A Beginners Guide to Gathering Azure Passwords

It has been a while since the initial release (August 2018) of the Get-AzurePasswords module within MicroBurst, so I figured it was time to do an overview post that explains how to use each option within the tool. Since each targeted service in the script has a different way of getting credentials, I want users of the tool to really understand how things are working.

For those that just want to jump to information on specific services, here's links to each section:

- Key Vaults
- App Services
- Automation Accounts
- Storage Accounts
- Azure Container Registries

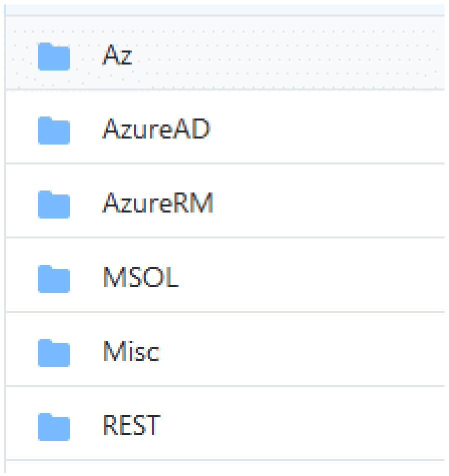
Additionally, we've renamed the function to Get-AzPasswords to match the PowerShell modules that we're using in the function.

AzureRM Versus Az

As of March 19, 2020, we pushed some changes to MicroBurst to switch the cmdlets over to the Az PowerShell modules (from AzureRM). This was a much needed switch, as the AzureRM modules are being replace by the Az modules.

> New updates for MicroBurst! I've ported the previous scripts to the Az module cmdlets (from AzureRM/Azure) and moved them to their own folder. All of the previous scripts will still be there, just under different folders (organized by module dependency). https://t.co/h3EwMt9JWZ
>
> — Karl (@kfosaaen) March 19, 2020

Along with these updates I also wanted to make some additions to the Get-AzurePasswords functionality. Since we reorganized all of the code in MicroBurst to match up with the related supporting modules (Az, AzureAD, MSOL, etc.), we thought it was important to separate out function names based off of the modules the script was using.



Going forward, Get-AzurePasswords will still live in the AzureRM folder in MicroBurst, but it will not be updated with any new functionality. Going forward, I highly recommend that everyone switches over to the newly renamed version "Get-AzPasswords" in the Az folder in MicroBurst.

Important Script Usage Note – Some of these functions can make minor temporary changes to an Azure subscription (see Automation Accounts). If you Ctrl+C during the script execution, you may end up with unintended files or changes in your subscription.

I'll cover each of these concerns in the sections below, but have patience when running these functions. I know Azure can have its slow moments, so (in most cases) just give the script a moment to catch up and everything should be good. I haven't been keeping track, but I believe I've lost several months of time waiting for automation runbook jobs to complete.

Function Usage

For each service section, I've noted the script flag that you can use to toggle ("-Keys Y" versus "-Keys N") the collection of that specific service. Running the script with no flags will gather credentials from all services.

Step 1. Import the MicroBurst Module

```
PS C:\MicroBurst> Import-Module .\MicroBurst.psm1
Imported Az MicroBurst functions
Imported AzureAD MicroBurst functions
Imported MSOnline MicroBurst functions
Imported Misc MicroBurst functions
Imported Azure REST API MicroBurst functions
```

Step 2. Authenticate to the Azure Tenant

```
PS C:\MicroBurst> Login-AzAccount

Account            SubscriptionName  TenantId
Environment
-------            ----------------  --------                                   ------
-----
```

```
test@example.com  TestSubscription  4712345a-6543-b5s4-a2b2-e01234567895
AzureCloud
```

Step 3. Gather Passwords

```
PS C:\MicroBurst> Get-AzPasswords -Verbose
VERBOSE: Logged In as test@example.com
VERBOSE: Getting List of Key Vaults...
VERBOSE:  Exporting items from testingKeys
VERBOSE:   Getting Key value for the testKey Key
VERBOSE:   Getting Secret value for the TestKey Secret
VERBOSE: Getting List of Azure App Services...
VERBOSE:  Profile available for microburst-application
VERBOSE: Getting List of Azure Container Registries...
VERBOSE: Getting List of Storage Accounts...
VERBOSE:  Getting the Storage Account keys for the teststorage account
VERBOSE: Getting List of Azure Automation Accounts...
VERBOSE:  Getting the RunAs certificate for autoadmin using the
XZvOYzsuBiGbfqe.ps1 Runbook
VERBOSE:   Waiting for the automation job to complete
VERBOSE:    Run AuthenticateAs-autoadmin-AzureRunAsConnection.ps1 (as a local
admin) to import the cert and login as the Automation Connection account
VERBOSE:   Removing XZvOYzsuBiGbfqe runbook from autoadmin Automation Account
VERBOSE:  Getting cleartext credentials for the autoadmin Automation Account
VERBOSE:   Getting cleartext credentials for test using the dOFtWgEXIQLlRfv.ps1
Runbook
VERBOSE:    Waiting for the automation job to complete
VERBOSE:    Removing dOFtWgEXIQLlRfv runbook from autoadmin Automation Account
VERBOSE: Password Dumping Activities Have Completed
```

*Running this will “Out-GridView” prompt you to select the subscription(s) to gather credentials from.

For easier output management, I’d recommend piping the output to either “Out-GridView” or “Export-CSV”.

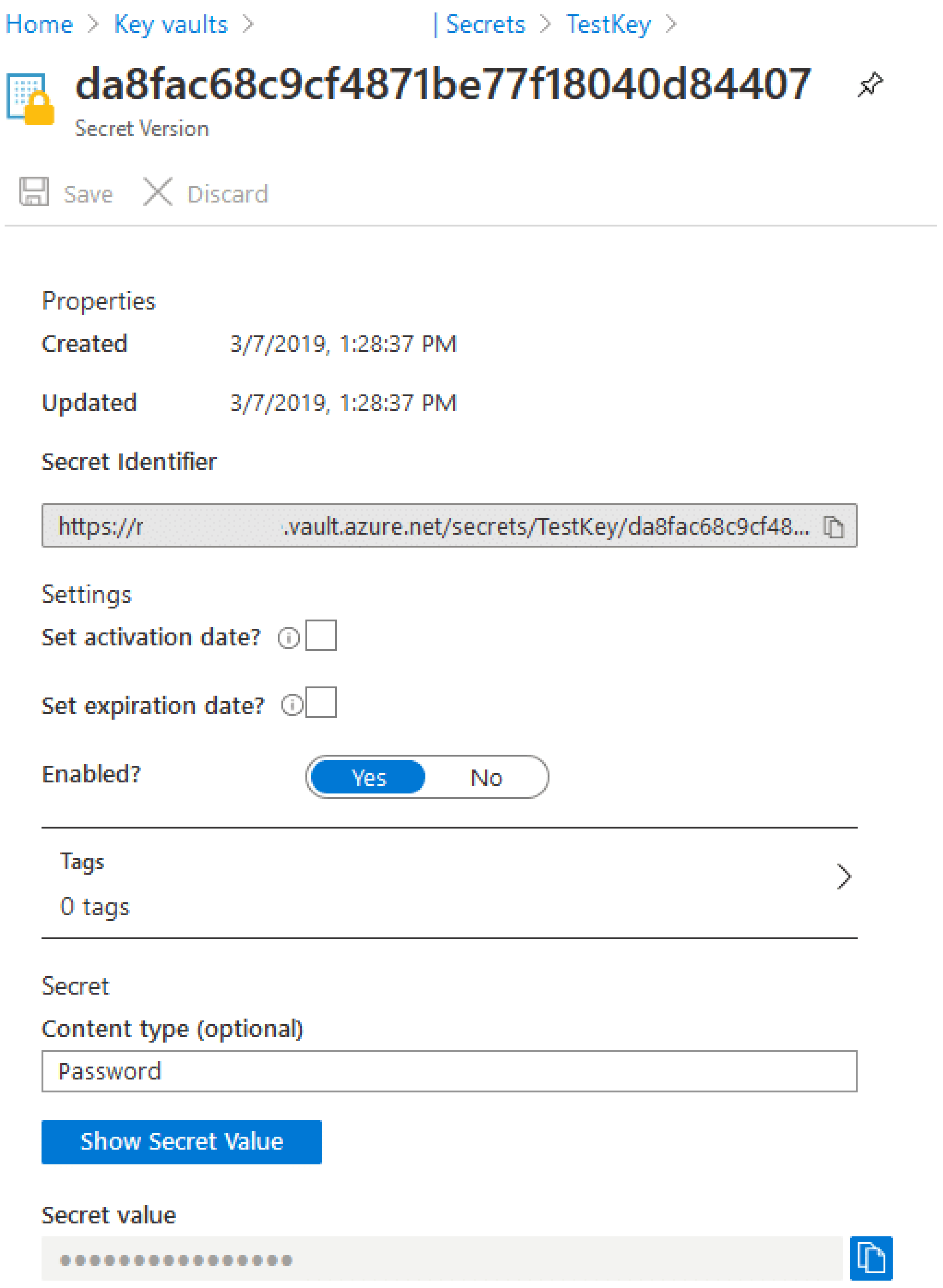
With that housekeeping out of the way, let’s dive into the credentials that we’re able to gather with Get-AzPasswords.

Key Vaults (-Keys Y)

Azure Key Vaults are Microsoft’s solution for storing sensitive data (Keys, Passwords/Secrets, Certs) in the Azure cloud. Inherently, Key Vaults are great sources for finding credential data. If you have a user with the correct rights, you should be able to read data out of the key stores.

Vault access is controlled by the Access Policies in each vault, but any users with Contributor rights are able to give themselves access to a Key Vault. Get-AzPasswords will not modify any Key Vault Access Policies, but you could give your account read permissions on the vault if you really needed to read a key.

An example Key Vault Secret:


Home > Key vaults > | Secrets > TestKey >

da8fac68c9cf4871be77f18040d84407

Secret Version

Save Discard

Properties

Created 3/7/2019, 1:28:37 PM

Updated 3/7/2019, 1:28:37 PM

Secret Identifier

https://r .vault.azure.net/secrets/TestKey/da8fac68c9cf48...

Settings

Set activation date? ☐

Set expiration date? ☐

Enabled? [✓ Yes] No

Tags

0 tags

Secret

Content type (optional)

Password

Show Secret Value

Secret value

●●●●●●●●●●●●●●●●

Get-AzPasswords will export all of the secrets in cleartext, along with any certificates. You also have the option to save the certificate files locally with the “-ExportCerts Y” flag.

Sample Output:

```
Type : Key
Name : DiskKey
Username : N/A
```

```
Value :
{"kid":"https://notArealVault.vault.azure.net/keys/DiskKey/63abcdefghijklmnop39
","kty
":"RSA","key_ops":["sign","verify","wrapKey","unwrapKey","encrypt","decrypt"],"
n":"v[REDACTED]w","e":"AQAB"}
PublishURL : N/A
Created : 5/19/2020 5:20:12 PM
Updated : 5/19/2020 5:20:12 PM
Enabled : True
Content Type : N/A
Vault : notArealVault
Subscription : NotARealSubscription

Type : Secret
Name : TestKey
Username : N/A
Value : Karl'sPassword
PublishURL : N/A
Created : 3/7/2019 9:28:37 PM
Updated : 3/7/2019 9:28:37 PM
Enabled : True
Content Type : Password
Vault : notArealVault
Subscription : NotARealSubscription
```

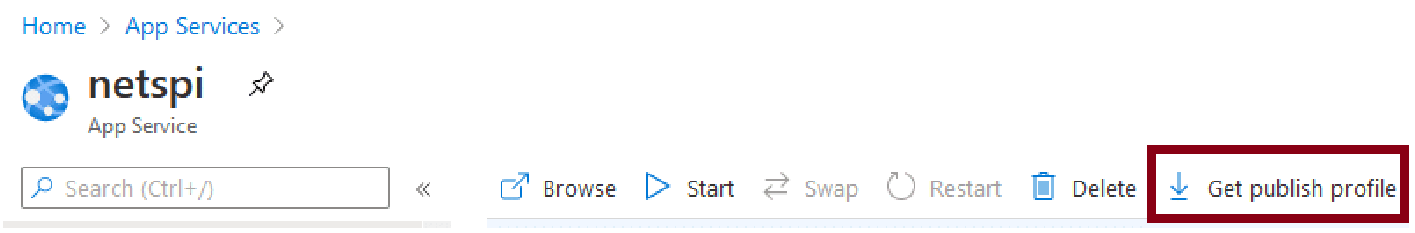
Finally, access to the Key Vaults may be restricted by network, so you may need to run this from an Azure VM on the subscription, or from an IP in the approved "Private endpoint and selected networks" list. These settings can be found under the Networking tab in the Azure portal.

Alternatively, you may need to use an automation account "Run As" account to request the keys. Steps to complete that process are outlined here.

App Services (-AppServices Y)

Azure App Services are Microsoft's option for rapid application deployment. Applications can be spun up quickly using app services and the configurations (passwords) are pushed to the applications via the App Services profiles.

In the portal, the App Services deployment passwords are typically found in the "Publish Profile" link that can be found in the top navigation bar within the App Services section. Any user with contributor rights to the application should be able to access this profile.



These publish profiles will contain Web and FTP credentials that can be used to get access to the App Service’s files. In addition to that, any stored connection strings should also be available in the file. All available profile credentials are all parsed by Get-AzPasswords, so it’s easy to gather credentials for multiple App Services applications at once.

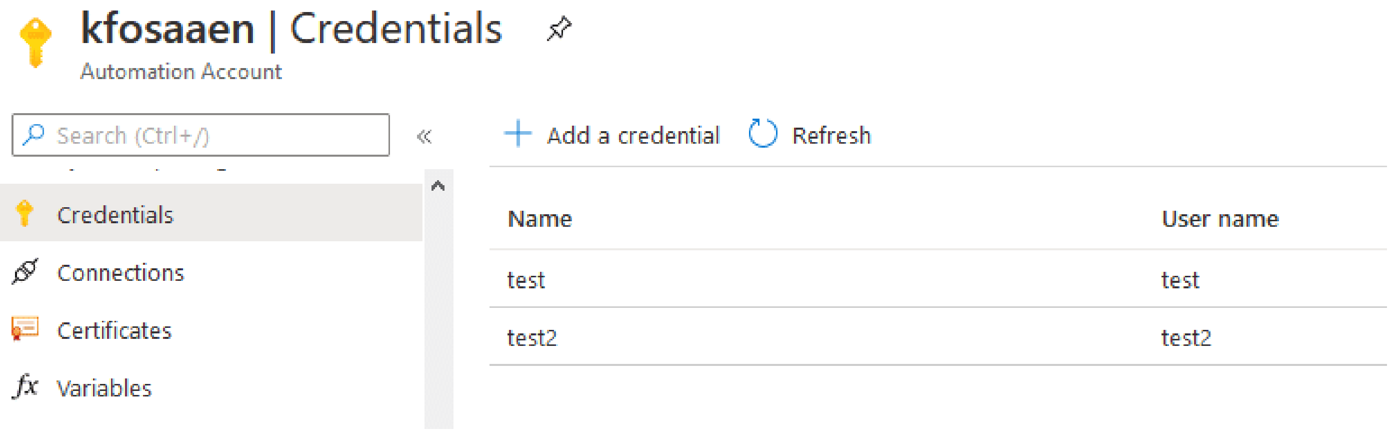
Sample Output:

```
Type         : AppServiceConfig
Name         : appServicesApplication - Web Deploy
Username     : $appServicesApplication
Value        : dS[REDACTED]jM
PublishURL   : appServicesApplication.scm.azurewebsites.net:443
Created      : N/A
Updated      : N/A
Enabled      : N/A
Content Type : Password
Vault        : N/A
Subscription : NotARealSubscription

Type         : AppServiceConfig
Name         : appServicesApplication - FTP
Username     : appServicesApplication\$appServicesApplication
Value        : dS[REDACTED]jM
PublishURL   :
ftp://appServicesApplication.ftp.azurewebsites.windows.net/site/wwwroot
Created      : N/A
Updated      : N/A
Enabled      : N/A
Content Type : Password
Vault        : N/A
Subscription : NotARealSubscription

Type : AppServiceConfig
Name : appServicesApplication-Custom-ConnectionString
Username : N/A
Value :
metadata=res://*/Models.appServicesApplication.csdl|res://*/Models.appServicesA
pplication.ssdl|res://*/Models.appServicesApplication.msl;provider=System.Data.
SqlClient;provider connection string="Data
Source=abcde.database.windows.net;Initial Catalog=app_Enterprise_Prod;Persist
Security Info=True;User ID=psqladmin;Password=somepassword9"
PublishURL : N/A
Created : N/A
Updated : N/A
Enabled : N/A
Content Type : ConnectionString
Vault : N/A
Subscription : NotARealSubscription
```

Potential next steps for App Services have been outlined in another NetSPI blog post here.

Automation Accounts (-AutomationAccounts Y)

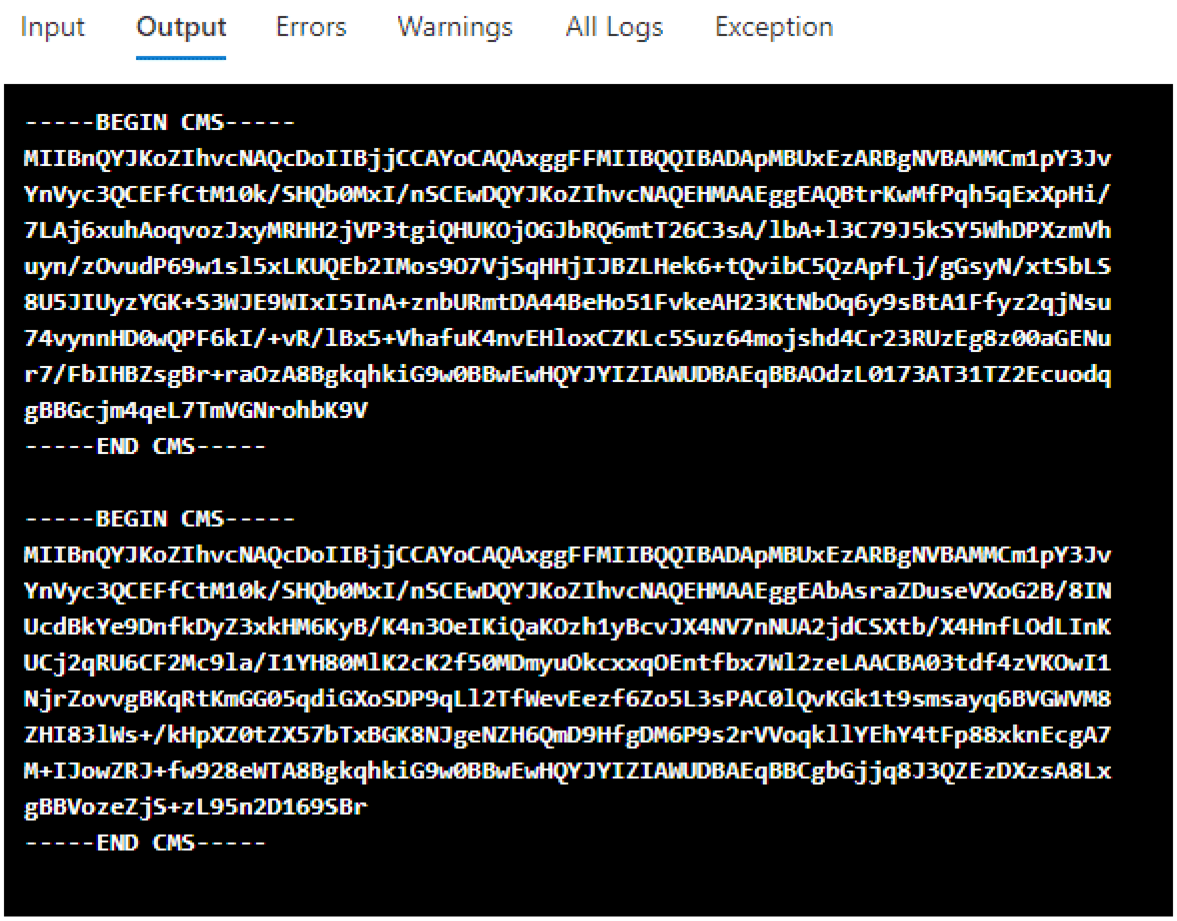
Automation Accounts are one of the ways that you can automate jobs and routine tasks within Azure. These tasks (Runbooks) are frequently run with stored credentials, or with the service account (Run As “Connections”) tied to the Automation Account.



Name	User name
test	test
test2	test2

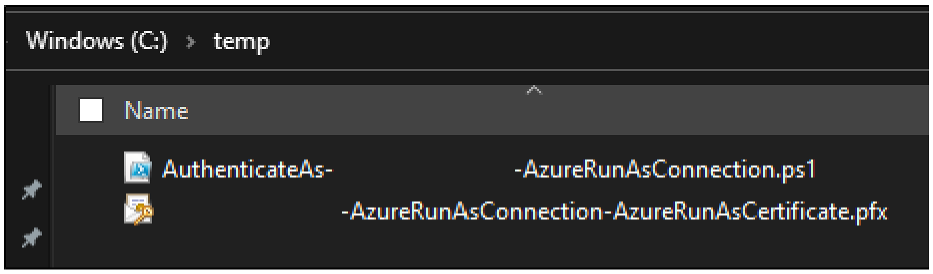
Both of these credential types can be returned with Get-AzPasswords and can potentially allow for privilege escalation. In order to gather these credentials from the Automation Accounts, we need to create new runbooks that will cast the credentials out to variables that are then printed to the runbook job output.

To protect these credentials in the output, we’ve implemented an encryption scheme in Get-AzPasswords that encrypts the job output.



```
-----BEGIN CMS-----
MIIBnQYJKoZIhvcNAQcDoIIBjjCCAYoCAQAxggFFMIIBQQIBADApMBUxEzARBgNVBAMMCm1pY3Jv
YnVyc3QCEFfCtM10k/SHQb0MxI/nSCEwDQYJKoZIhvcNAQEHMAAEggEAQBtrKwMfPqh5qExXpHi/
7LAj6xuhAoqvozJxyMRHH2jVP3tgiQHUKOjOGJbRQ6mtT26C3sA/lbA+l3C79J5kSY5WhDPXzmVh
uyn/zOvudP69w1sl5xLKUQEb2IMos9O7VjSqHHjIJBZLHek6+tQvibC5QzApfLj/gGsyN/xtSbLS
8U5JIUyzYGK+S3WJE9WIxI5InA+znbURmtDA44BeHo51FvkeAH23KtNbOq6y9sBtA1Ffyz2qjNsu
74vynnHD0wQPF6kI/+vR/lBx5+VhafuK4nvEHloxCZKLc5Suz64mojshd4Cr23RUzEg8z00aGENu
r7/FbIHBZsgBr+raOzA8BgkqhkiG9w0BBwEwHQYJYIZIAWUDBAEqBBAOdzL0173AT31TZ2Ecuodq
gBBGcjm4qeL7TmVGNrohbK9V
-----END CMS-----

-----BEGIN CMS-----
MIIBnQYJKoZIhvcNAQcDoIIBjjCCAYoCAQAxggFFMIIBQQIBADApMBUxEzARBgNVBAMMCm1pY3Jv
YnVyc3QCEFfCtM10k/SHQb0MxI/nSCEwDQYJKoZIhvcNAQEHMAAEggEAbAsraZDuseVXoG2B/8IN
UcdBkYe9DnfkDyZ3xkHM6KyB/K4n3OeIKiQaKOzh1yBcvJX4NV7nNUA2jdCSXtb/X4HnfLOdLInK
UCj2qRU6CF2Mc9la/I1YH80MlK2cK2f50MDmyuOkcxxqOEntfbx7Wl2zeLAACBA03tdf4zVKOwI1
NjrZovvgBKqRtKmGG05qdiGXoSDP9qLl2TfWevEezf6Zo5L3sPAC0lQvKGk1t9smsayq6BVGWVM8
ZHI83lWs+/kHpXZ0tZX57bTxBGK8NJgeNZH6QmD9HfgDM6P9s2rVVoqkllYEhY4tFp88xknEcgA7
M+IJowZRJ+fw928eWTA8BgkqhkiG9w0BBwEwHQYJYIZIAWUDBAEqBBCgbGjjq8J3QZEzDXzsA8Lx
gBBVozeZjS+zL95n2D169SBr
-----END CMS-----
```

The Run As certificates (covered in another blog) can then be used to authenticate (run the AuthenticateAs PS1 script) from your testing system as the stored Run As connection.



```
AuthenticateAs-[redacted]-AzureRunAsConnection.ps1
[redacted]-AzureRunAsConnection-AzureRunAsCertificate.pfx
```

Any stored credentials may have a variety of uses, but I've frequently seen domain accounts being stored here, so that can lead to some interesting lateral movement options.

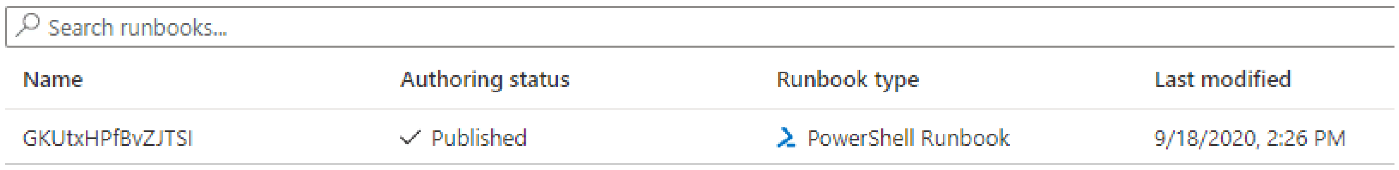
Sample Output:

```
Type : Azure Automation Account
Name : kfosaaen
Username : test
Value : testPassword
PublishURL : N/A
Created : N/A
Updated : N/A
Enabled : N/A
```

```
Content Type : Password
Vault : N/A
Subscription : NotARealSubscription
```

As a secondary note here, you can also request bearer tokens for the Run As automation accounts from a custom runbook. I cover the process in this blog post, but I think it's worth noting here, since it's not included in Get-AzPasswords, but it is an additional way to get a credential from an Automation Account.

And one final note on gathering credentials from Automation Accounts. It was noted above, but sometimes Azure Automation Accounts can be slow to respond. If you're having issues getting a runbook to run and cancel the function execution before it completes, you will need to manually go in and clean up the runbooks that were created as part of the function execution.

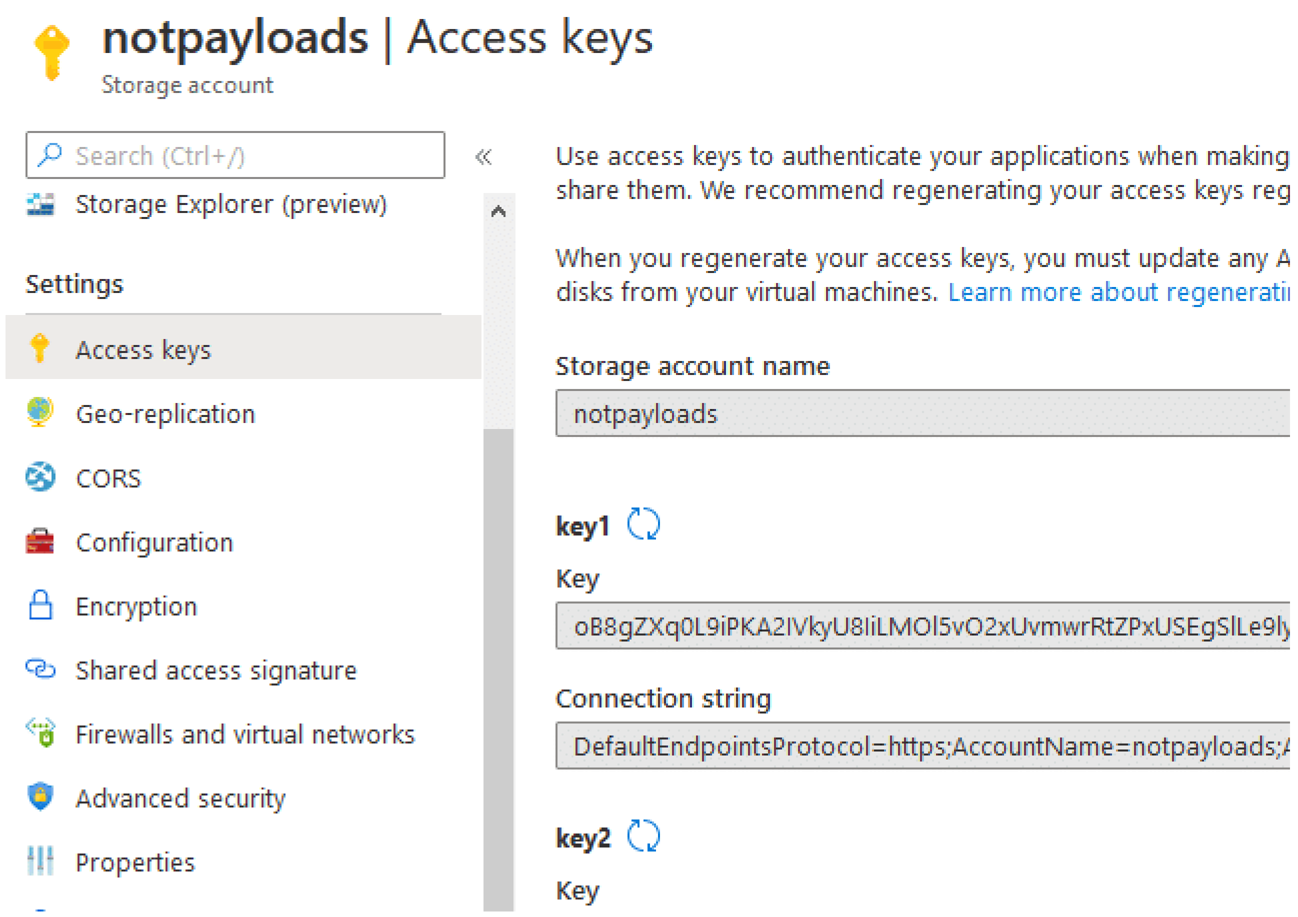

Search runbooks...

Name	Authoring status	Runbook type	Last modified
GKUtxHPfBvZJTSI	✓ Published	PowerShell Runbook	9/18/2020, 2:26 PM

These will always be named with a 15-character random string of letters (IE: lwVSNvWYpPXCcDd). You will also have local files in your execution directory to clean up as well, and they will have the same name as the ones that were uploaded for execution.

Storage Account Keys (-StorageAccounts Y)

Storage Accounts are the multipurpose (Public/Private files, tables, queues) service for storing data in an Azure subscription. This section is pretty simple compared to the previous ones, but gathering the account keys is an easy way to maintain persistence in a sensitive data store. These keys can be used with the Azure storage explorer application to remotely mount storage accounts.


notpayloads | Access keys

Storage account

Use access keys to authenticate your applications when making
share them. We recommend regenerating your access keys reg

When you regenerate your access keys, you must update any A
disks from your virtual machines. Learn more about regenerati

Storage account name

notpayloads

key1

Key

oB8gZXq0L9iPKA2IVkyU8IiLMOl5vO2xUvmwrRtZPxUSEgSlLe9ly

Connection string

DefaultEndpointsProtocol=https;AccountName=notpayloads;A

key2

Key

These access keys can easily be cycled, but if you're looking for persistence in a Storage Account, these would be your best bet. Additionally, if you're modifying Cloud Shell files for escalation/persistence, I'd recommend holding on to a copy of these keys for any Cloud Shell storage accounts.

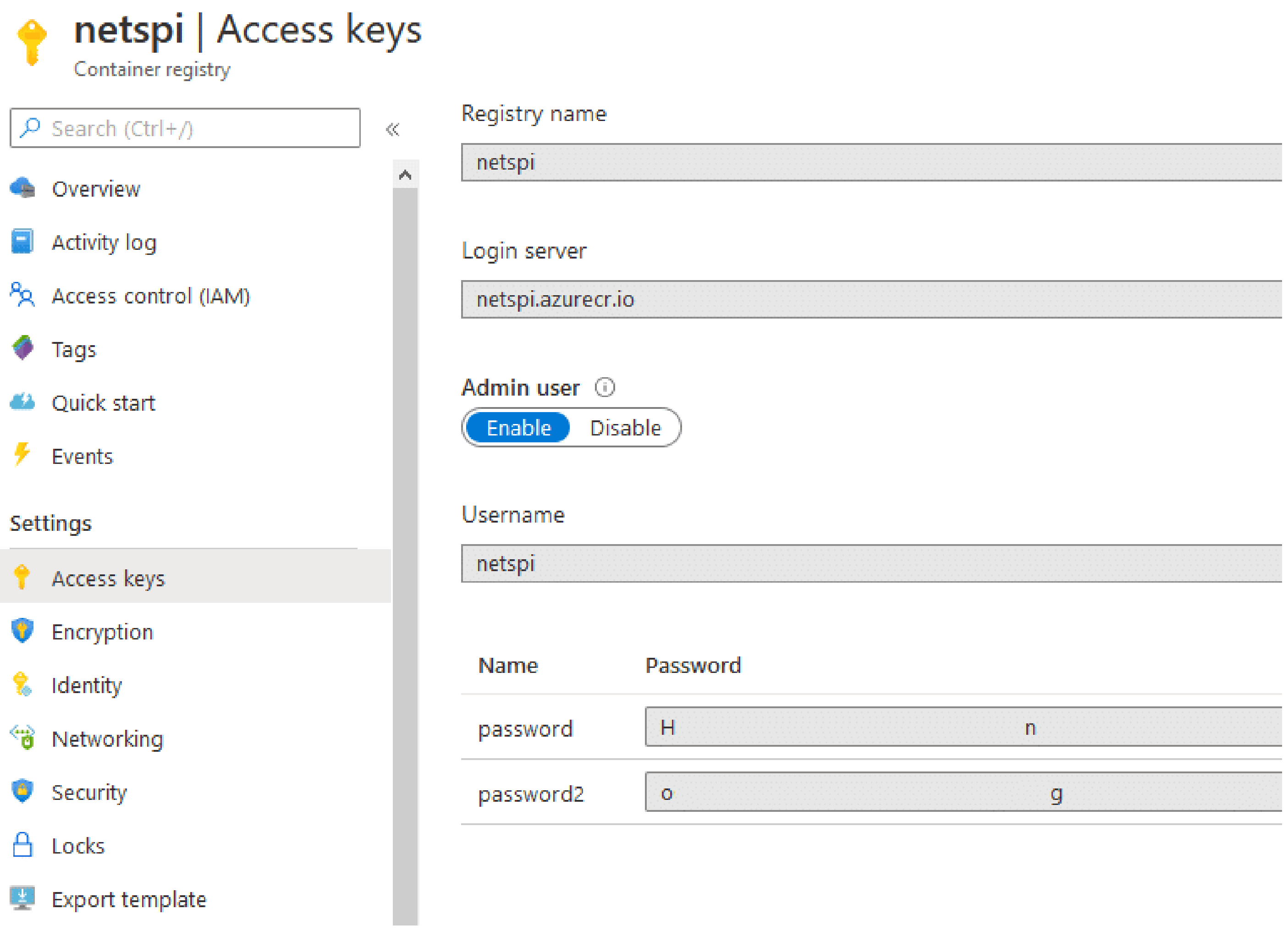
Sample Output:

```
Type         : Storage Account
Name         : notArealStorageAccount
Username     : key1
Value        : W84[REDACTED]==
PublishURL   : N/A
Created      : N/A
Updated      : N/A
Enabled      : N/A
Content Type : Key
Vault        : N/A
Subscription : NotARealSubscription
```

Azure Container Registries (-ACR Y)

Azure Container Registries are used in Azure to manage Docker images for use within Azure Kubernetes Service (AKS) or as container instances (either in Azure or elsewhere). More often than not, we will find keys and credentials stored in these container images.

In order to authenticate to the repositories, you can either use the AZ CLI with AzureAD credentials, or there can be an “Admin user”. If you have AzureAD user rights to pull the admin password for a container registry, you should already have rights to authenticate to the repositories with the AZ CLI, but if an Admin user is enabled for the registry, this user could then be used for persistence when you lose access to the initial AzureAD user.



Registry name: netspi

Login server: netspi.azurecr.io

Admin user: [✓ Enable] Disable

Username: netspi

Name	Password
password	H [redacted] n
password2	o [redacted] g

Fun Fact – Any AzureAD user with “Reader” permissions on the Container Registry is able to connect to the repositories and pull images down with Docker.

For more information on using these credentials, here’s another NetSPI blog.

Conclusion

There was a lot of ground to cover here, but hopefully this does a good job of explaining all of the functionality available in Get-AzPasswords. It’s worth noting that most of this functionality relies on your AzureAD user having Contributor IAM rights on the applicable service. While some may argue that Contributor access is equivalent to admin in a subscription, I would argue that most subscriptions primarily consist of Contributor users.

If there are any sources for Azure passwords that you would like to see added to Get-AzPasswords, feel free to make a pull request on the MicroBurst GitHub repository.