Investigating Powershell Attacks

The widespread availability of PowerShell in an average corporate Windows environment, the maturation of PowerShell attack toolkits, and the steady increase in PowerShell "know-how" among intruders has created a perfect storm for those seeking to protect a network or investigate a compromise.

The goals of this research were to identify the sources of evidence on disk, in logs, and in memory, resulting from malicious usage of PowerShell - particularly when used to target a remote host. Understanding these artifacts can help reconstruct an attacker's activity during forensic analysis of a compromised system. In addition, they can help analysts recognize the sources of evidence that are suitable for proactive monitoring - both on a single system and at scale - to detect PowerShell attacks.

The following sections summarize each of the sources of evidence that may provide evidence of malicious PowerShell usage on a compromised system. These sources include the registry, prefetch files, memory, event logs, and network traffic.

REGISTRY
The authors did not identify any registry keys or values that recorded the execution of PowerShell scripts, commands, or remoting activity. However, an attacker may tamper with PowerShell configuration settings that are resident in the registry to facilitate their activity. One such example is the PowerShell execution policy, which controls the profiles and scripts that a user is permitted to load and execute on a system. The registry stores this setting in the value ExecutionPolicy within key

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

By default, the policy is set to Restricted. An attacker may change the setting to before attempting to execute malicious PowerShell Bypass script. This would result in an update to the Last Modified timestamp of the registry key. An attacker could avoid modifying this setting and simply include the command-line option -*ExecutionPolicy Bypass* - each time they invoked PowerShell.

PREFETCH
Windows Prefetch is a performance enhancement feature, first introduced in Windows XP, designed to shorten load times during boot and application startup. The operating system stores prefetch files, denoted with extension .PF, in the directory *%systemroot%\prefetch*. Forensic investigators often use the prefetch as a source of evidence of executable files that previously ran on a system.

In order to be present within the prefetch file's accessed file list, a given script must be loaded within the first ten seconds of *powershell.exe* execution. This reliably occurs when running *powershell.exe* command line with a script argument, but not when using an interactive PowerShell session.

As part of an investigative process, the authors recommend the following basic steps:

- Examine the PowerShell prefetch file creation timestamp and last run timestamp to determine if they correlate with other periods of suspected attacker activity.
- Parse or string-search the accessed file list and examine the names and paths of any referenced .PS1 files

It also may be possible to conduct frequency analysis of script names and paths referenced across all of the gathered prefetch files, in order to identify uncommon or suspicious entries.

NETWORK TRAFFIC
The authors did not extensively analyze network-based evidence resulting from PowerShell remoting activity. As of PowerShell version 2.0, all remoting traffic occurs over ports 5985 (HTTP) and 5986 (HTTPS) by default. In both cases, the request payloads are encrypted - use of HTTPS only adds header encryption since all content is sent over SSL.

Investigators may have more success conducting network flow analysis to identify anomalous usage of PowerShell remoting. If remoting is legitimately used for system administration activities in an environment, it should originate from a predictable set of source systems. Organizations with the capability to monitor flows across internal-to-internal, DMZ-to-internal, or VPN-to-internal network segments should attempt to baseline traffic over ports 5985 and 5986. This may help identify unauthorized usage of remoting by an attacker.

MEMORY
As is always the case with memory forensics, time is of the essence. The authors' research concluded that it is possible to reconstruct at least fragments of PowerShell remoting activity in memory - even at the completion of a session. These techniques may be practical when conducting analysis of a single system of interest; however, they do not readily lend themselves to at-scale, proactive monitoring of systems in an enterprise environment.

EVENT LOGS
Windows event logs are instrumental when examining a potentially compromised system for evidence of attacker activity. Earlier versions of Windows PowerShell (version 2.0 and prior) provide few useful audit settings, thereby limiting the availability of evidence (such as a command history) useful for forensic analysts. PowerShell 3.0 and later has largely addressed this shortcoming with the introduction of a more robust module logging feature.

Nevertheless, even the default level of logging in older versions can provide sufficient evidence to identify signs of PowerShell usage, distinguish remoting from local activity, and provide context such as the duration of sessions and associated user account. This may help an analyst correlate other forensic evidence on a single system of interest with PowerShell activity.

The most significant addition to PowerShell 3.0 is the Module capability. This feature can provide detailed logging of all PowerShell command input and output, and can be configured on an individual system or through Group Policy. Module Logging records PowerShell commands and the resulting output regardless of whether they are executed locally or through remoting. This evidence is captured in EID 4103 events stored in the Microsoft Windows PowerShell Operational event log.

PERSISTENT POWERSHELL
As with any other type of malware, attackers can configure a Windows system to automatically execute PowerShell upon system startup or user logon, and thereby persist beyond the point of initial infection. Persistence is essential for certain types of malware, such as backdoors or keystroke loggers, to survive reboot and serve their objectives.

For example, an attacker could ensure that PowerShell automatically executed upon startup by setting the following registry key, value, and data:

- **Key:** `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`
- **Value:** `RunTotallyLegitPowerShell`
- **Data:**
  ```
  powershell.exe -NonInteractive -WindowStyle Hidden -ExecutionPolicy bypass -
  File "C:\windows\system32\evil.ps1"
  ```

Forensic investigation of this and other registry-based persistence mechanisms are well-documented. They leave easy-to-detect footprints, and tools such as RegRipper or AutoRuns greatly simplify the process of enumerating and detecting such anomalies.

Recurring scheduled tasks can be identified through analysis of *.job* files within *%systemroot%\tasks* and evidence in the Task Scheduler Operational Event Log. Use of the “StartUp” folder requires the creation of a file in one of a limited number of locations (either the “StartUp” folder for each targeted user, or the system-wide “All Users” directory).

Profiles and WMI
One noteworthy feature distinguishes PowerShell from other built-in Windows scripting languages (and, conveniently, can be used for malware persistence): the use of profiles. A profile is simply a up script that executes whenever PowerShell starts

An attacker could add code to a user or system profile that executes an external binary, or more covertly, loads shellcode or a malicious DLL encoded in the profile itself. Under these conditions, the attacker need only ensure that *powershell.exe* executes upon startup or user logon.

How can an investigator identify evidence of this technique? Analysts should first review all system and user PowerShell profiles on disk for the presence of malicious code. Although this technique is not strictly required for persistence via WMI, its relative simplicity and inclusion in the PowerSploit Persistence module increases the likelihood that an attacker may use it.