Powershell Deep Dive

Among the various PowerShell options, the following are the most commonly used by attackers:

	Command Line Argument	Description
Stay Hidden	`-WindowsStyle (-w) Hidden`	Hides the PowerShell window session
	`-NoProfile (-nop)`	You can create PowerShell profiles that define various aliases, defaults, and other options. This is really just a convenience, not a security measure. Most attackers will avoid any such unknowns by simply passing "-noprofile" or "-nop" on the command line.
	`-NonInteractive (-noni)`	Does not present an interactive prompt to the user
	`-NoLogo (-nol)`	Does not present the PowerShell copyright startup banner

Execute	`-ExecutionPolicy (-ep) Bypass`
	or
	`-ExecutionPolicy (-ep) Unrestricted`
		Execution policies let you decide the conditions under which scripts can be run or not (e.g., whether to warn on remotely downloaded scripts or not). Execution policies are not a security measure – they are really just intended to keep users from making mistakes. Attackers can simply override any default policies.
		The bypass option allows any script to run without warning. The unrestricted option allows unsigned scripts to run without warning.

	`-Command (-c) <Commands>`	Any command that can be entered interactively in the PowerShell window can be specified as a command line argument. Multiple commands can be specified in this manner.
	`-File (-f) <FilePath>`	Passes in an external script file for execution
	`-EncodedCommand (-e) <Base64EncodedCommand>`	If a command sequence is complex, or contains quotation marks or other special characters, it can be difficult to pass them properly on a command line. The EncodedCommand lets you pass in these commands as a base64 encoded string. While not a security measure by any means, administrators may also use this option to pass usernames or passwords as arguments, to avoid them appearing in plain text. Of course, attackers love this option because it obfuscates their activity; resulting in commands that look like: `powershell.exe -e ZQBjAGgAbwAgACcAWQBvAHUAIABhAHIAZQAgAHAAdwBuAGUAZAAhACcA`

The "-EncodedCommand" (-e or -enc) option was by far the most popular technique found in our research because it allows the attacker to pass complete binaries directly on the command line. This option is most commonly coupled with "-ExecutionPolicy Bypass" to ensure execution occurs.

A number of the popular PowerShell exploit toolkits open the door for base64 encode hacking tools such as mimikatz (used to gather credential data) and invoke these tools entirely from memory without ever creating a file on disk. The resulting command lines tend to be thousands of characters long, and this characteristic can be a valuable indicator when looking for malicious PowerShell use.

Within the PowerShell commands (or cmdlets), whether passed on the command line or contained within script files, are the following highly useful commands and common indicators of suspicious activity:

`Invoke-Expression (iex)`	Evaluates and executes a string. Since most malicious code is either encoded or obfuscated, the actual code ends up in some variable that gets passed to Invoke-Expression (or "iex") for execution.
`Invoke-Command`	Can execute a PowerShell command on either a local or a remote computer. This is similar to the use of PsExec that is often used by attackers for remote inspection or lateral movement.

`DownloadString`
 	
`DownloadFile`
	From the System.Net.WebClient library, will download the content of a URI into either a string variable or directly to a file.

While different organizations may have different cultures in terms of flexibility and control, there really is no reason to ignore setting standards regarding how PowerShell should be used.

Examples of standards for PowerShell you can set include:

- Change ExecutionPolicy to only allow signed scripts to run.
- Require all PowerShell scripts to be run from a specific location or path.
- Discourage (or require exception for) the use of encoded parameters on the command line
- Discourage (or block) PowerShell scripts from downloading content from the Internet (or specify a "whitelist" of allowed IP addresses only).
- Discourage (or block) the use of PowerShell to invoke commands on remote systems
- Require a custom parameter to be passed on all "legitimate" PowerShell usage.
- Restrict PowerShell to specific users in your organization.
- Require PowerShell to be launched from a specific process.

It is true that attackers may be able to learn your policies or even bypass them, but by having clearly defined standards, you (a) make it much harder for the attacker to go undetected, and (b) make it easier for your security team to identify and investigate the suspicious instances.

Once you are capturing all PowerShell executions, you can set up alerts on key indicators.

Suspicious command line arguments	As noted earlier, command lines that include "bypass" and "encodedcommand" (or "-e") are highly suspicious. Even if there are legitimate instances of these, they should be relatively easy to identify and filter out. Command lines that contain "downloadfile" or "downloadstring" or "invoke-expression" (or "iex") are also suspicious.
Parent process	One of the most valuable techniques for identifying malicious use is to monitor which process launched PowerShell. Attackers can vary command lines and scripts with ease, but often have less control over how their initial malicious code is executed in the first place. For example, was PowerShell launched from Word or Excel (e.g., an infected Office document)? Was PowerShell launched from a browser process such as iexplore.exe or chrome.exe? Also, because attackers will often nest encoded payloads as command lines, the net result is a multilevel process tree where PowerShell launches PowerShell. Those types of patterns can be extremely useful for identifying malicious use.
Command Line Length	PowerShell command lines that are more than 1,000 characters long are highly suspicious. As discussed, attackers will often encode an entire binary payload on the command line to prevent hitting the file system, and this shows up as a ridiculously long command line.
Network connectivity	Does the PowerShell instance connect to an external IP address?
Cross Process Activity	Does the PowerShell process open handles to other processes such as lsass.exe or wmiprvse. exe? While some legitimate applications access lsass.exe and other system processes, when it comes from PowerShell, it is highly indicative of attempted credential theft or privilege escalation.

PowerShell enables you to define your own custom parameters, so if you have set a standard that a specific keyword or argument must be passed, looking for PowerShell launches without that keyword can be useful. If PowerShell is only authorized for specific users, you can alert on PowerShell processes running as any other user.

PowerShell is just a tool. Because it is a ubiquitous technology, used more often for legitimate purposes than not, it is an ideal way for attackers to remain undetected. It's ability to dynamically load and execute code without touching the file system makes it especially difficult to secure.