Improper Control of Generation of Code ('Code Injection') in Azure CLI

Critical **chasewilson** published **GHSA-47xc-9rr2-q7p4** on Oct 24

Package

Azure CLI (Windows)

Affected versions	Patched versions
< 2.40.0	>= 2.40.0

Description

Description

In versions previous to 2.40.0, Azure CLI contains a vulnerability for potential code injection. Critical scenarios are where a hosting machine runs an Azure CLI command where parameter values have been provided by an external source.

For example: Application X is a web application with a feature that allows users to create Secrets in an Azure KeyVault. Instead of constructing API calls based on user input, Application X uses Azure CLI commands to create the secrets. Application X has input fields presented to the user and the Azure CLI command parameter values are filled based on the user input fields. This input, when formed correctly, could potentially be run as system commands. Below is an example of the resulting Azure CLI command run on the web app's hosting machine.

```
az keyvault secret set --vault-name SomeVault --name foobar --value "abc123|whoami"
```

The above command could potentially run the `whoami` command on the hosting machine.

Interactive, in-terminal use and automation/pipeline scenarios have not been identified as critical risk scenarios.

Code injection prerequisites

The vulnerability is only applicable when the Azure CLI command is run on a Windows machine ***and*** with any version of PowerShell **_and_**when the parameter value contains the `&` or `|` symbols. If any of these prerequisites are not met, this vulnerability is not applicable.

1. The command has to be run on Windows

The Azure CLI has an entry script that, when run on Windows, calls cmd.exe to then call Python. This leads into the next prerequisite.

2. The command has to be executed by PowerShell.

PowerShell has input parsing designs that strip out the quotation marks of input with the expectation that it will be taken as a string. When used in a PowerShell environment, the command is input like the above command. However, when it passes through PowerShell into cmd.exe, it looks like the following.

```
az keyvault secret set --vault-name SomeVault --name foobar --value abc123|whoami
```

This leads to the 3rd prerequisite as it won't just try to run any parameter value as a command.

3. The parameter value has to contain a `&` or `|` symbols

In cmd.exe, the `&` and `|` symbols invoke command execution. When a string containing this symbols is passed directly to cmd.exe, quotes are kept and command execution is invoked. However, When a string is passed into PowerShell, the quotes are stripped and passed into cmd.exe making it open to execution.

So, in the `keyvault` example above, the `abc123` portion of the value will be accepted correctly but the value after the `|` symbol will be interpreted as a command.

Impact

Code injection

As mentioned in the above scenario where the value is being provided by and outside source to run an Azure CLI command, system commands or even scripts could be run on a hosting machine.

Patches

Upgrade to Azure CLI 2.40.0 or greater.

As of Azure CLI 2.40.0, a new .ps1 entry script is used as the entry point to call Python rather than cmd.exe. This removes the opportunity for cmd.exe to interpret input as a command invocation. Using this approach has introduced new issues however that you can read about in the "More information" section.

Upgrade to 2.41.0 or greater and manually call the azps.ps1 entry script in identified critical scenarios.

In Azure CLI 2.41.0 we have reverted back to using the cmd.exe entry script as the default while keeping the azps.ps1 entry script for manual Azure CLI calls if users require it.

```
C:\Program Files (x86)\Microsoft SDKs\Azure\CLI2\wbin\azps.ps1 keyvault secret set --vault-name
```

More information

PowerShell Parsing with Azure CLI

PowerShell's input parsing design has caused regressions and issues in Azure CLI's behavior resulting in broken scripts and pipelines. Below are the known issues and links to GitHub issues. This should not be taken as a complete list since these are ***only the reported*** issues. Users should verify command effectiveness before use in production environments.

1. PowerShell arrays can't be passed to Azure CLI
2. Argument passthrough token (`--`) doesn't work with Azure CLI in PowerShell
3. Stop parsing token (`--%`) no longer works with Azure CLI in PowerShell
4. stdin passing is interrupted for Azure CLI in PowerShell
5. Azure CLI returns 0 when failing in PowerShell
6. Azure CLI can no longer be invoked by `Start-Process`

To avoid these breaking changes, in Azure CLI 2.41.0 we have reverted back to using the cmd.exe entry script as the default while keeping the azps.ps1 entry script for manual Azure CLI calls if users require it.

> The .ps1 entry script is only required for similarly identified scenarios like the example above. Interactive use and automation scenarios have not been identified as high risk.

If the azps.ps1 script is needed, you can call it like this:

```
C:\Program Files (x86)\Microsoft SDKs\Azure\CLI2\wbin\azps.ps1 vm create
```

The Azure CLI team is still considering further mitigations to remove the vulnerability altogether. Please check back with this CVE in the future for any further updates.

If you have any questions or comments about this advisory:

- Open an issue in Azure CLI GitHub repo
- Email us at AzPyCLI@microsoft.com

Severity

Critical

CVE ID

CVE-2022-39327

Weaknesses

CWE-94