Get-AzurePasswords: A Tool for Dumping Credentials from Azure Subscriptions

During different types of assessments (web app, network, cloud), we will run into situations where we obtain domain credentials that can be used to log into Azure subscriptions. Most commonly, we will externally guess credentials for a privileged domain user, but we've also seen excessive permissions in web applications that use Azure AD for authentication.

If we're really lucky, we'll have access to a user that has rights (typically Owner or Contributor) to access sensitive information in the subscription. If we have privileged access, there are three specific areas that we typically focus on for gathering credentials:

- Key Vaults
- App Services Configurations
- Automation Accounts

There are other places that application/domain credentials could be hiding (See Storage Account files), but these are the first couple of spots that we want to check for credentials.

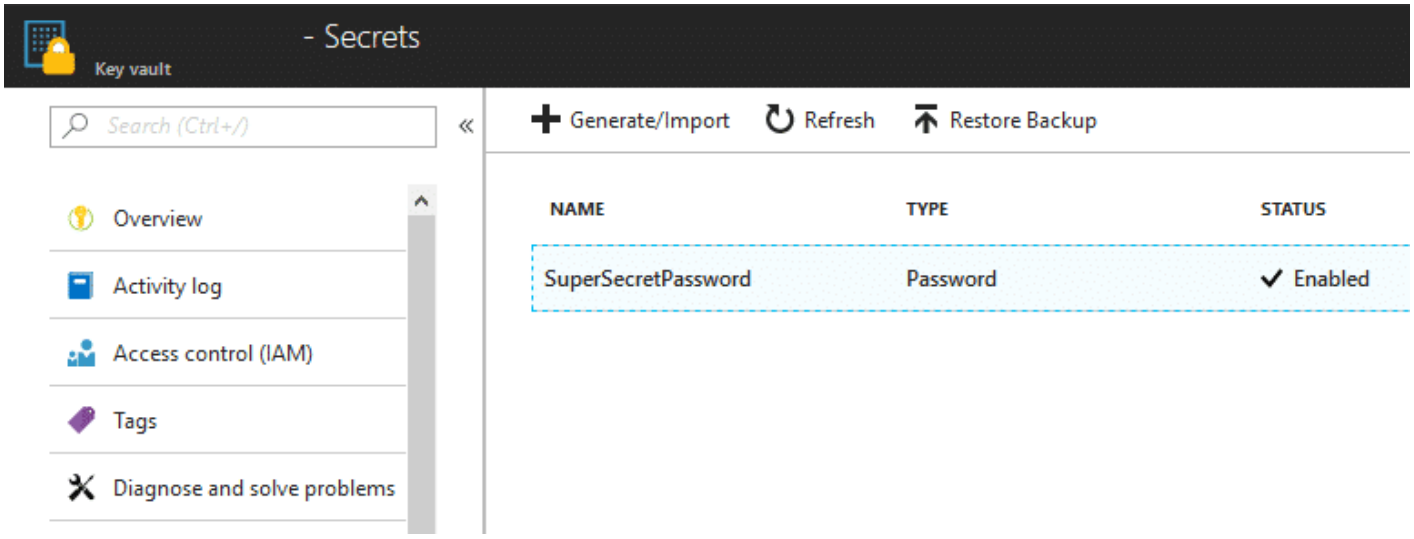
In this post, we'll go over the key areas where credentials are commonly found and the usage of a PowerShell script (a part of MicroBurst) that I put together to automate the process of gathering credentials from an Azure environment.

Key Vaults

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.

Here's a quick overview of setting permissions for Key Vaults – https://docs.microsoft.com/en-us/azure/key-vault/key-vault-secure-your-key-vault

An example Key Vault Secret:



For dumping Key Vault values, we're using some standard Azure PowerShell commands:

- Get-AzureKeyVaultKey
- Get-AzureKeyVaultSecret

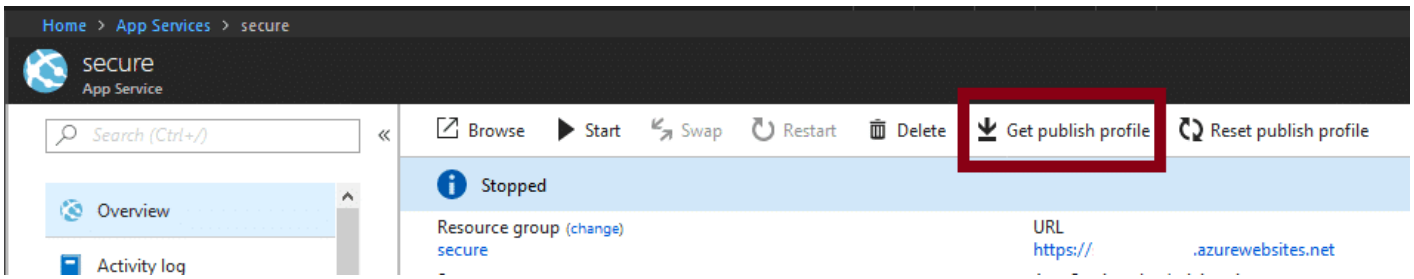
If you're just looking at exporting one or two secrets, these commands can be run individually. But since we're typically trying to access everything that we can in an Azure subscription, we've automated the process in the script. The script will export all of the secrets in cleartext, along with any certificates. You also have the option to save the certificates locally with the -ExportCerts flag.

With access to the keys, secrets, and certificates, you may be able to use them to pivot through systems in the Azure subscription. Additionally, I've seen situations where administrators have stored Azure AD user credentials in the Key Vault.

App Services Configurations

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.



For dumping App Services configurations, we're using the following AzureRM PowerShell commands:

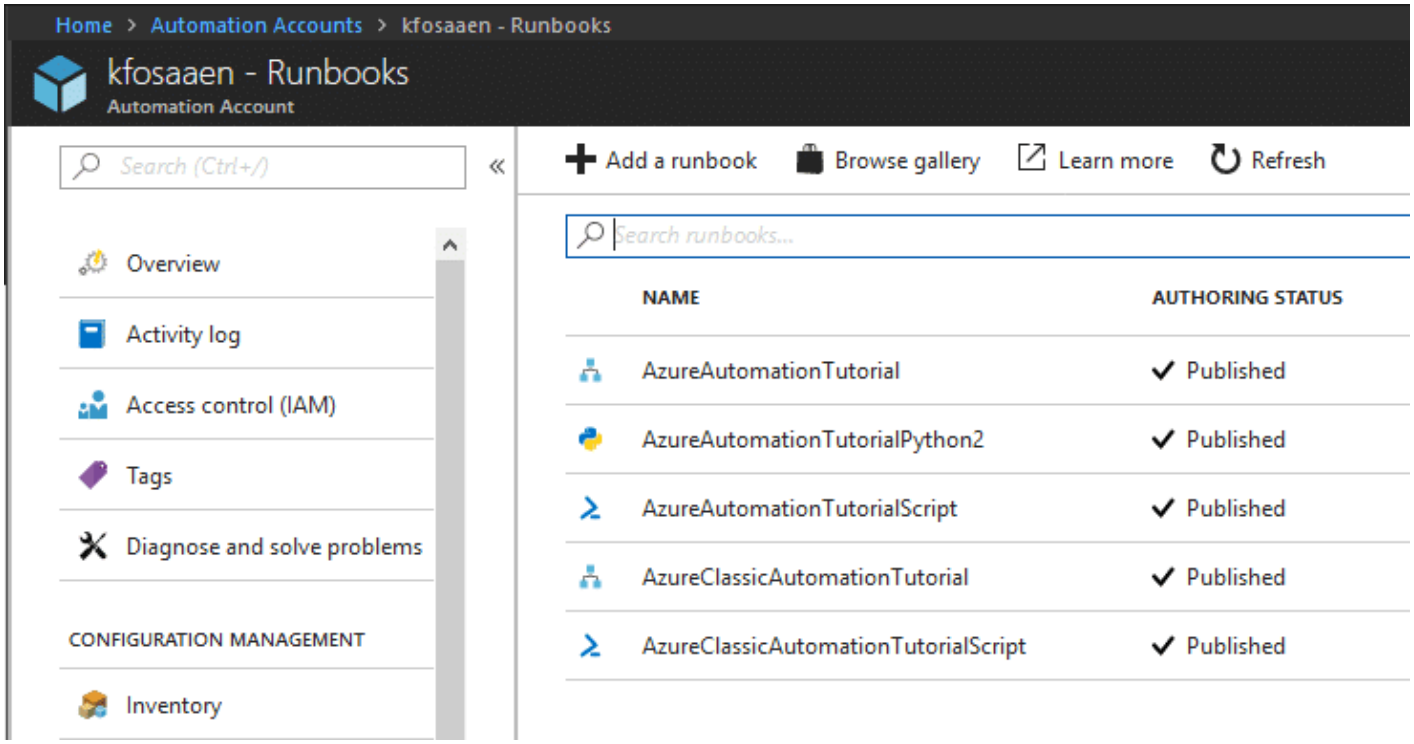
- Get-AzureRmWebApp
- Get-AzureRmResource
- Get-AzureRmWebAppPublishingProfile

Again, if this is just a one-off configuration dump, it's easy to grab the profile from the web portal. But since we're looking to automate this process, we use the commands above to list out the available apps and profiles for each app. Once the publishing profile is collected by the script, it is then parsed and credentials are returned in the final output table.

Potential next steps include uploading a web shell to the App Services web server, or using any parsed connection strings included in the deployment to access the databases. With access to the databases, you could potentially use them as a C2 channel. Check out Scott's post for more information on that.

Automation Accounts

Automation accounts are one of the ways that you can automate tasks on Azure subscriptions. As part of the automation process, Azure allows these accounts to run code in the Azure environment. This code can be PowerShell or Python, and it can also be very handy for pentesting activities.



The automation account credential gathering process is particularly interesting, as we will have to run some PowerShell in Azure to actually get the credentials for the automation accounts. This section of the script will deploy a Runbook as a ps1 file to the Azure environment in order to get access to the credentials.

Basically, the automation script is generated in the tool and includes the automation account name that we're gathering the credentials for.

```
$myCredential = Get-AutomationPSCredential -Name 'ACCOUNT_NAME_HERE'$userName =
```

```
$myCredential.UserName$password =
$myCredential.GetNetworkCredential().Passwordwrite-output "$userName"write-
output "$password"
```

This Microsoft page was a big help in getting this section of the script figured out. Dumping these credentials can take a minute, as the automation script needs to be spooled up and ran on the Azure side.

This method of grabbing Automation Account credentials is not the most OpSec safe, but the script does attempt to clean up after itself by deleting the Runbook. As long as the Runbook is successfully deleted at the end of the run, all that will be left is an entry in the Jobs page.

STATUS	RUNBOOK	RAN ON
✓ Completed	fGntiyCDOxUWeSu	Azure

To help with obscuring these activities, the script generates 15-character job names for each Runbook so it's hard to tell what was actually run. If you want, you can modify the jobName variable in the code to name it something a little more in line with the tutorial names, but the random names help prevent issues with naming conflicts.

	NAME	AUTHORING STATUS	LAST MODIFIED
	AzureAutomationTutorialPython2	✓ Published	8/13/2018, 12:30 PM
	AzureAutomationTutorialScript	✓ Published	8/13/2018, 12:30 PM
	AzureClassicAutomationTutorial	✓ Published	8/13/2018, 12:30 PM
	AzureClassicAutomationTutorial2	✓ Published	8/15/2018, 10:36 AM
	AzureClassicAutomationTutorialScript	✓ Published	8/13/2018, 12:30 PM

Since the Automation Account credentials are user generated, there's a chance that the passwords are being reused somewhere else in the environment, but your mileage may vary.

Script Usage

In order for this script to work, you will need to have the AzureRM and Azure PowerShell modules installed. Both modules have different ways to access the same things, but together, they can access everything that we need for this script.

- Install-Module -Name AzureRM
- Install-Module -Name Azure

The script will prompt you to install if they're not already installed, but it doesn't hurt to get those installed before we start.

*Update (3/19/20) – I've updated the scripts to be Az module compliant, so if you're already using the Az modules, you can use the Get-AzPasswords (versus Get-AzurePasswords) instead.

The usage of this tool is pretty simple.

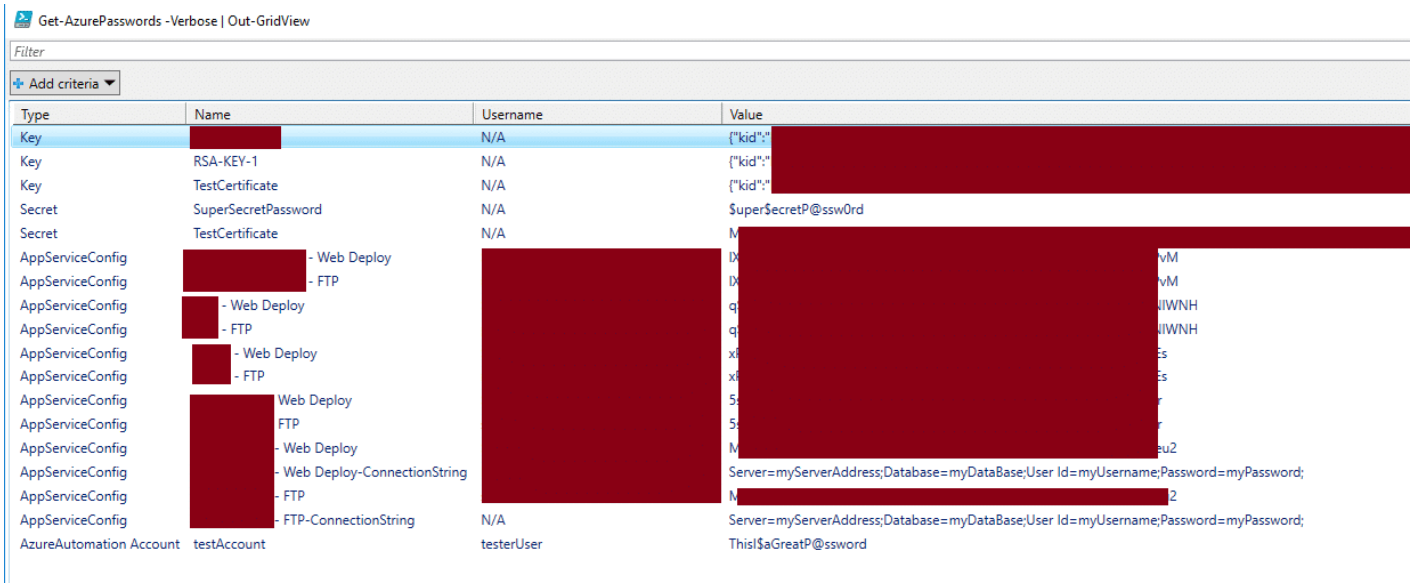
1. Download the code from GitHub – https://github.com/NetSPI/MicroBurst
2. Load up the module
    1. Import-Module .\Get-AzurePasswords.ps1
    2. or load the script file into the PowerShell ISE and hit F5
3. Get-AzurePasswords -Verbose
    1. Either pipe to Out-Gridview or to Export-CSV for easier parsing
    2. If you're not already authenticated to the Azure console, it will prompt you to login.
    3. The script will also prompt you for the subscription you would like to use
4. Review your creds, access other systems, take over the environment

If you're having issues with the PowerShell execution policy, I have it on good authority that there's at least 15 different ways that you can bypass the policy.

Sample Output:

- Get-AzurePasswords -Verbose | Out-GridView

```
PS C:\MicroBurst> Get-AzurePasswords -Verbose | Out-GridView
VERBOSE: Logged In as testaccount@example.com
VERBOSE: Getting List of Key Vaults...
VERBOSE:    Exporting items from example-private
VERBOSE:    Exporting items from PasswordStore
VERBOSE:        Getting Key value for the example-Test Key
VERBOSE:        Getting Key value for the RSA-KEY-1 Key
VERBOSE:        Getting Key value for the TestCertificate Key
VERBOSE:        Getting Secret value for the example-Test Secret
VERBOSE:            Unable to export Secret value for example-Test
VERBOSE:        Getting Secret value for the SuperSecretPassword Secret
VERBOSE:        Getting Secret value for the TestCertificate Secret
VERBOSE: Getting List of Azure App Services...
VERBOSE:    Profile available for example1
VERBOSE:    Profile available for example2
VERBOSE:    Profile available for example3
VERBOSE: Getting List of Azure Automation Accounts...
VERBOSE:    Getting credentials for testAccount using the lGVeLPZARrTJdDu.ps1 Runbook
VERBOSE:        Waiting for the automation job to complete
VERBOSE: Password Dumping Activities Have Completed
```



*The PowerShell output above and the Out-Gridview output has been redacted to protect the privacy of my test Azure subscription.

Alternatively, you can pipe the output to Export-CSV to save the credentials in a CSV. If you don't redirect the output, the credentials will just be returned as data table entries.

Conclusion

There's a fair number of places where credentials can hide in an Azure subscription, and there's plenty of uses for these credentials while attacking an Azure environment. Hopefully this script helps automate your process for gathering those credentials.

For those that have read "*Pentesting Azure Applications*", you may have noticed that they call out the same credential locations in the "Other Azure Services" chapter. I actually had most of this script written prior to the book coming out, but the book really helped me figure out the Automation account credential section.

If you haven't read the book yet, and want a nice deep dive on Azure security, you can get it from no starch press – https://nostarch.com/azure