Internet Engineering Task Force (IETF)
Request for Comments: 6384
Category: Standards Track
ISSN: 2070-1721

I. van Beijnum
Institute IMDEA Networks

October 2011

An FTP Application Layer Gateway (ALG) for IPv6-to-IPv4 Translation

Abstract

The File Transfer Protocol (FTP) has a very long history, and despite the fact that today other options exist to perform file transfers, FTP is still in common use. As such, in situations where some client computers only have IPv6 connectivity while many servers are still IPv4-only and IPv6-to-IPv4 translators are used to bridge that gap, it is important that FTP is made to work through these translators to the best possible extent.

FTP has an active and a passive mode, both as original commands that are IPv4-specific and as extended, IP version agnostic commands. The only FTP mode that works without changes through an IPv6-to-IPv4 translator is extended passive. However, many existing FTP servers do not support this mode, and some clients do not ask for it. This document specifies a middlebox that may solve this mismatch.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6384.

Copyright Notice

Table of Contents

1. Introduction

[RFC0959] specifies two modes of operation for FTP: active mode, in which the server connects back to the client, and passive mode, in which the server opens a port for the client to connect to. Without additional measures, active mode with a client-supplied port does not work through NATs or firewalls. With active mode, the PORT command has an IPv4 address as its argument, and with passive mode, the server responds to the PASV command with an IPv4 address. This makes both the passive and active modes, as originally specified in [RFC0959], incompatible with IPv6. These issues were solved in [RFC2428], which introduces the EPSV (extended passive) command, where the server only responds with a port number and the EPRT (extended port) command, which allows the client to supply either an IPv4 or an IPv6 address (and a port) to the server.

A survey done in April 2009 of 25 randomly picked and/or well-known FTP sites reachable over IPv4 showed that only 12 of them supported EPSV over IPv4. Additionally, only 2 of those 12 indicated that they supported EPSV in response to the FEAT command introduced in [RFC2389] that asks the server to list its supported features. One supported EPSV but not FEAT. In 5 cases, issuing the EPSV command to the server led to a significant delay; in 3 of these cases, a control channel reset followed the delay. Due to lack of additional information, it is impossible to determine conclusively why certain FTP servers reset the control channel connection some time after issuing an EPSV command. However, a reasonable explanation would be that these FTP servers are located behind application-aware firewalls that monitor the control channel session and only allow the creation of data channel sessions to the ports listed in the responses to PASV (and maybe PORT) commands. As the response to an EPSV command is different (a 229 code rather than a 227 code), a firewall that is unaware of the EPSV command would block the subsequent data channel setup attempt. If no data channel connection has been established after some time, the FTP server may decide to terminate the control channel session in an attempt to leave this ambiguous state.

All 25 tested servers were able to successfully complete a transfer in traditional PASV passive mode as required by [RFC1123]. More testing showed that the use of an address family argument with the EPSV command is widely misimplemented or unimplemented in servers. Additional tests with more servers showed that approximately 65% of FTP servers support EPSV successfully and around 96% support PASV successfully. Clients were not extensively tested, but the author's previous experience suggests that most clients support PASV, with the notable exception of the command line client included with Windows, which only supports active mode. This FTP client uses the original PORT command when running over IPv4 and EPRT when running over IPv6.

Although these issues can and should be addressed by modifying clients and servers to support EPSV successfully, such modifications may not appear widely in a timely fashion. Also, network operators who may want to deploy IPv6-to-IPv4 translation generally do not have control over client or server implementations. As such, this document standardizes an FTP Application Layer Gateway (ALG) that will allow unmodified IPv6 FTP clients to interact with unmodified IPv4 FTP servers successfully when using FTP for simple file transfers between a single client and a single server.

Clients that want to engage in more complex behavior, such as server-to-server transfers, may make an FTP Application Layer Gateway (ALG) go into transparent mode by issuing the ALGS command as explained in Section 5.

The recommendations and specifications in this document apply to all forms of IPv6-to-IPv4 translation, including stateless translation such as [RFC6145] as well as stateful translation such as [RFC6146].

This documentation does not deal with the LPRT and LPSV commands specified in [RFC1639] as these commands do not appear to be in significant use.

2. Notational Conventions

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 [RFC2119].

3. Terminology

Within the context of this document, the words "client" and "server" refer to FTP client and server implementations, respectively. An FTP server is understood to be an implementation of the FTP protocol running on a server system with a stable address, waiting for clients to connect and issue commands that eventually start data transfers. Clients interact with servers using the FTP protocol; they store (upload) files to and retrieve (download) files from one or more servers. This either happens interactively under control of a user or is done as an unattended background process. Most operating systems provide a web browser that implements a basic FTP client as well as a command line client. Third-party FTP clients are also widely available.

Other terminology is derived from the documents listed in the References section. Note that this document cannot be fully understood on its own; it depends on background and terminology outlined in the references.

4. ALG Overview

The most robust way to solve an IP version mismatch between FTP clients and FTP servers would be by changing clients and servers rather than using an IPv6-to-IPv4 translator for the data channel and using an Application Layer Gateway on the control channel. As such, it is recommended to update FTP clients and servers as required for IPv6-to-IPv4 translation support where possible to allow proper operation of the FTP protocol without the need for ALGs.

On the other hand, network operators or even network administrators within an organization often have little influence over the FTP client and server implementations used over the network. For those operators and administrators, deploying an ALG may be the only way to

provide a satisfactory customer experience. So, even though not the preferred solution, this document standardizes the functionality of such an ALG in order to promote consistent behavior between ALGs in an effort to minimize their harmful effects.

Operators and administrators are encouraged to only deploy an FTP ALG for IPv6-to-IPv4 translation when the FTP ALG is clearly needed. In the presence of the ALG, EPSV commands that could be handled directly by conforming servers are translated into PASV commands, introducing additional complexity and reducing robustness. As such, a "set and forget" policy on ALGs is not recommended.

Note that the translation of EPSV through all translators and EPRT through a stateless translator is relatively simple, but supporting translation of EPRT through a stateful translator is relatively difficult, because in the latter case, a translation mapping must be set up for each data transfer using parameters that must be learned from the client/server interaction over the control channel. This needs to happen before the EPRT command can be translated into a PORT command and passed on to the server. As such, an ALG used with a stateful translator MUST support EPSV translation and MAY support EPRT translation. However, an ALG used with a stateless translator MUST support EPSV translation and SHOULD also support EPRT translation.

The ALG functionality is described as a function separate from the IPv6-to-IPv4 translation function. However, in the case of EPRT translation, the ALG and translator functions need to be tightly coupled, so if EPRT translation is supported, it is assumed that the ALG and IPv6-to-IPv4 translation functions are integrated within a single device.

5. Control Channel Translation

The IPv6-to-IPv4 FTP ALG intercepts all TCP sessions towards port 21 for IPv6 destination addresses that map to IPv4 destinations reachable through an IPv6-to-IPv4 translator. The FTP ALG implements the Telnet protocol ([RFC0854]), used for control channel interactions, to the degree necessary to interpret commands and responses and re-issue those commands and responses, modifying them as outlined below. Telnet option negotiation attempts by either the client or the server, except for those allowed by [RFC1123], MUST be refused by the FTP ALG without relaying those attempts. For the purpose of Telnet option negotiation, an FTP ALG MUST follow the behavior of an FTP server as specified in [RFC1123], Section 4.1.2.12. This avoids the situation where the client and the server negotiate Telnet options that are unimplemented by the FTP ALG.

There are two ways to implement the control channel ALG:

1. The ALG terminates the IPv6 TCP session, sets up a new IPv4 TCP session towards the IPv4 FTP server, and relays commands and responses back and forth between the two sessions.

2. Packets that are part of the control channel are translated individually.

As they ultimately provide the same result, either implementation strategy, or any other that is functionally equivalent, can be used.

In the second case, an implementation MUST have the ability to track and update TCP sequence numbers when translating packets as well as the ability to break up packets into smaller packets after translation, as the control channel translation could modify the length of the payload portion of the packets in question. Also, FTP commands/responses or Telnet negotiations could straddle packet boundaries, so in order to be able to perform the ALG function, it can prove necessary to reconstitute Telnet negotiations and FTP commands and responses from multiple packets.

Some FTP clients use the TCP urgent data feature when interrupting transfers. An ALG MUST either maintain the semantics of the urgent pointer when translating control channel interactions, even when crossing packet boundaries, or clear the URG bit in the TCP header.

If the client issues the AUTH command, then the client is attempting to negotiate [RFC2228] security mechanisms that are likely to be incompatible with the FTP ALG function. For instance, if the client attempts to negotiate Transport Layer Security (TLS) protection of the control channel ([RFC4217]), an ALG can do one of three things:

1. Transparently copy data transmitted over the control channel back and forth, so the TLS session works as expected but the client commands and server responses are now hidden from the ALG.

2. Block the negotiation of additional security, which will likely make the client and/or the server break off the session, or if not, perform actions in the clear that were supposed to be encrypted.

3. Negotiate with both the client and the server so two separate protected sessions are set up and the ALG is still able to modify client commands and server responses. Again, clients and servers are likely to reject the session because this will be perceived as a man-in-the-middle attack.

An ALG MUST adopt the first option and allow a client and a server to negotiate security mechanisms. To ensure consistent behavior, as soon as the initial AUTH command is issued by the client, an ALG MUST stop translating commands and responses, and start transparently copying TCP data sent by the server to the client and vice versa. The ALG SHOULD ignore the AUTH command and not go into transparent mode if the server response is in the 4xx or 5xx ranges.

It is possible that commands or responses that were sent through the ALG before the AUTH command was issued were changed in length so TCP sequence numbers in packets entering the ALG and packets exiting the ALG no longer match. In transparent mode, the ALG MUST continue to adjust sequence numbers if it was doing so before entering transparent mode as the result of the AUTH command. The ALGS command (Section 11) can also be used to disable the ALG functionality, but the control channel MUST then still be monitored for subsequent ALGS commands that re-enable the ALG functionality.

5.1. Language Negotiation

[RFC2640] specifies the ability for clients and servers to negotiate the language used between the two of them in the descriptive text that accompanies server response codes. Ideally, IPv6-to-IPv4 FTP ALGs would support this feature, so that if a non-default language is negotiated by a client and a server, the ALG also transmits its text messages for translated responses in the negotiated language. However, even if the ALG supports negotiation of the feature, there is no way to make sure that the ALG has text strings for all possible languages. Thus, the situation where the client and server try to negotiate a language not supported by the ALG is unavoidable. The proper behavior for an FTP ALG in this situation may be addressed in a future specification, as the same issue is present in IPv4-to-IPv4 FTP ALGs. For the time being, ALG implementations MAY employ one of the following strategies regarding LANG negotiation:

1. Monitor LANG negotiation and send text in the negotiated language if text in that language is available. If not, text is sent in the default language.

2. Not monitor LANG negotiation. Text is sent in the default language.

3. Block LANG negotiation by translating the LANG command to a NOOP command and translating the resulting 200 response into a 502 response, which is appropriate for unsupported commands. Text is sent in the default language.

In the first two cases, if a language is negotiated, text transmitted by the client or the server MUST be assumed to be encoded in UTF-8 [RFC3629] rather than be limited to 7-bit ASCII. An ALG that implements the first or second option MUST translate and/or forward commands and responses containing UTF-8-encoded text when those occur. The ALG itself MUST NOT generate characters outside the 7-bit ASCII range unless it implements the first option and a language was negotiated.

Note that Section 3.1 of [RFC2640] specifies new handling for spaces and the carriage return (CR) character in pathnames. ALGs that do not block LANG negotiation SHOULD comply with the specified rules for path handling. Implementers should especially note that the NUL (%x00) character is used as an escape whenever a CR character occurs in a pathname.

In the sections that follow, a number of well-known response numbers are shown, along with the descriptive text that is associated with that response number. However, this text is not part of the specification of the response. As such, implementations MAY use the response text shown, or they MAY show a different response text for a given response number. Requirements language only applies to the response number.

6. EPSV to PASV Translation

Although many IPv4 FTP servers support the EPSV command, some servers react adversely to this command (see Section 1 for examples), and there is no reliable way to detect in advance that this will happen. As such, an FTP ALG SHOULD translate all occurrences of the EPSV command issued by the client to the PASV command and reformat a 227 response as a corresponding 229 response. However, an ALG MAY forego EPSV to PASV translation if it has positive knowledge, either gained through administrative configuration or learned dynamically, that EPSV will be successful without translation to PASV.

For instance, if the client issues EPSV (or EPSV 2 to indicate IPv6 as the network protocol), this is translated to the PASV command. If the server with address 192.0.2.31 then responds with:

```
227 Entering Passive Mode (192,0,2,31,237,19)
```

The FTP ALG reformats this as:

```
229 Entering Extended Passive Mode (|||60691|)
```

The ALG SHOULD ignore the IPv4 address in the server's 227 response. This is the behavior that is exhibited by most clients and is needed to work with servers that include [RFC1918] addresses in their 227 responses. However, if the 227 response contains an IPv4 address that does not match the destination of the control channel, the FTP ALG MAY send a 425 response to the client instead of the 229 response, for example:

```
425 Can't open data connection
```

It is important that the response is in the 4xx range to indicate a temporary condition.

If the client issues an EPSV command with a numeric argument other than 2, the ALG MUST NOT pass the command on to the server but rather respond with a 522 error, for example:

```
522 Network protocol not supported
```

If the client issues EPSV ALL, the FTP ALG MUST NOT pass this command to the server, but respond with a 504 error, for example:

```
504 Command not implemented for that parameter
```

This avoids the situation where an FTP server reacts adversely to receiving a PASV command after the client used the EPSV ALL command to indicate that it will only use EPSV during this session.

7. EPRT to PORT Translation

Should the IPv6 client issue an EPRT command, the FTP ALG MAY translate this EPRT command to a PORT command. The translation is different depending on whether the translator is a stateless one-to-one translator or a stateful one-to-many translator.

7.1. Stateless EPRT Translation

If the address specified in the EPRT command is the IPv6 address used by the client for the control channel session, then the FTP ALG reformats the EPRT command into a PORT command with the IPv4 address that maps to the client's IPv6 address. The port number MUST be preserved for compatibility with stateless translators. For instance, if the client with IPv6 address 2001:db8:2::31 issues the following EPRT command:

```
EPRT |2|2001:db8:2::31|5282|
```

Assuming the IPv4 address that goes with 2001:db8:2::31 is 192.0.2.31, the FTP ALG reformats this as:

PORT 192,0,2,31,20,162

If the address specified in the EPRT command is an IPv4 address or an IPv6 address that is not the IPv6 address used by the client for the control session, the ALG SHOULD NOT attempt any translation but pass along the command unchanged.

7.2. Stateful EPRT Translation

If the address in the EPRT command is the IPv6 address used by the client for the control channel, the stateful translator selects an unused port number in combination with the IPv4 address used for the control channel towards the FTP server and sets up a mapping from that transport address to the one specified by the client in the EPRT command. The PORT command with the IPv4 address and port used on the IPv4 side of the mapping is only issued towards the server once the mapping is created. Initially, the mapping is such that either any transport address or the FTP server's IPv4 address with any port number is accepted as a source, but once the three-way handshake is complete, the mapping SHOULD be narrowed to only match the negotiated TCP session.

If the address specified in the EPRT command is an IPv4 address or an IPv6 address that is not the IPv6 address used by the client for the control session, the ALG SHOULD NOT attempt any translation but pass along the command unchanged.

If the client with IPv6 address 2001:db8:2::31 issues the EPRT command:

EPRT |2|2001:db8:2::31|5282|

And the stateful translator uses the address 192.0.2.31 on its IPv4 interface, a mapping with destination address 192.0.2.31 and destination port 60192 towards 2001:db8:2::31 port 5282 may be created, after which the FTP ALG reformats the EPRT command as:

PORT 192,0,2,31,235,32

8. Default Port 20 Translation

If the client does not issue an EPSV/PASV or EPRT/PORT command prior to initiating a file transfer, it is invoking the default active FTP behavior where the server sets up a TCP session towards the client.

In this situation, the source port number is the default FTP data port (port 20), and the destination port is the port the client uses as the source port for the control channel session.

In the case of a stateless translator, this does not pose any problems. In the case of a stateful translator, the translator MAY accept incoming connection requests from the server on the IPv4 side if the transport addresses match that of an existing FTP control channel session, with the exception that the control channel session uses port 21 and the new session port 20. In this case, a mapping is set up towards the same transport address on the IPv6 side that is used for the matching FTP control channel session.

An ALG/translator MAY monitor the progress of FTP control channels and only attempt to perform a mapping when an FTP client has started a file transfer without issuing the EPSV, PASV, EPRT, or PORT commands.

9. Both PORT and PASV

[RFC0959] allows a client to issue both PORT and PASV to use non-default ports on both sides of the connection. However, this is incompatible with the notion that with PASV, the data connection is made from the client to the server, while PORT reaffirms the default behavior where the server connects to the client. As such, the behavior of an ALG is undefined when a client issues both PASV and PORT. Implementations SHOULD NOT try to detect the situation where both PASV and PORT commands are issued prior to a command that initiates a transfer, but rather, translate commands as they occur. So, if a client issues PASV, PASV is then translated to EPSV. If after that, but before any transfers have occurred, the client issues PORT and the ALG supports PORT translation for this session, the ALG translates PORT to EPRT.

10. Default Behavior

Whenever the client issues a command that the ALG is not set up to translate (because the command is not specified in this document, the command is not part of any FTP specification, the ALG functionality is disabled administratively for the command in question, or translation does not apply for any other reason), the command MUST be passed on to the server without modification, and the server response MUST be passed on to the client without modification. For example, if the client issues the PASV command, this command is passed on to the server transparently, and the server’s response is passed on to the client transparently.

11. The ALGS Command

ALGs MUST support the new ALGS (ALG status) command that allows clients to query and set the ALG's status. FTP servers (as opposed to ALGs) MUST NOT perform any actions upon receiving the ALGS command. However, FTP servers MUST still send a response. If FTP servers recognize the ALGS command, the best course of action would be to return a 202 response:

```
202 Command not implemented, superfluous at this site
```

However, there is no reason for FTP servers to specifically recognize this command; returning any 50x response that is normally returned when commands are not recognized is appropriate.

A client can use the ALGS command to request the ALG's status and to enable and disable EPSV to PASV translation and, if implemented, EPRT to PORT translation. There are three possible arguments to the ALGS command:

ALGS STATUS64	The ALG is requested to return the EPSV and EPRT translation status.
ALGS ENABLE64	The ALG is requested to enable translation.
ALGS DISABLE64	The ALG is requested to disable translation.

The ALG MUST enable or disable EPSV to PASV translation as requested. If EPRT to PORT translation is supported, ALGS ENABLE64 SHOULD enable it, and ALGS DISABLE64 MUST disable it along with enabling or disabling EPSV to PASV translation, respectively. If EPRT to PORT translation is not supported, ALGS ENABLE64 only enables EPSV to PASV translation. After an ALGS command with any of the three supported arguments, the ALG MUST return a 216 response indicating the type of translation that will be performed.

216 NONE	Neither EPSV nor EPRT translation is performed.
216 EPSV	EPSV is translated to PASV; no EPRT translation is performed.
216 EPSVEPRT	EPSV is translated to PASV; EPRT is translated to PORT.

The translation type MAY be followed by a space and additional descriptive text until end-of-line. If the ALG is unable to set the requested translation mode, for instance, because of lack of certain

resources, this is not considered an error condition. In those cases, the ALG returns a 216 response followed by the keyword that indicates the current translation status of the ALG.

If there is no argument to the ALGS command, or the argument is not one of STATUS64, ENABLE64, or DISABLE64 (or an argument specified by a supported newer document), a 504 or 502 error SHOULD be returned.

The Augmented Backus-Naur Form (ABNF) notation (see [RFC5234]) of the ALGS command and its response are as follows:

```
algs-command      = "ALGS" SP algs-token CRLF
algs-token        = "STATUS64" / "ENABLE64" / "DISABLE64"

algs-response     = (ok-response / error-response) CRLF
ok-response       = "216" SP response-token [ freetext ]
response-token    = "NONE" / "EPSV" / "EPSVEPRT"
error-response    = not-implemented / invalid-parameter
not-implemented   = "502" [ freetext ]
invalid-parameter = "504" [ freetext ]
freetext          = (SP *VCHAR)
```

12. Timeouts and Translating to NOOP

Wherever possible, control channels SHOULD NOT time out while there is an active data channel. A timeout of at least 30 seconds is RECOMMENDED for data channel mappings created by the FTP ALG that are waiting for initial packets.

Whenever a command from the client is not propagated to the server, the FTP ALG instead issues a NOOP command in order to keep the keepalive state between the client and the server synchronized. The response to the NOOP command MUST NOT be relayed back to the client. An implementation MAY wait for the server to return the 200 response to the NOOP command and translate that 200 response into the response the ALG is required to return to the client. This way, the ALG never has to create new packets to send to the client, but it can limit itself to modifying packets transmitted by the server. If the server responds with something other than a 200 response to the NOOP command, the ALG SHOULD tear down the control channel session and log an error.

13. IANA Considerations

IANA has added the following entry to the "FTP Commands and Extensions" registry:

Command Name	ALGS
FEAT Code	-N/A-
Description	FTP64 ALG status
Command Type	-N/A-
Conformance Requirements	o
Reference	RFC 6384 Section 11

14. Security Considerations

In the majority of cases, FTP is used without further security mechanisms. This allows an attacker with passive interception capabilities to obtain the login credentials and an attacker that can modify packets to change the data transferred. However, FTP can be used with TLS in order to solve these issues. IPv6-to-IPv4 translation and the FTP ALG do not impact the security issues in the former case nor the use of TLS in the latter case. However, if FTP is used with TLS as per [RFC4217], or another authentication mechanism that the ALG is aware of, the ALG function is not performed so only passive transfers from a server that implements EPSV or a client that supports PASV will succeed.

For general FTP security considerations, see [RFC2577].

15. Contributors

Dan Wing, Kentaro Ebisawa, Remi Denis-Courmont, Mayuresh Bakshi, Sarat Kamisetty, Reinaldo Penno, Alun Jones, Dave Thaler, Mohammed Boucadair, Mikael Abrahamsson, Dapeng Liu, Michael Liu, Andrew Sullivan, Anthony Bryan, Ed Jankiewicz Pekka Savola, Fernando Gont, Rockson Li, and Donald Eastlake contributed ideas and comments. Dan Wing’s experiments with a large number of FTP servers were very illuminating; many of the choices underlying this document are based on his results.

16. Acknowledgements

Iljitsch van Beijnum is partly funded by Trilogy, a research project supported by the European Commission under its Seventh Framework Program.

17. References

17.1. Normative References

[RFC0854] Postel, J. and J. Reynolds, "Telnet Protocol Specification", STD 8, RFC 854, May 1983.

[RFC0959] Postel, J. and J. Reynolds, "File Transfer Protocol", STD 9, RFC 959, October 1985.

[RFC1123] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2228] Horowitz, M., "FTP Security Extensions", RFC 2228, October 1997.

[RFC2428] Allman, M., Ostermann, S., and C. Metz, "FTP Extensions for IPv6 and NATs", RFC 2428, September 1998.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

17.2. Informative References

[RFC1639] Piscitello, D., "FTP Operation Over Big Address Records (FOOBAR)", RFC 1639, June 1994.

[RFC1918] Rekhter, Y., Moskowitz, R., Karrenberg, D., Groot, G., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, February 1996.

[RFC2389] Hethmon, P. and R. Elz, "Feature negotiation mechanism for the File Transfer Protocol", RFC 2389, August 1998.

[RFC2577] Allman, M. and S. Ostermann, "FTP Security Considerations", RFC 2577, May 1999.

[RFC2640] Curtin, B., "Internationalization of the File Transfer Protocol", RFC 2640, July 1999.

[RFC4217] Ford-Hutchinson, P., "Securing FTP with TLS", RFC 4217, October 2005.

[RFC6145] Li, X., Bao, C., and F. Baker, "IP/ICMP Translation Algorithm", RFC 6145, April 2011.

[RFC6146] Bagnulo, M., Matthews, P., and I. van Beijnum, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers", RFC 6146, April 2011.

Author's Address

Iljitsch van Beijnum
Institute IMDEA Networks
Avda. del Mar Mediterraneo, 22
Leganes, Madrid 28918
Spain

EMail: iljitsch@muada.com