Investigating PowerShell Attacks

By PowerShell Magazine

Published: 2014-07-16 · Archived: 2026-04-06 03:19:45 UTC

“Huh, that’s weird. Look at this system. I think the attacker used PowerShell.” It was late summer 2012, and we were working on an incident response investigation for a Fortune 100 technology company compromised by an intruder attempting to steal intellectual property. The evidence wasn’t terribly exciting: just a simple reconnaissance script to enumerate domain users and systems. But it was an anomaly – or at least, a rare occurrence within the scope of our previous case work. We conduct hundreds of incident response investigations every year, most of which involve targeted attacks for the purposes of espionage, stealing intellectual property, or theft of financial data. Attacker tools, tactics, and procedures regularly change and evolve – and PowerShell was a new wrinkle.

Fast forward almost two years later. Stories of PowerShell usage in both targeted compromises and opportunistic malware are hitting infosec media with alarming frequency. We also see these trends in our daily casework: an increasing number of investigations involve attacker reconnaissance, command execution, or data theft facilitated by PowerShell. Just a few months ago, we responded to a case where the attacker evolved from relying on custom tools and PsExec, to exclusive use of PowerShell remoting, for lateral movement and command-and-control (and thereby evaded detection for many months). The gap between attackers’ PowerShell skills, and organizations’ ability to detect and respond to its misuse, is growing.

Prior articles by Matthew Graeber, Joseph Bialek, and Will Schroeder did a great job of explaining why PowerShell is so dangerous in the hands of an attacker – particularly given elevated privileges during the post-exploitation phase of an incident. It provides:

- A built-in mechanism for remote command execution
- The ability to execute malicious code without ever touching disk
- The ability to evade many Anti-Virus and Intrusion Prevention Systems
- Full access to WMI and .NET Framework

The unauthorized use of PowerShell presents several challenges to forensic analysts and system administrators alike:

- As a legitimate component of Windows, PowerShell execution does not necessarily indicate malicious behavior.
- PowerShell 2.0 does not provide a practical mechanism for detailed (e.g. per-command) auditing. PowerShell 3.0 and later provides comprehensive module logging – but is only installed by default on Windows 8 or Server 2012, which remain uncommon in many corporate environments.
- PowerShell remoting sessions occur in ephemeral process memory with few-to-no persistent artifacts.
- Many system administrators and security professionals remain unfamiliar with PowerShell and its management or security controls.

Faced with these mounting challenges, we decided to research the forensic "footprints" left behind by the ways that an attacker might use PowerShell – a topic for which publicized information is scarce. Our work focused on three fundamental scenarios: local PowerShell execution, PowerShell remoting, and the configuration of a persistent PowerShell backdoor through the use of WMI. We examined multiple sources of evidence, including the registry, file system, event logs, process memory, and network traffic.

Unfortunately, we never found the "white whale" – a single source of evidence, consistently available across all versions of Windows and PowerShell, that provides a complete history of all activity on an endpoint regardless of how it occurs. However, we did identify multiple artifacts containing tidbits of information that, when combined, can solve common investigative questions.

In the upcoming weeks, we'll be releasing a whitepaper and presentation at Black Hat USA and DEF CON that focuses on the forensic analysis portion of our research. However, in this article we wanted to share a corollary of our findings – the sources of evidence that are best suited for establishing a baseline and monitoring a Windows enterprise environment.

As alluded to by Microsoft in their recent update to the Mitigating Pass-the-Hash whitepaper, organizations should orient their detection and prevention efforts around the assumption that a breach has occurred. More specifically, that means assuming that an attacker has successfully compromised credentials that provide local administrator-equivalent access to targeted system(s) (if not domain administrator outright). The worst-case scenario is unfortunately the reality for the majority of Windows environments that we encounter during investigations. Any security control put in place to limit the use of PowerShell – be it the execution policy, disabled remoting, or constrained endpoints, may be bypassed altogether.

Instead of depending on these settings to prevent malicious usage of PowerShell, we recommend using them to establish a baseline of normal activity in an environment. Deviations from this baseline may serve as an indication of attacker activity. We recommend that organizations formulate a PowerShell monitoring strategy by first assessing and enumerating the following:

- Which servers/server groups are administered via PowerShell remoting? By local PowerShell script execution? What about workstations/end-user systems?
- Which domain accounts use PowerShell remoting? What are the source hostnames from which these users would administer systems?
- What are the names and common directories used for legitimate PowerShell scripts within the environment? Are legitimate scripts used by the organization digitally signed?
- Are any systems configured to automatically load and execute PowerShell scripts for maintenance or administration purposes?

Once complete, administrators can rely on centralized Windows event log forwarding and collection (for at-scale monitoring) or local event log analysis (for targeted forensics and investigations) to identify signs of anomalous PowerShell usage. This effort will require filtering and tuning – that's where having a baseline, or even monitoring a subset of known-good systems with common configuration for a period of time, can help.

In the interest of providing recommendations applicable to all versions of PowerShell, inclusive of 2.0, we recommend evaluating the following log sources and events:

- Windows PowerShell event log entries indicating the start and stop of PowerShell activity:
    - Event ID 400 (“Engine state is changed from None to Available”), upon the start of any local or remote PowerShell activity.
    - Event ID 600 referencing “WSMan” (e.g. “Provider WSMan Is Started”), indicating the onset of PowerShell remoting activity on both source and destination systems.
    - Event ID 403 (“Engine state is changed from Available to Stopped”), upon the end of the PowerShell activity.
- System event log entries indicating a configuration change to the Windows Remote Management service:
    - Event ID 7040 “The start type of the Windows Remote Management (WS-Management) service was changed from [disabled / demand start] to auto start.” – recorded when PowerShell remoting is enabled.
    - Event ID 10148 (“The WinRM service is listening for WS-Management requests”) – recorded upon reboot on systems where remoting has been enabled.
- WinRM Operational event log entries indicating authentication prior to PowerShell remoting on an accessed system:
    - Event ID 169 (“User [DOMAIN\Account] authenticated successfully using [authentication_protocol]”)
- Security event log entries indicating the execution of the PowerShell console or interpreter:
    - Event ID 4688 (“A new process has been created”) – includes account name, domain, and executable name in the event message.
- AppLocker event log entries recording the local execution of PowerShell scripts. We recommend enabling AppLocker in audit mode across an environment for this specific purpose. Upon script execution in audit mode, the AppLocker MSI and Script Event Log may record:
    - Event ID 8006 (“[script_path] was allowed to run but would have been prevented from running if the AppLocker policy were enforced”)
    - Event ID 8005 (“[script_path] was allowed to run”).

Both of these events will include the user account that attempted to execute a script.

We fully expect that threat actors will continue to employ more sophisticated PowerShell techniques and improve their counter-forensic strategies over time. It is our hope that our work will increase awareness of these attacks, motivate organizations to enhance their detection and monitoring capabilities, and drive additional research. We look forward to sharing further details in our forthcoming whitepaper and presentations at Black Hat USA and DEF CON.

Authors are Ryan Kazanciyan and Matt Hastings.

Ryan Kazanciyan is a Technical Director with Mandiant and has eleven years of experience in incident response, forensic analysis, and penetration testing. Since joining Mandiant in 2009, he has led investigation and remediation efforts for dozens of Fortune 500 organizations, focusing on targeted attacks, industrial espionage, and financial crime.

Matt Hastings is a Consultant with Mandiant, a division of FireEye, Inc. Based in the Washington D.C area, Matt focuses on enterprise-wide incident response, high-tech crime investigations, penetration testing, strategic corporate security development, and security control assessments; working with the Federal government, defense industrial base, financial industry, Fortune 500 companies, and global organizations.

Share on:
