INTERNET-DRAFT M. Rosenau
Expires: March 18, 2009 September 18, 2008
Category: Experimental

FTP EXTENSION ALLOWING IP FORWARDING (NATs)
<draft-rosenau-ftp-single-port-05.txt>

Status of this Memo

Copyright Notice

Abstract

The FTP protocol [1] needs two connections. A control and a data connection. Using networks with "port forwarding" (eg. SSH, DSL routers, VPN etc.) there is often only the possibility for the server to listen on only one TCP port. In such networks the traditional FTP protocol cannot be used.

This memo describes an extension to the FTP protocol that allows the use of FTP using only one TCP port number for both control connections and passive-mode data connections.

The extension can also be used to use the FTP protocol over network protocols that do not have a "port" concept (such as the NetBIOS protocol).

Changes in version "-01"

- The SPDT command was no longer proposed. Instead a method using a special user name and password is proposed.

- The data returned by SPSV is only valid for one connection. This is necessary for better resource management within the FTP server. version-00 did not clarify this.

- Spelling or grammar mistakes were corrected and unclear statements were replaced.

Change in version "-02"

- Added chapter 6 about implementation issues.

Change in version "-03"

- Fixed a mistake in chapter 6

Changes in version "-04"

The changes in version "-04" are done because of comments of a third party:

- Changed back to the SPDT variant no longer proposing the method using a special user name and password.

- Added chapter 7 about security issues.

Changes in this version

- Changed "user name and password" to "identifier" in chapter 3 because the "SPDT" variant is used instead of a "USER" based variant.

- Changed many language, grammar, terms, ... due to a third party review

1. INTRODUCTION

To transfer data via FTP [1] you need two connections: A control connection and a data connection.

In traditional FTP there are two methods to establish the data connection:

1) The active method: The server establishes the data connection; the client must send its address to the server and listen on a TCP port.

2) The passive method: The server sends its address (including the TCP port number) to the client, which establishes the connection.

The active method requires that the client can listen for incoming connections and that the server can establish a connection to the client. The client must know its own address as it is seen by the server. This is not the case when the client is behind a firewall [3].

The passive method requires the server to be able to listen for incoming TCP connections on a second port and to know its own IP address. If the server is behind a NAT firewall with TCP port forwarding this is often not the case.

Both variants do originally only work with IPv4; FTP using other network protocols requires special extensions such as the "EPRT" [4] or the "LPRT" [2] command. Both extensions do not work with protocols that do not have a "port" concept (such as NetBIOS).

This draft describes an extension to the FTP protocol that allows using FTP with TCP port forwarding ("IP forwarding") or network protocols that do not have a "port" concept.

2. OVERVIEW

The reason for the problems described above is the fact that either the server or the client must know its own transport address (IP address AND port number) and tell it to the other side. The server needs a second transport address (port) if using passive mode. This can be a problem when using TCP port forwarding. (Note that in this case neither of the two computers knows its own address and port number.)

The extension proposed here allows the server to use the same transport address (port) for passive mode data connections as it is used for the control connection.

To establish a data connection, the client establishes a second connection to the same port that is used for the control connection and tells the server that it desires to establish a data connection.

Because the client already knows the transport address of the control connection (it would not have been able to connect otherwise) neither the server nor the client need to know their own transport addresses.

3. EXTENSION COMMANDS ADDED

3.1. USE SINGLE PORT PASSIVE MODE ("SPSV" proposed)

Command syntax:

```
    "SPSV" CRLF
```

This command is sent by the client to indicate that it wants to use the method described in this document to establish a data connection. The command has no parameter.

If the command succeeds the server will reply

```
    "227" SP any informational text "(" identifier ")" CRLF
```

The identifier in parentheses is used to establish a data connection using the "SPDT" command described later.

The identifier MUST NOT contain parentheses, colons or characters above 126 or below 33. The identifier must not be longer than 32 characters. The identifier is case-sensitive. The informational text MUST NOT contain parentheses.

Unlike the "PASV" command the data replied to "SPSV" is only valid for one data connection. After establishing a data connection "SPSV" must be sent again to get a new identifier before establishing a new data connection. This is necessary to allow the server to implement the "Service not ready" behaviour as it is described below.

3.2. ESTABLISHING A DATA CONNECTION ("SPDT" proposed)

Command syntax:

```
"SPDT" SP identifier CRLF
```

This command is the only command sent over the connection dedicated to become the data connection. It is sent instead of a "USER" command.

The argument is the identifier received in the answer to the "use singe port passive mode" command.

If the server accepted the command it returns EXACTLY the following sequence:

```
"200" SP "DATA" CR LF
```

After transmitting this sequence the former control connection becomes the new data connection.

If the identifier was not correct the server returns a reply with the code 504 and drops the TCP connection.

The server SHOULD be prepared to accept the command instead of the "PASS" command or directly after the "PASS" command because some FTP proxy servers require sending an USER/PASS combination before establishing a connection.

4. EXAMPLE TRANSMISSION SCENARIO

(Each line in the following scenario ends with a CRLF sequence that is not written because of better readability.)

```
- The control connection ("--1-->") is established.
  S--1-->C   220 FTP server ready
  C--1-->S   USER u001
  S--1-->C   331 Enter password
  C--1-->S   PASS xyz
  S--1-->C   230 You are logged in
  C--1-->S   SPSV
  S--1-->C   227 Entering single-port mode (xYab1234)
- The data connection ("---2->") is established
  S---2->C   220 FTP server ready
  C---2->S   SPDT xYab1234
  S---2->C   200 DATA
  C--1-->S   RETR example.file
  S--1-->C   150 Transmitting data
  S---2->C   (Contents of example.file (not followed by CRLF))
- The server closes the data connection ("---2->")
  S--1-->C   226 Data transferred
  ...
```

5. THE "SERVICE NOT READY" MESSAGE

If there are too many control connections open many FTP servers deny new users to connect to the server. The server sends the following message:

```
    "120 Too many users connected. Try later." CRLF
```

and drops the connection.

If a client sent an "SPSV" command but did not establish a data connection, yet, this is not possible. In this case the following behaviour is proposed (CRLF not shown here):

```
   <--- 220 Too many users; only SPSV connections allowed
   ---> SPDT xYz1234
   <--- 200 DATA
```

or with a real user name:

```
   <--- 220 Too many users; only SPSV connections allowed
   ---> USER u001
   <--- 421 Too many users connected. Try later.
```

6. POSSIBLE IMPLEMENTATION

In many implementations a new process is started everytime a connection to the FTP service port is opened. The "inetd" service on UNIX operating systems is an example for this behavior.

Adding the "SPSV" option to existing FTP server programs may be done by implementing two extensions:

```
Extension behavior on SPSV:
  The "SPSV" command behaves like the "PASV" command with the
  exception that the format of the response is different. Example:
    --> SPSV
      (Server is actually doing "PASV"; TCP port is 12345 in this
      example; the response differs from "PASV":)
    <-- 227 Entering single-port mode (12345)

Extension behaviour on SPDT:
  The SPDT command is used to create a tunneling connection to a
  port number given in the argument. Example:
    <-- 220 FTP server ready
    --> SPDT 80
      (Server tries to establish a connection to localhost:80
      which is actually the HTTP server)
    <-- 200 DATA
      (Server is simply "tunneling" the data between the two
      TCP connections)
    --> GET /index.html HTTP/1.0
    -->
    <-- HTTP/1.0 200 OK
    <-- Content-type: text/html
    <--
    <-- <HTML>
    <-- Hello world!
    <-- </HTML>
```

This simple way of implementing the extensions would make it easy to bypass server security systems such as firewalls. To avoid this such an implementation should first check if the port number given by the "SPDT" argument really is used by the FTP service.

7. SECURITY ISSUES

This extension can be combined with any other FTP extension. Therefore all security extensions of FTP can be combined with this extension.

Extensions that normally would transfer data over the data connection after it has been established will transfer their data after receiving the "200 DATA" CRLF characters.

When using an implementation that is based on tunneling (as described in chapter 6) the process receiving a "SPDT" command should not allow a tunneling connection to a non-FTP process. Using non-TCP/IP connections (e.g. AF_UNIX) for tunneling could make this easier.

The "identifier" should contain information that is really needed for establishing the connection (such as the port number in the example in chapter 6) as well as a random number cookie that is used to verify if the identifier is really valid. The SPDT command will check if the random number part in the identifier matches the one generated by the SPSV command.

The port number should also not be included directly in the identifier but it should only be transferred encrypted or indirectly (e.g. file name of a temporary file containing the port number). The "file name" variant would also ensure that SPDT can only tunnel FTP connections (and not HTTP connections as shown in chapter 6.)

REFERENCES

[1] FILE TRANSFER PROTOCOL (FTP), RFC 959
[2] FTP Operation Over Big Address Records, RFC 1545
[3] Firewall-Friendly FTP, RFC 1579
[4] FTP Extensions for IPv6 and NATs, RFC 2428

Author's Address

Martin Rosenau
Johannes Schuster Weg 14
76185 Karlsruhe
Germany

Email: martin@rosenau-ka.de