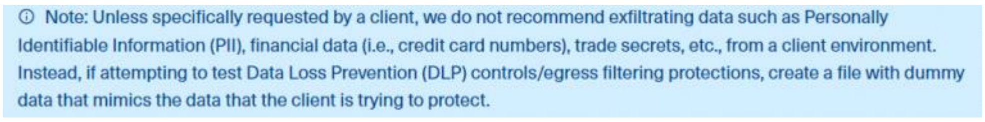
# Protected File Transfers

vendredi 4 octobre 2024 6:39 PM

As penetration testers, we often gain access to highly sensitive data such as user lists, credentials (i.e., downloading the NTDS.dit file for offline password cracking), and enumeration data that can contain critical information about the organization's network infrastructure, and Ac tive Directory (AD) environment, etc. Therefore, it is essential to encrypt this data or use encrypted data connections such as SSH, SFTP, and HT TPS. However, sometimes these options are not available to us, and a different approach is required.


Note: Unless specifically requested by a client, we do not recommend exfiltrating data such as Personally Identifiable Information (PII), financial data (i.e., credit card numbers), trade secrets, etc., from a client environment. Instead, if attempting to test Data Loss Prevention (DLP) controls/egress filtering protections, create a file with dummy data that mimics the data that the client is trying to protect.

Therefore, encrypting the data or files before a transfer is often necessary to prevent the data from being read if intercept ed in transit.

Data leakage during a penetration test could have severe consequences for the penetration tester, their company, and the clie nt. As information security professionals, we must act professionally and responsibly and **take all measures to protect any data we encounter during an assessment.**

| File Encryption | Technique |
|---|---|
| **Windows** | Many different methods can be used to encrypt files and information on Windows systems. One of the simplest methods is the [Invoke-AESEncryption.ps1]() PowerShell script. **This script is small and provides encryption of files and strings.** |

## Invoke-AESEncryption.ps1

```
.EXAMPLE ( Encryption  Text )
Invoke-AESEncryption -Mode Encrypt -Key "p@ssw0rd" -Text "Secret Text"

Description
-----------
Encrypts the string "Secret Test" and outputs a Base64 encoded ciphertext.

.EXAMPLE ( Decryption )
Invoke-AESEncryption -Mode Decrypt -Key "p@ssw0rd" -Text "LtxcRelxrDLrDB9rBD6JrfX/czKjZ2CUJkrg++kAMfs="

Description
-----------
Decrypts the Base64 encoded string "LtxcRelxrDLrDB9rBD6JrfX/czKjZ2CUJkrg++kAMfs=" and outputs plain text.

.EXAMPLE   ( Encryption  File )
Invoke-AESEncryption -Mode Encrypt -Key "p@ssw0rd" -Path file.bin

Description
-----------
Encrypts the file "file.bin" and outputs an encrypted file "file.bin.aes"

.EXAMPLE
Invoke-AESEncryption -Mode Decrypt -Key "p@ssw0rd" -Path file.bin.aes

Description
-----------
Decrypts the file "file.bin.aes" and outputs an encrypted file "file.bin"

#>
function Invoke-AESEncryption {
    [CmdletBinding()]
    [OutputType([string])]
    Param
    (
        [Parameter(Mandatory = $true)]
        [ValidateSet('Encrypt', 'Decrypt')]
        [String]$Mode,

        [Parameter(Mandatory = $true)]
        [String]$Key,

        [Parameter(Mandatory = $true, ParameterSetName = "CryptText")]
        [String]$Text,
        [Parameter(Mandatory = $true, ParameterSetName = "CryptFile")]
        [String]$Path
    )

    Begin {
        $shaManaged = New-Object System.Security.Cryptography.SHA256Managed
        $aesManaged = New-Object System.Security.Cryptography.AesManaged
        $aesManaged.Mode = [System.Security.Cryptography.CipherMode]::CBC
        $aesManaged.Padding = [System.Security.Cryptography.PaddingMode]::Zeros
        $aesManaged.BlockSize = 128
        $aesManaged.KeySize = 256
    }

    Process {
        $aesManaged.Key = $shaManaged.ComputeHash([System.Text.Encoding]::UTF8.GetBytes($Key))

        switch ($Mode) {
            'Encrypt' {
                if ($Text) {$plainBytes = [System.Text.Encoding]::UTF8.GetBytes($Text)}

                if ($Path) {
                    $File = Get-Item -Path $Path -ErrorAction SilentlyContinue
                    if (!$File.FullName) {
                        Write-Error -Message "File not found!"
                        break
                    }
                    $plainBytes = [System.IO.File]::ReadAllBytes($File.FullName)
                    $outPath = $File.FullName + ".aes"
                }

                $encryptor = $aesManaged.CreateEncryptor()
                $encryptedBytes = $encryptor.TransformFinalBlock($plainBytes, 0, $plainBytes.Length)
                $encryptedBytes = $aesManaged.IV + $encryptedBytes
                $aesManaged.Dispose()

                if ($Text) {return [System.Convert]::ToBase64String($encryptedBytes)}

                if ($Path) {
                    [System.IO.File]::WriteAllBytes($outPath, $encryptedBytes)
                    (Get-Item $outPath).LastWriteTime = $File.LastWriteTime
                    return "File encrypted to $outPath"
                }
            }

            'Decrypt' {
                if ($Text) {$cipherBytes = [System.Convert]::FromBase64String($Text)}

                if ($Path) {
                    $File = Get-Item -Path $Path -ErrorAction SilentlyContinue
                    if (!$File.FullName) {
                        Write-Error -Message "File not found!"
                        break
                    }
                    $cipherBytes = [System.IO.File]::ReadAllBytes($File.FullName)
                    $outPath = $File.FullName -replace ".aes"
                }

                $aesManaged.IV = $cipherBytes[0..15]
                $decryptor = $aesManaged.CreateDecryptor()
                $decryptedBytes = $decryptor.TransformFinalBlock($cipherBytes, 16, $cipherBytes.Length - 16)
                $aesManaged.Dispose()

                if ($Text) {return [System.Text.Encoding]::UTF8.GetString($decryptedBytes).Trim([char]0)}

                if ($Path) {
                    [System.IO.File]::WriteAllBytes($outPath, $decryptedBytes)
                    (Get-Item $outPath).LastWriteTime = $File.LastWriteTime
                    return "File decrypted to $outPath"
                }
            }
        }
    }

    End {
        $shaManaged.Dispose()
        $aesManaged.Dispose()
        }
    }
```

**We can use any previously shown file transfer methods to get this file onto a target host**.
After the script has been transferred, **it only needs to be imported as a module, as shown below.**

### Import Module Invoke-AESEncryption.ps1

```
PS C:\htb> Import-Module .\Invoke-AESEncryption.ps1
```

After the script is imported, it can encrypt strings or files, as shown in the following examples. This command creates an encrypted file with the same name as the encrypted file but with the extension ".aes."

Using very strong and unique passwords for encryption for every company where a penetration test is performed is essential. This is to prevent sensitive files and information from being decrypted using one single password that may have been leaked and cracked by a third party.

**Linux**

OpenSSL is frequently included in Linux distributions, with sysadmins using it to generate security certificates, among other tasks.
**OpenSSL can be used to send files "nc style" to encrypt files.**

To encrypt a file using **openssl** we can select different ciphers, see OpenSSL man page. Let's use **-aes256** as an example. We can also override the default iterations counts with the option **-iter 100000** and add the option **-pbkdf2** to use the **Password-Based Key Derivation Function 2** algorithm. When we hit enter, we'll need to provide a password.

### Encrypting /etc/passwd with openssl

```
openssl enc -aes256 -iter 100000 -pbkdf2 -in /etc/passwd -out passwd.enc

    enter aes-256-cbc encryption password:
    Verifying - enter aes-256-cbc encryption password:
```

Remember to use a strong and unique password to avoid brute-force cracking attacks should an unauthorized party obtain the file. To decrypt the file, we can use the following command:

### Decrypt passwd.enc with openssl

```
$ openssl enc -d -aes256 -iter 100000 -pbkdf2 -in passwd.enc -out passwd

    enter aes-256-cbc decryption password:
```

We can use any of the previous methods to transfer this file, but it's recommended to use a secure transport method such as HTTPS, SFTP, or SSH. As always, practice the examples in this section against target hosts in this or other modules and reproduce what you can (such as the openssl examples using the Pwnbox. The following section will cover different ways to transfer files over HTTP and HTTPS.

.