File Transfer Guidelines

Scope

The scope of this guide is to document, explain, and provide best practices for the company product, file transfers. The guide includes information about the different protocols and clients used and examples of scripts that can be used for future file transfers.

Communication

File transfers are a client to server connection over an area network. In general, there are two paths the communication will take, connections will traverse either the Internet or an encrypted tunnel created by a VPN device.

Connections over a vpn device will use a private IP address in the following ranges 10.0.0.0 to 10.255.255.255, 172.16.0.0 to 172.31.255.255, and 192.168.0.0 to 192.168.255.255. Note that for company hosted IP's they will use the following convention, 10.102.x.x or 10.202.x.x.

Connections over the internet will use a public IP address or a url. A public IP address is any IP not included in the private IP address ranges above. For example, the company sftp site uses the url example.com and the IP address of 1.1.1.1.

To pass security measures, the connections will use specific port numbers; these port numbers can be specific by protocol or use a custom port defined by the server owner. The default port numbers by protocol are as follows, Secure File Transfer Protocol (SFTP) Port 22, File Transfer Protocol over TLS (FTPS) Ports 21 or 990, and File Transfer Protocol (FTP) Port 21.

Protocols

A file transfer protocol is the language a client and server use to communicate. Common protocols include SFTP, FTP over TLS, and FTP. SFTP and FTP over TLS are encrypted protocols, while FTP is not encrypted.

SFTP uses Secure Shell encryption to secure communication, for more information read this article. SFTP supports username/password and private key authentication. Either method can be used on its own or together to authenticate the client to the server.

FTPS uses a TLS certificate to encrypt communication, which is the same method used by https sites. Encryption can be Implicit or Explicit, for more information read this article. FTPS uses username/password authentication.

FTP communicates in plain text between the client and server. FTP uses username/password authentication or anonymous authentication where all connections are accepted.

Clients

This section includes descriptions of each file transfer client used in the company hosted environment.

FTP.exe

FTP exe is a command line ftp client included with all Windows operating systems. FTP exe supports the ftp protocol. Documentation can be found here.

FTPS.exe

FTPS exe is a command line FTPS client, it can be downloaded for free from here. Documentation can be found here. FTPS exe supports the FTPS protocol.

PSFTP

PSFTP is a command line SFTP client, it can be downloaded for free from here. Documentation can be found here. PSFTP supports the SFTP protocol.

WinSCP

WinSCP support all protocols and can be used on the command line or through a graphical interface. WinSCP can be downloaded here. The documentation can be found here.

Logging

WinSCP is the only client listed that has native logging capabilities; to enable logging in FTP exe, FTPS exe, and PSFTP the command line output must be redirected to a text file. This has the disadvantage of lacking the authentication and startup conversation between the client and server. If there is an error during this phase of the connection, it will not be logged by the client. If a client connection with FTP exe, FTPS exe, or PSFTP consistently fails with no log output, it may be necessary to connect with WinSCP to troubleshoot the error.

Examples

Comments are inside the following characters, /* */, those lines should be ignored when creating a new script file.

Batch (.bat) and script (.txt) files must be used in combination, there will be a .bat and .txt for each script, the .bat code will typically be the same used with multiple script files.

FTP exe

Batch File (.bat)

FTP Batch
```
/* Batch file commands (.bat) */
cd "D:\Path to Script File"
ftp.exe -s:"Script File Name.txt" -i > "D:\Path to Log File"
```

Download Files (Get)

```
/* Text file commands (.txt)
For downloading multiple files the m prefix is needed on each command, e.g.
mget, mdelete */
open 192.168.1.1
username
password
lcd "Folder where files will be downloaded"
cd "Folder on ftp server"
mget *
mdelete *
bye
```

Upload Files (Put)

```
/* Text file commands (.txt)
For uploading multiple files, the m prefix is needed on each command, e.g.
Mput */
open 192.168.1.1
username
password
lcd "Folder where files will be downloaded"
cd "Folder on ftp server"
mput *
bye
```

FTPS exe

Batch File (.bat)

```
/* Batch file commands (.bat)
The following example uses implicit encryption on port 990, depending on
the server that may need to be changed. */
CD "\Path to Script file"
ftps.exe -e:implicit -z -a -user:username -password:password ftp.url.com 990
-s:ScriptFileName.txt > "Path to Log File"
```

Download Files (Get)

```
/* Text File Commands (.txt)
For downloading multiple files the m prefix is needed on each command, e.g.
mget, mdelete. The prompt command will ignore interactive prompts, this is
not required for a single file with get. FTPS does not have a lcd command, it
will download to the folder in which the script was started. */
prompt
cd "Folder on remote server"
mget *.*
mdelete *.*
quit
```

Upload Files (Put)

```
/* Text File Commands (.txt)
For downloading multiple files the m prefix is needed on each command, e.g.
mget, mdelete. For uploading multiple files with mput, the prompt command is
needed to ignore interactive prompts, this is not required for a single file
with put. */
prompt
cd "Folder on remote server"
mput "Path to Local file"
quit
```

PSFTP

Batch File (.bat)

```
/* Batch File Commands (.bat)
The -be switch tells the client to continue on error, this may not be needed
for some setups. This setup is for user/password and private key
authentication, if the private key is not needed the private key, remove that
section. */
cd "Path to Script folder"
psftp.exe ftp.bankurl.com -l username -i "Path to Private Key File" -pw
password -hostkey hostkeyvalue -b ScriptFile.txt -bc -be > "Path to Log File"
```

Download Files (Get)

```
/* Text File Commands (.txt)
For downloading multiple files the m prefix is needed on the get command,
e.g. mget. */
lcd "Folder where files will be downloaded"
cd "Remove Server folder"
mget *.*
del *.*
quit
```

Upload Files (Put)

```
/* Text File Commands (.txt)
For uploading multiple files the m prefix is needed on the get command, e.g.
mput. */
lcd "Folder where files will be downloaded"
cd "Remove Server folder"
mput *.*
quit
```

WinSCP

Batch File (.bat)

```
/* Batch File commands (.bat) */
cd "Path to Script Folder"
winscp.exe /console /ini=nul /script="ScriptFile.txt" /log="Logs/LogFile.log"
/rawconfig Logging\LogFileAppend=0
```

Download Files (Get)

```
/* Text File commands (.txt) */
open sftp://user:password@FI.url/ -hostkey="hostkey value"
lcd "Local Server Path"
cd "Remote Server Path"
get -delete *.*
exit
```

Upload Files (Put)

```
/* Text File commands (.txt) */
open sftp://user:password@FI.url/ -hostkey="hostkey value"
cd "Remote Server Path"
put "Path to file *.*"
exit
```

Advanced Scripting

Evaluate Exit Code

There may be times when you need to evaluate the exit code of the ftp script to determine a course of action if there was an error. Use the example below to accomplish this. This method can be used to create alerts for job failures, detect files on the remote server and create a trigger, or resubmit a job that previously failed. **Note**: PSFTP has an additional exit code of 2, this will need to be accounted for when implementing this method with psftp.

```
/* Add the following to the end of the batch file. Zero is success, 1 is a
failure. */

IF %ERRORLEVEL% EQU 0 Echo Success > "Path to File"

IF %ERRORLEVEL% EQU 1 Echo Error > "Path to File"
```

File Delete Control

To have control over what files are deleted from the remote server after download, use the following method. This script will download everything in the remove folder to the local folder, then delete only successfully downloaded files. **Note**: this method uses the winscp .net assembly and is not available for any other client.

```
/* This requires the WinSCPnet.dll file to be in the folder as WinSCP.exe
Execute this by saving the code below in a .ps1 file, then call it as follows
powershell.exe -ExecutionPolicy Unrestricted -File "Path to ps1 file" */

param (
    $localPath = "Local Download Path",
    $remotePath = "Remote Server Path"
)

try
{
```

```
# Load WinSCP .NET assembly
Add-Type -Path "D:\PathtoFile\WinSCPnet.dll"

# Setup session options
$sessionOptions = New-Object WinSCP.SessionOptions -Property @{
    Protocol = [WinSCP.Protocol]::Sftp
    HostName = "Remote Server Address"
    UserName = "username"
    Password = "password"
    SshHostKeyFingerprint = "hostkey value"
    SshPrivateKeyPath = "Path to Private Key"
}

$session = New-Object WinSCP.Session

try
{
    $session.SessionLogPath = "Path to Log File"
    $session.AddRawConfiguration("Logging\LogFileAppend", "0")
    # Connect
    $session.Open($sessionOptions)

    # Synchronize files to local directory, collect results
    $synchronizationResult = $session.SynchronizeDirectories(
    [WinSCP.SynchronizationMode]::Local, $localPath, $remotePath, $False)

    # Deliberately not calling $synchronizationResult.Check
    # as that would abort our script on any error.
    # We will find any error in the loop below
    # (note that $synchronizationResult.Downloads is the only operation
    # collection of SynchronizationResult that can contain any items,
    # as we are not removing nor uploading anything)

    # Iterate over every download
    foreach ($download in $synchronizationResult.Downloads)
    {
        # Success or error?
        if ($download.Error -eq $Null)
        {
           Write-Host "Download of $($download.FileName) succeeded,
           removing from source"
            # Download succeeded, remove file from source
            $filename =
           [WinSCP.RemotePath]::EscapeFileMask($download.FileName)
            $removalResult = $session.RemoveFiles($filename)

            if ($removalResult.IsSuccess)
            {
              Write-Host "Removing of file $($download.FileName)
              succeeded"
            }
            else
            {
           Write-Host "Removing of file $($download.FileName) failed"
           Send-MailMessage -To "Email <address@email.com>" -From "Email
           <address@email.com>" -Subject "Email Message" -SmtpServer
           "server name"
```

```
                }
            }
            else
            {
                Write-Host (
               "Download of $($download.FileName) failed:
               $($download.Error.Message)")
            }
        }
    }
    finally
    {
        # Disconnect, clean up
        $session.Dispose()
    }

    exit 0
}
catch
{
    Write-Host "Error: $($_.Exception.Message)"
    exit 1
}
```

File Check

Use the following example to check for all files before beginning a file transfer.

```
/* Change the paths in $patharray match the file paths on the file share. */

$patharray = "C:\FolderPath\Trigger.txt", "C:\FolderPath\File1.file",
"C:\FolderPath\File2.File", "C:\FolderPath\File3.file"
$dir = ($patharray | Test-Path) -notcontains $false

if($dir -eq $True) {
    remove-item "Path to Trigger" -Verbose
    remove-item "Path to temp files" -Verbose

    move-item "Path to file File Share" "Path to Temp folder" -Verbose

    copy-item "Path to temp folder" "Path to prod folder" -Verbose

    copy-item "Path to Trigger backup" "Path to prod folder" -Verbose

    write-host "got production processing files"
}

else
{
    Write-Host "Trigger doesn't exist"
        $host.SetShouldExit(999)
}
```