Anonymously Enumerating Azure File Resources

written by Karl Fosaaen | July 17, 2018

In recent years, we have seen Microsoft Azure services gathering a larger market share in the cloud space. While they're not seeing quite the adoption that AWS has, we are running into more clients that are using Microsoft Azure services for their operations. If everything is configured correctly, this can be totally fine, but it's pretty rare for an environment to be perfectly secured (and that's why we do security testing). Given the increase in Azure usage, we wanted to dive deeper into automating our standard Azure testing tasks, including the enumeration of publicly available files. In this blog, we'll go over the different types of Azure file stores and how we can potentially enumerate and access publicly available "Blob" files.

Storage Accounts

One of the issues that we've seen within Azure environments is publicly exposing files through storage accounts. These issues are pretty similar to the issues that have come up around public S3 buckets (A good primer here). "Storage Accounts" are Microsoft's way of handling data storage in Azure. Each storage account has a unique subdomain assigned to it in the core.windows.net domain.

For example, if I create the netspiazure storage account, I would have netspiazure.core.windows.net assigned to the account.

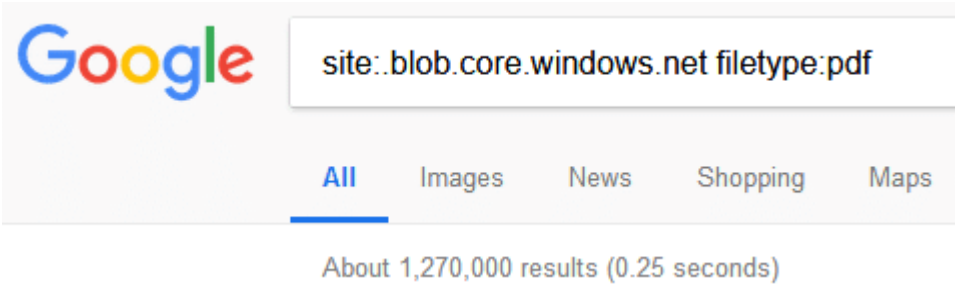
This subdomain structure also extends out to the different

file types within the storage accounts

- Blobs – netspiazure.blob.core.windows.net
- File Services – netspiazure.file.core.windows.net
- Data Tables – netspiazure.table.core.windows.net
- Queues – netspiazure.queue.core.windows.net

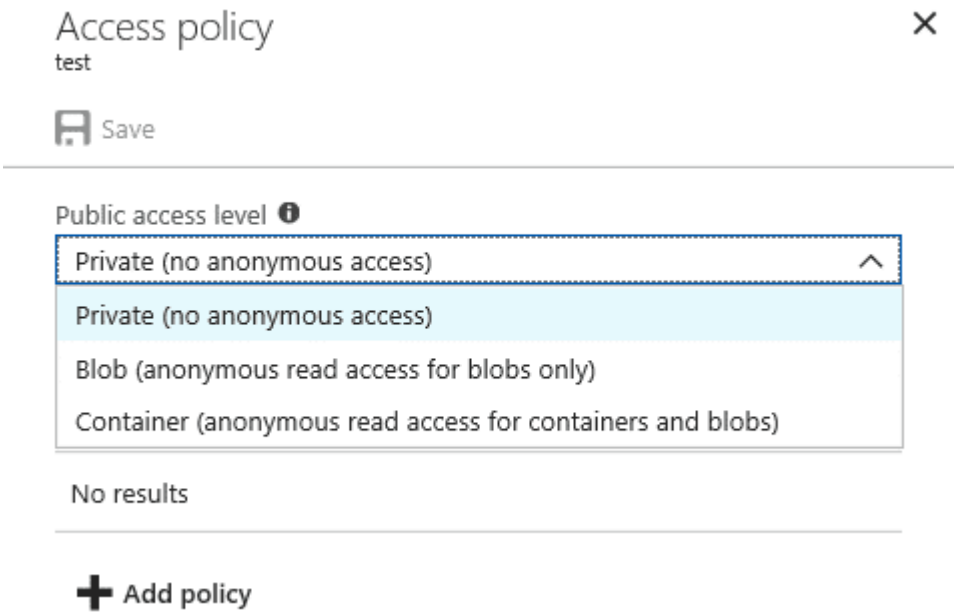
Blobs

For the purpose of this blog, we're just going to focus on the "Blobs", but the other data types are also interesting. Blobs are Microsoft's unstructured data storage objects. Most frequently, we're seeing them used for serving static public data, but we have found blobs being used to store sensitive information (config files, database backups, credentials). Given that Google is indexing about 1.2 million PDFs from the Azure "blob.core.windows.net" subdomain, I think there's pretty decent surface area here.



Permissions

The blobs themselves are stored within "Containers", which are basically folders. Containers have access policies assigned to them, which determines the level of public access that is available for the files.



If a container has a “Container” public access policy, then anonymous users can list and read any of the files that are in the container. The “Blob” public access policy still allows anonymous users to read files, but they can’t list the container files. “Blob” permissions also prevent the basic confirmation of container names via the Azure Blob Service Rest APIs.

Automation

Given the number of Azure environments that we’ve been running into, I wanted to automate our process for enumerating publicly available blob files. I’m partial to PowerShell, but this script could potentially be ported to other languages.

Code for the script can be found on NetSPI’s GitHub – https://github.com/NetSPI/MicroBurst

At the core of the script, we’re doing DNS lookups for blob.core.windows.net subdomains to enumerate valid storage accounts and then brute-force container names using the Azure Blob Service REST APIs. Additionally, the Bing Search API can be used within the tool to find additional storage accounts and containers that are already publicly indexed. Once a valid container name is identified, we use the Azure Blob APIs again

to see if the container allows us to list files via the "Container" public access policy. To come up with valid storage account names, we start with a base name (netspi) and either prepend or postpend additional words (dev, test, qa, etc.) that come from a permutations wordlist file.

The general idea for this script along with parts of the permutations list comes from a similar tool for AWS S3 buckets – inSp3ctor

Invoke-EnumerateAzureBlobs Usage

The script has five parameters:

- Base – The base name that you want to run permutations on (netspi)
- Permutations – The file containing words to use with the base to find the storage accounts (netspidev, testnetspi, etc.)
- Folders – The file containing potential folder names that you want to brute force (files, docs, etc.)
- BingAPIKey – The Bing API Key to use for finding additional public files
- OutputFile – The file to write the output to

Example Output:

```
PS C:> Invoke-EnumerateAzureBlobs -Base secure -BingAPIKey
1234567890123456789876543210123
Found Storage Account -  secure.blob.core.windows.net
Found Storage Account -  testsecure.blob.core.windows.net
Found Storage Account -  securetest.blob.core.windows.net
Found Storage Account -  securedata.blob.core.windows.net
Found Storage Account -  securefiles.blob.core.windows.net
Found Storage Account -
securefilestorage.blob.core.windows.net
Found Storage Account -
securestorageaccount.blob.core.windows.net
Found Storage Account -  securesql.blob.core.windows.net
Found Storage Account -  hrsecure.blob.core.windows.net
```

```
Found Storage Account -  secureit.blob.core.windows.net
Found Storage Account -  secureimages.blob.core.windows.net
Found Storage Account -  securestorage.blob.core.windows.net
Bing Found Storage Account - notrealstorage.blob.core.windows.net
Found Container - hrsecure.blob.core.windows.net/NETSPItest
Public File Available: https://hrsecure.blob.core.windows.net/NETSPItest/SuperSecretFile.txt
Public File Available: https://hrsecure.blob.core.windows.net/NETSPItest/TaxReturn.pdf
Found Container - secureimages.blob.core.windows.net/NETSPItest123
Empty Public Container Available: https://secureimages.blob.core.windows.net/NETSPItest123?restype=container&comp=list
```

By default, both the “Permutations” and “Folders” parameters are set to the permutations.txt file that comes with the script. You can increase your chances for finding files by adding any client/environment specific terms to that file.

Adding in a Bing API Key will also help find additional storage accounts that contain your base word. If you don’t already have a Bing API Key set up, navigate to the “Cognitive Services” section of the Azure Portal and create a new “Bing Search v7” instance for your account. There’s a free pricing tier that will get you up to 3,000 calls per month, which will hopefully be sufficient.

If you are using the Bing API Key, you will be prompted with an out-gridview selection window to select any storage accounts that you want to look at. There are a few publicly indexed Azure storage accounts that seem to hit on most company names. These accounts seem to be indexing documents for or data on multiple companies, so they have a tendency to frequently show up in your Bing results. Several of these accounts also have public file listing, so that may give you a large list of public files that you don’t care about. Either

way, I’ve had pretty good luck with the script so far, and hopefully you do too.

If you have any issues with the script, feel free to leave a comment or pull request on the GitHub page.