [GFD-083] and [GFD-142].

2 Overview

In this section, the history, the terminology, and the FiTP model are discussed. The terms defined in this section are only those that have special significance in 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.

Some of the terminology is very specific to the FiTP model; some readers may wish to turn to the section on the FiTP model while reviewing the terminology.

2.1 History

Grid-Projects with external partners lead to communication relationships between external and internal 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.

Supercomputers, PC-clusters and/or special systems had to be connected via dedicated networks assuming a “Net of Trust”, since configuring access rules for those systems and their grid applications would be much to complicated or insecure, because of granted access which is not used for long periods. So compromises of these systems led to increased security problems.

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), weaker policies or no security policies at all, general decreasing the security level to that of the partner installation and security vulnerability because of open ports for long time periods.

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.

2.2 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 transferred itself 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 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 attack 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 DTP 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 which needed the port openings 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.

2.3 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, a secure file transfer over an unsecure connection, i.e. cleartext messages, requires some additional procedures.

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.

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 will 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 server acknowledges the request and if needed, configures the firewall accordingly. (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 will be deleted when the FiTP control connection has ended.

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

- The client has to record any grant request he has sent to the server, to be able to close those requests accordingly.
- The firewall has to control the connection and act on any grant request accepted by the server side. Furthermore, the firewall has to 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 command. For this reason, the firewall has to log status information for every FiTP session in progress.

- The server has to check authentication and authorization of the user and correlate requests with his “user access rights” table. This will allow to differ between rights of individual users. E.g. a system administrator may be allowed to open a bigger port range than other users or a user may be allow to open more ports to a host with a certificate than with userid and password. This server access rights checking is server dependent and not covered within this specification.

2.4 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.

2.5 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 has to 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 will close the data path grants just after the control connection has been ended.

2.6 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 disaster recovery any side can 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.

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. This can be used e.g. to hold up the control connection when a timeout would appear at the remote side otherwise.

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 may be sent by auth_PI process at any time to check if the client is still available. This can be used e.g. 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 is sent by user_PI process for finalizing control connection.

8000,**PCtl,ACKN,**[messageID]

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

9000,**PCtl,DATA,**[messageID]

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

0001, 0002 and 9000 messages can 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, authentication and authorization

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**

The last command sent by user_PI has been illegally formed or is out of sequence. Key exchange and authentication not started yet.

1005,**KeAA,NACK,**[messageID],**Awaiting_KeAA_cmd_with_code_1000**

This is a negative acknowledgement, since the client has not send a 1000 KeAA initiation command.

1009,**KeAA,NACK,**[messageID]

Auth_PI is not ready for key exchange, authentication, and authorization. User_PI has to restart the session with 1000,Auth messages again.

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**

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 can include the HMAC and user_PI can check its correctness. Nevertheless user_PI has to start with a new 2000 sequence.

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**

Similar to user_PI 2002 message above. Auth_PI acknowledges to the user_PI that he also agrees on having negotiated a shared key.

2005,**KeyE,NACK,**[messageID],**Awaiting_KeyE_cmd_with_code_2000**

This is a negative acknowledgement, since the client has not send a 2000 KeyE initiation command.

2006,KeyE,NACK,[messageID],**Awaiting_KeyE_end_cmd_2002**

This is a negative acknowledgement, since the client has not send a 2002 KeyE initiation command.

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”. User_PI has to restart the whole key negotiation and authentication command sequence from beginning again with a 1000 request.

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. User_PI has to restart the whole key negotiation and authentication command sequence from beginning again with a 1000 request.

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**

Negative response to a 3000 request, becauser user_PI message sent is out of sequence or illegally formed.

3005,Auth,NACK,[messageID],**Awaiting_Auth_cmd_with_code_3000**

This is a negative acknowledgement, since the client has not send a 3000 Auth initiation command.

3007,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. User-PI has to start authentication again with 3000 request.

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

Negative response to a 3000 request, because requested authentication method is not supported.

3009,**Auth,NACK,**[messageID]

Auth_PI is not willing to do authentication currently. User_PI has to restart the initiation session with a 1000 request later.

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. If commands and responses become unsynchronized, both sides can reset authentication by a 3011 response.

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

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

3011,Auth,NACK,[messageID],**Form_error**

Negative response to a 3010 request, e.g. user_PI messages sent are out of sequence. User-PI has to start authentication again with 3000 request.

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.

3015,**Auth,NACK,**[messageID],**Awaiting_Auth_cmd_with_code_3010**

This is a negative acknowledgement, since the client has not send a 3010 Auth initiation command.

3018,Auth,NACK,[messageID]

If authentication has not been finished, but the user_PI tries to start authorization the Auth_PI will sent an 3018 response to indicate ”No authentication finished”. User_PI has to restart the authentication command sequence from beginning again with a 3000 request.

3019,**Auth,NACK,**[messageID]

This is the reply an auth_PI sends back to a user_PI when authentication has failed. User_PI has to restart the key exchange session with a 1000 request

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**

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.

4005,**Adat,NACK,**[messageID],**Awaiting_Adat_cmd_with_code_4000**

This is a negative acknowledgement, since the client has not send a 4000 Adat initiation command.

4007,Adat,NACK,[messageID],**No_Authorization_Method_agreed_on**

User_PI request received for an Authorization or Grant Access, but yet no Authorization method has been agreed on. User-PI has to start authorization again with 4000 request.

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

Negative response to a 4000 request, because requested authorization Method is not supported.

4009,Adat,NACK,[messageID]

This is the reply an auth_PI sends back to a user_PI when authorization is not possible currently.

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

4020 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 4020 messages also.

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

Positive response to a user_PI authorization exchange data message with potential answers to the user_PI.

4011,**Adat**,**NACK**,[messageID],**Form_error**

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.

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

User_PI informs auth_PI, that authorization data is complete. User_PI informs auth_PI about the completion of authorization command requests. So this command ends a sequence of 40xx user_PI messages. Since the amount of authorization data is method dependent this allows a sequence of 40xx 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.

4015,**Adat,NACK,**[messageID],**Awaiting_Adat_cmd_with_code_4010**

This is a negative acknowledgement, since the client has not send a 4010 Adat initiation command.

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”. User_PI has to restart the authorization command sequence from beginning again with a 4000 request.

4019,Adat,NACK,[messageID]

This is the reply an auth_PI sends back to a user_PI when authorization has failed. User_PI has to restart the key exchange session with a 1000 request

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 this allows the auth_PI to send responses related to each individual grant request.

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**

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

5005,**GACR,NACK,**[messageID],**Awaiting_GACR_cmds_5x00_or_5x20**

This is a negative acknowledgement, since the client has not send a 5000, 5020, 5600 or 5620 command.

5009,**GACR,NACK,**[messageID],**Deny,***n*

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 or 5020 message, but got a malformed or quite different message.

5022,**GACR,NACK,**[messageID],*n*,**NoConnection**

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**

IPv6 version of server negative response 5001 message

5609,**GACR,NACK,**[messageID],**Deny6,**n

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

5622,**GACR,NACK,**[messageID],{n},**NoConnection**

IPv6 version of server response 5022 message

4 FiTP replies

4.1 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 set of state diagrams 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 the user_PI must check that the appended HMAC to this message is valid, so that it is guaranteed that the response comes from the server and is 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.

xyz{n}

This is a negative response. The previous request cannot be acknowledged. {n} may very from 2 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.)

4.2 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**

1000,**KeAA,DATA,**[messageID]<CRLF>
1000,KeAA,ACKN,[messageID]<CRLF>
1001,KeAA,NACK,[messageID],**Form_error**<CRLF>
1005,**KeAA,NACK,**[messageID],**Awaiting_KeAA_cmd_with_code_1000**
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**,{HMAC}<CRLF>
2002,KeyE,DATA,[messageID],**DONE-Shared_Key_Available**,{HMAC}<CRLF>
2002,KeyE,ACKN,[messageID],**DONE-Shared_Key_Available**,{HMAC}<CRLF>
2005,**KeyE,NACK,**[messageID],**Awaiting_KeyE_cmd_with_code_2000**
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**,{HMAC}<CRLF>
3005,Auth,NACK,[messageID],**Awaiting_Auth_cmd_with_code_3000**
3007,Auth,NACK,[messageID],**No_Authentication_Method_agreed_on**
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**,{HMAC}<CRLF>
3012,Auth,DATA,[messageID],**DONE-Authentication_Complete**,{HMAC}<CRLF>
3012,Auth,ACKN,[messageID],**DONE-Authentication_Complete**,{HMAC}<CRLF>
3015,**Auth,NACK,**[messageID],**Awaiting_Auth_cmd_with_code_3010**
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**,{HMAC}<CRLF>
4005,**Adat,NACK,**[messageID],**Awaiting_Adat_cmd_with_code_4000**
4007,Adat,NACK,[messageID],**No_Authorization_Method_agreed_on**
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**,{HMAC}<CRLF>
4012,Adat,DATA,[messageID],**DONE-Authorization_Complete,**{HMAC}<CRLF>
4012,Adat,ACKN,[messageID],**DONE-Authorization_Complete**,{HMAC}<CRLF>
4015,**Adat,NACK,**[messageID],**Awaiting_Adat_cmd_with_code_4010**
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**,{HMAC}<CRLF>
5005,**GACR,NACK,**[messageID],**Awaiting_GACR_cmds_5x00_or_5x20**
5009,**GACR,NACK,**[messageID],**Deny,***n*,{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>
5022,**GACR,NACK,**[messageID],*n*,**NoConnection**,{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,{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>
5622,**GACR,NACK,**[messageID],{n},**NoConnection**,{HMAC}<CRLF>

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

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

5 Declarative specifications

5.1 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”).

5.2 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 shall 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 should promptly read any remaining data queued on the connection and issue the close on its own side.

5.3 Commands

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

The command syntax is specified here.

The commands begin with a four digit alphanumeric code followed by a human readable phase descriptor and an argument field. 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,meth,certificate**

3030,**Auth,DATA,Meth,certificate**

3030,**Auth,daTA,METh,CertiFicate**

3030,**AuTH,DaTA,MetH,certifiCATE**

3030,**AUTH,DATA,METH,CERTIFICATE**

And any of the following may represent a grant access request command:

5000,GACR,DATA,Allow,

***00020,TCP*,123.045.067.089/32*,12345,12359,124.111/32.222.233,05000,05010**

5000,gacR,data,allow,

***00020,tcp*,123.045.067.089/32*,12345,12359,124.111/32.222.233,05000,05010**

5000,GACR,datA,ALLOW,

***00020,TCP*,123.045.067.089/32*,12345,12359,124.111/32.222.233,05000,05010**

5000,Gacr,Data,Allow,

***00020,Tcp*,123.045.067.089/32*,12345,12359,124.111/32.222.233,05000,05010**

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". Any text string following x"00" will be ignored and deleted.

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.

5.4 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>

5.5 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

5.6 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.

Spontaneous Replies

Sometimes "the system" spontaneously has to sent 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

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

command. Receiving such a message, both sides should 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.

Command-Reply Sequences

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

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

2000,2001,2002,2008,2009

Authentication

Authentication data method request

3000

3000,3001,3008,3009

Authentication data exchanged

3010

3007,3010,3011,3012,3018,3019

Authentication data exchange complete

3012

3010,3011,3012,3018,3019

Authorization

Authorization Method request

4000

4000,4001,4008,4009

Exchanging of authentication data

4010

4010,4011,4012,4018,4019

Grant Access Request and Responses for IPv4

Grant Access (Allow=)

5000

5000,5001,5009

Closing of Access Grants (Close=)

5020

5020,5021,5022

Grant Access Request and Responses for IPv6

Grant Access (Allow6=)

5600

5600,5601,5609

Closing of Access Grants (Close6=)

5620

5620,5621,5622

Connection Closing

8000

8000

Abnormal Closing of connections

9000

9000

6 A typical FiTP scenario

.

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 will communicate to the server “AAS” via this control connection the request to use the communication point quadruple [S,sp,D,dp]. Server “AAS” authenticates the user and checks for his authorization to start communication between these endpoints. All firewalls on this control path are able to read the requested grants and are able to dynamically open the required ports in their access lists. The following message exchanges may be a typical scenario. '---->' represents commands from host U to host AAS, and '<----' represents replies from host AAS to host U.

[FW]

[A] ← --- --- --- --- → [B]

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

← TCP: 2002,KeyE,ACKN,{messageID},DONE

→ TCP: 2020,KeyE,DATA,{messageID},Shared_Key_Available

← TCP: 2020,KeyE,ACKN,{messageID},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

← TCP: 3012,Auth,ACKN,{messageID},DONE

###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,Auth,ACKN,{messageID},DATA,{authorization_data}

→ TCP: 4012,Adat,DATA,{messageID},DONE

← TCP: 4012,Adat,ACKN,{messageID},DONE

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: 3000,GAcR.DATA,{messageID},Allow,
00001,TCP,192.168.015.116/32,30123,30123,172.100.014.123/20,02811,02811

← TCP: 3000,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,

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

Delete grants for Gridftp control connection

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

← TCP: 5000,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: 5000,GAcR.DATA,{messageID}Close,
00002,TCP,192.168.015.116/32,30124,30173,172.100.014.123/32,20001,20050

← TCP: 5000,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.

7 A sample FiTP program in PERL

```
#!/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);

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

################################## Subroutines ###################################
sub call_applications_using_opened_port { print "Starting data connections\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 and agreed on later)

9 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 would 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.

10 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.

11 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.

12 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 OGF security area director David Groep and infrastructure directors Richard Hughes-Jones and Cees de Laat for supporting this work.

13 Author Information

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

14 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 which 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.

15 Intellectual Property Statement

The OGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the OGF Secretariat.

The OGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the OGF Executive Director.

16 Disclaimer

This document and the information contained herein is provided on an “As Is” basis and the OGF disclaims all warranties, express or implied, including but not limited to any warranty that the use of the information herein will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

17 Full Copyright Notice

18 References

[TELNET]	Telnet Protocol Specification, RFC 854, J.Postel, J.Reynolds, May 1983
[FTP-RFC]	FTP Protocol, RFC 959, http://www.ietf.org/rfc/rfc959.txt, J.Postel, J.Reynolds, October 1995
[FTP-XSEC]	M.Horowitz, S.Lunt, FTP Security Extensions, RFC 2228, http://www.ietf.org/rfc/rfc2228.txt, October 1997
[GFD-083]	Niederberger,R. (Editor) Fiirewall Issues Overview, Open Grid Forum, Oct. 2006, http://www.ogf.org/documents/GFD.83.pdf
[GFD-142]	Metsch,T. (Editor), Requirements on operating Grids in Firewalled Environments, , Open Grid Forum, Oct. 2008, http://www.ogf.org/documents/GFD.142.pdf
[HMAC-6151]	Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms, RFC 6151, S.Turner, L.Chen, March 2011
[HMAC]	H.Krawczyk, M.Bellare, R.Canetti, February 1997, HMAC: Keyed-Hashing for Message Authentication, RFC 2104, http://www.ietf.org/rfc/rfc2104.txt
[GridFTP-1]	Allock,W. (Editor), GFD-20: GridFTP: Protocol Extensions to FTP for the Grid, Open Grid Forum, April 2003
[GridFTP-2]	Mandrichenko,I. (Editor), GFD-21: GridFTP Protocol Improvements, Open Grid Forum, July 2003
[GridFTP-3]	Mandrichenko,I. (Editor), GFD-47: GridFTP v2 Protocol Description, Open Grid Forum, May 2005