Powershell Power Hell: Hunting For Malicious Powershell With Splunk

Ryan Chapman & Lisa Tawfall
Bechtel Corporation
Disclaimer

During the course of this presentation, we may make forward looking statements regarding future events or the expected performance of the company. We caution you that such statements reflect our current expectations and estimates based on factors currently known to us and that actual events or results could differ materially. For important factors that may cause actual results to differ from those contained in our forward-looking statements, please review our filings with the SEC. The forward-looking statements made in this presentation are being made as of the time and date of its live presentation. If reviewed after its live presentation, this presentation may not contain current or accurate information. We do not assume any obligation to update any forward looking statements we may make. In addition, any information about our roadmap outlines our general product direction and is subject to change at any time without notice. It is for informational purposes only and shall not, be incorporated into any contract or other commitment. Splunk undertakes no obligation either to develop the features or functionality described or to include any such feature or functionality in a future release.
Agenda

• Who Are We?
• Why Focus On Powershell?
• Setting Up Powershell Logging
• Finding Malicious Powershell
Bechtel Corporation

Bechtel Corporation is the largest construction and civil engineering company in the U.S., making the company a target rich environment. Since 2011, Bechtel has set out to build a world-class Security Operations Center, which relies heavily on Splunk.
Ryan Chapman

- Network Security Monitoring Analyst
- Incident Handler
- CIRT / SOC Liaison
- “Did You Check Splunk?” Guy
- *No, Really* – Did You Check Splunk?

@rj_chap
Lisa Tawfall

- Security Unicorn (Yes, really)
- Lead for the team that manages security infrastructure at Bechtel
- Splunk Administrator
- Breaker of Splunk
- Fixer of Splunk

@ltawfall
“We wouldn’t be able to do our jobs without Splunk.”
Why Focus on PowerShell?
PowerShell Shenanigans

Attackers LOVE PowerShell

• Why Are Attackers Using PowerShell?
  – Powerful, Built-in Tool – (Nearly) Always Available
    ‣ PowerShell is Already in Your Environment!
    ‣ A Hacker’s Best Toolkit = Tools on the Box!
  – Can Execute in Memory (Diskless)
  – Easy to Avoid Detection

• PowerShell is a Growing Concern
Powershell Shenanigans Cont’d.

Cmdlet Mayhem

• Remote Access Methods (more later)
• Integrates w/Core Windows Components
  – Native Win32 API
  – .NET Framework
  – WMI
• Can Access Registry, Firewall, etc.
Nature Of The Threat

Some Statistics

• Per Carbon Black Analysis of 1,100 Incidents:
  – PowerShell Used: **38%**
    ‣ No Security Alerts Reported: 31%
  – APT-Related Attacks: 13%

• Frameworks Leveraging PowerShell:
  – PowerSploit, Nishang, MetaSploit, etc.
In Q4 2015, We Were Pentested
  – Very Strong Red Team

We Caught Them Immediately 😊
  – They Love PowerShell
  – *But We Love PowerShell Too*

Pre-Existing Saved Search FTW
  – Team Member Read an OSINT Article
    ‣ Hunting ➔ Saved Search

This was a Huge Win For Us
Setting Up PowerShell Logging
Types of PowerShell Logging

Four Ways to Win the Game

- Process Auditing – Event ID **4688**
  - Also enable Command Line Capture

- Module Logging – Event ID **4103** *(Payload)*
  - Records Pipeline Execution

- Script Block Logging – Event ID **4104** *(ScriptBlockText)*
  - Captures Script Blocks w/Deobfuscated Commands

- Transcription *(std_out)*
  - Logs All PowerShell Events to Text Files
Enabling Process Creation Auditing

Event ID 4688 + Command Line Logging

- No WLS? *No Problem!* Bad Credit? [Actually That’s a Problem]
- Computer Configuration → Policies → ...

**Enable Process Creation Auditing**
Windows Settings → Security Settings → Advanced Audit Policy
Configuration → Audit Policies → Detailed Tracking →
Audit Process Creation

**Include Command Line**
Administrative Templates → System → Audit Process Creation →
Include command line in process creation events
Enabling PowerShell Logging

Event IDs 4103 + 4104 & Transcription

- Let’s Create Another GPO!
- Computer Configuration ➔ Policies ➔ Administrative Templates ➔ Windows Components ➔ Windows PowerShell
- **Enable:** Turn on Module Logging
  - Add Module Names
- **Enable:** Turn on PowerShell Script Block Logging
- **Enable:** Turn on PowerShell Transcription
To turn on logging for one or more modules, type the following module names in the list:

- Microsoft.PowerShell.*
- Microsoft.WSMAN.Management
- *

To turn on logging for the Windows core modules, type the following module names in the list:

- Microsoft.PowerShell.*
- Microsoft.WSMAN.Management

The LogPipelineExecutionDetails property of the module is set to True.

If you disable this policy setting, logging of execution events is disabled for all Windows PowerShell modules. Disabling this policy setting for a module is equivalent to setting the LogPipelineExecutionDetails property of the module to False.

If this policy setting is not configured, the LogPipelineExecutionDetails property of a module or snap-in determines whether the execution events of a module or snap-in are logged. By default, the LogPipelineExecutionDetails property of all modules and snap-ins is set to False.
Ingesting Transcription Logs

The Basics

- Custom SplunkUF Deployment App
  - [monitor://C:\windows\system32\logfiles\powershell\...\*.txt]
    - followTail=false
    - sourcetype=ps_transcript
    - index=powershell
    - disabled = false
    - crcSalt=<SOURCE>

- index=powershell

- Props/Transforms
Windows PowerShell transcript
Start time: 20160714111156
Username: CURLY\SYSTEM
Run As User: CURLY\SYSTEM
Machine: XXXXXXXX (Microsoft Windows NT 10.0.10586.0)
Host Application: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -ExecutionPolicy bypass -file C:\Program Files\SplunkUniversalForwarder\etc\apps\windows_pipes\bin\enum_pipes.ps1
Process ID: 4820
PSVersion: 5.0.10586.122
PSCompatibleVersions: 1.0, 2.0, 3.0, 4.0, 5.0.10586.122
BuildVersion: 10.0.10586.122
CLRVersion: 4.0.30319.42000
WSManStackVersion: 3.0
PSRemotingProtocolVersion: 2.3
SerializationVersion: 1.1.0.1

********************
They're Ugly 1/2
Transcription Logs

They're Ugly 2/2

******************************************************************************
Command start time: 20160714111156
******************************************************************************
PS>CommandInvocation(enum_pipes.ps1): "enum_pipes.ps1"
\\.\\pipe\\InitShutdown
\\.\\pipe\\lsass
\\.\\pipe\\ntsvc
....
\\.\\pipe\\TDLN-5692-41
\\.\\pipe\\PShost.131129935166577748.4820.DefaultAppDomain.powershell
******************************************************************************
Command start time: 20160714111156
******************************************************************************
PS>$global:?  
True    
******************************************************************************
Windows PowerShell transcript end
End time: 20160714111156
******************************************************************************

Script's Std Out
Line Breaker Challenge

MUST_BREAK_AFTER = End time: .+\$\*+$
MUST_NOT_BREAK_AFTER = \*+$
NO_BINARY_CHECK = true
TIME_FORMAT = %Y%m%d%H%M%S
TIME_PREFIX = Start time:

Windows PowerShell transcript start
Start time: 20160803161741
Username: CURLY\SYSTEM
RunAs User: CURLY\SYSTEM
Machine: XXXXXX (Microsoft Windows NT 10.0.10586.0)
Host Application: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
-ExecutionPolicy bypass -file C:\Program Files\SplunkUniversalForwarder\etc\apps\windows_pipes\linenum_pipes.ps1
Process ID: 8492
PSVersion: 5.0.10586.494
PSCompatibleVersions: 1.0, 2.0, 3.0, 4.0, 5.0.10586.494
BuildVersion: 10.0.10586.494
CLRVersion: 4.0.30319.42000
WSManStackVersion: 3.0
PSRemotingProtocolVersion: 2.3
SerializationVersion: 1.1.0.1

Game start time: 20160803161741
=========================================================================
PS> Command start time: 20160803161741
PS>MyInvocation("enum_pipes.ps1"); "enum_pipes.ps1"
\\\pipe\sconf\n\\\pipe\ntsvcs
\\\pipe\acercp
....
\\\pipe\va5b1bc44-53bc-464a-8eba-e79f825fa282
\\\pipe\browser
=========================================================================
PS> Command start time: 20160803161741
PS> cmdlet? True
Windows PowerShell transcript end
End time: 20160803161741
The Gory Details

Regex Is Your Friend
Field Parsing

```
***************
Windows PowerShell transcript start
Start time: 20160803161741
Username: CURLY\SYSTEM
RunAs User: CURLY\SYSTEM
Machine: XXXXXX (Microsoft Windows NT 10.0.10586.0)
Host Application: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
-ExecutionPolicy bypass -file c:\Program Files (x86)\SetGrace\bin\cham_pipes.psl
Process ID: 8492
PSVersion: 5.0.10586.494
PSCompatibilityVersions: 1.0, 2.0, 3.0, 4.0, 5.0.10586.494
BuildVersion: 10.0.10586.494
WSManStackVersion: 3.0
PSRemotingProtocolVersion: 2.3
SerializationProtocolVersion: 1.1.0.1
```
Regex Magic

Remember this Giant Std. Out Bit?

```
EXTRACT std_out = (?s)(?m).*+SerializationVersion: \[\d+.\]+(?<std_out>+.)(?m).*+Windows PowerShell transcript end
```
What Timezone is This Log?

Things that Drive Your Splunk Admin Ballistic

NOTHING HURTS MY HEAD MORE

THAN TIMEZONE MATH
Finding Malicious PowerShell
Base64 Encoded Commands

This Caught Our Red Team 😊

```
index=wls* EventID=4688 BaseFileName="powershell.exe"
CommandLine="*-en*" NOT ([REDACTED])
| sort 0 _time
| table _time, host, SubjectDomainName,
  SubjectUserName, BaseFileName, CommandLine,
  CompanyName, CreatorProcessName, NewProcessName,
  FileDescription, FileVersion, MD5
```
Enhanced Alert w/B64 Decryption

Requires Decrypt (App 2655)

index=wls* EventID=4688 (BaseFileName="powershell*.exe"
OR BaseFileName="cmd.exe")

(CommandLine="*-en*" OR CommandLine="*base64string*"

| rex field=CommandLine "-((?!enc|encodedcommand|
encode|en)\s')?(<?<base64_command>\w{20,1000}\=\=?)'\?"

| decrypt field=base64_command atob()
emit('base64_decoded_command')

| search base64_decoded_command=*
<table>
<thead>
<tr>
<th>_time</th>
<th>Computer</th>
<th>CommandLine</th>
</tr>
</thead>
<tbody>
<tr>
<td>11/9/15</td>
<td>[REDACTED]</td>
<td><code>powershell.exe -nop -w hidden -encodedcommand JABzAD0ATgBIAHcALQBPAGIAagBlAGMAdAAgAEkATwAuAE0AZQBtAG...AKQA7AA==</code></td>
</tr>
<tr>
<td>11/9/15</td>
<td>[REDACTED]</td>
<td><code>powershell -nop -exec bypass -EncodedCommand 'SQBFAFgAIAAoACgAbgBIAHcALQBvAGIAagBlAGMAdAAgAG4AZQB0A...AKQA7AA=='</code></td>
</tr>
</tbody>
</table>
Saved Search Results
Decoded Base64 – MOAR Base64 + Zipped!

$s$=New-Object IO.MemoryStream(,
(Convert)::**FromBase64String**("H4sIAAAAAAAAAL1We2/aSBD/
Gz7FqopkW+UZuJREi...DAAA");

IEX (New-Object IO.StreamReader(New-Object
IO.Compression.GzipStream **$s**,
[IO.Compression.CompressionMode]::**Decompress**))).ReadToEnd();
Secondary PowerShell Script
  ‣ $var_code == Shellcode

Shellcode Creates Named Pipe
  ‣ Inter-Process Communication

Errrrrrr.... No
  ‣ ALL HANDS ON DECK
Remote PowerShell

Common Remote Methods

Get-Service *winrm*
Enable-*PSRemoting*
New-*PSSession*
Enter-*PSSession*
Invoke-Command -computer

*General use of:* -computer

*NOTE:* -computer can specify 127.0.0.1*)
PowerShell: WSMAN

### Get-WSManInstance

Displays management information for a resource instance specified by a Resource URI.

```powershell
PS C:\> get-help *wsman*
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
<th>Module</th>
<th>Synopsis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable-WSManCredSSP</td>
<td>Cmdlet</td>
<td>Microsoft.WSMAN.Manage...</td>
<td>Disables Credential Security Support Provider.</td>
</tr>
<tr>
<td>Enable-WSManCredSSP</td>
<td>Cmdlet</td>
<td>Microsoft.WSMAN.Manage...</td>
<td>Enables Credential Security Support Provider.</td>
</tr>
<tr>
<td>Get-WSManCredSSP</td>
<td>Cmdlet</td>
<td>Microsoft.WSMAN.Manage...</td>
<td>Gets the Credential Security Support Provider.</td>
</tr>
<tr>
<td>Set-WSManQuickConfig</td>
<td>Cmdlet</td>
<td>Microsoft.WSMAN.Manage...</td>
<td>Configures the local computer for remote access.</td>
</tr>
<tr>
<td>Test-WSMan</td>
<td>Cmdlet</td>
<td>Microsoft.WSMAN.Manage...</td>
<td>Tests whether the WinRM service is running.</td>
</tr>
<tr>
<td>Invoke-WSManAction</td>
<td>Cmdlet</td>
<td>Microsoft.WSMAN.Manage...</td>
<td>Invokes an action on the object that is managed.</td>
</tr>
<tr>
<td>Connect-WSMan</td>
<td>Cmdlet</td>
<td>Microsoft.WSMAN.Manage...</td>
<td>Connects to the WinRM service on a remote computer.</td>
</tr>
<tr>
<td>Disconnect-WSMan</td>
<td>Cmdlet</td>
<td>Microsoft.WSMAN.Manage...</td>
<td>Disconnects the client from the WinRM service.</td>
</tr>
<tr>
<td>Get-WSManInstance</td>
<td>Cmdlet</td>
<td>Microsoft.WSMAN.Manage...</td>
<td>Displays management information for a resource instance.</td>
</tr>
<tr>
<td>Set-WSManInstance</td>
<td>Cmdlet</td>
<td>Microsoft.WSMAN.Manage...</td>
<td>Displays management information for a resource instance.</td>
</tr>
<tr>
<td>Remove-WSManInstance</td>
<td>Cmdlet</td>
<td>Microsoft.WSMAN.Manage...</td>
<td>Deletes a management resource instance.</td>
</tr>
<tr>
<td>New-WSManInstance</td>
<td>Cmdlet</td>
<td>Microsoft.WSMAN.Manage...</td>
<td>Creates a new instance of a management resource.</td>
</tr>
<tr>
<td>New-WSManSessionOption</td>
<td>Cmdlet</td>
<td>Microsoft.WSMAN.Manage...</td>
<td>Creates a WS-Management session option.</td>
</tr>
<tr>
<td>Disable-PSWSManCombinedTrace</td>
<td>Function</td>
<td>PS.Diagnostics</td>
<td>...</td>
</tr>
<tr>
<td>Disable-WSManTrace</td>
<td>Function</td>
<td>PS.Diagnostics</td>
<td>...</td>
</tr>
<tr>
<td>Enable-PSWSManCombinedTrace</td>
<td>Function</td>
<td>PS.Diagnostics</td>
<td>...</td>
</tr>
<tr>
<td>Enable-WSManTrace</td>
<td>Function</td>
<td>PS.Diagnostics</td>
<td>...</td>
</tr>
</tbody>
</table>
Remote PowerShell Search

Module Logging Method (4103)

index=wls* EventID=4103
ProviderName="Microsoft-Windows-PowerShell"

(Payload="*winrm*" OR Payload="*psremoting*" OR Payload="*pssession*" OR Payload="*invoke-command*" OR Payload="*wsman*"
[OR Payload="*–computer*"])


Remote PowerShell Search

Script Block Logging Method (4104)

index=wls* EventID=4104
ProviderName="Microsoft-Windows-PowerShell"

(ScriptBlockText="*winrm*" OR ScriptBlockText="*psremoting*" OR ScriptBlockText="*pssession*" OR ScriptBlockText="*invoke-command*" OR ScriptBlockText="*wsman*"
[OR ScriptBlockText="*–computer*"]}

38
PowerShell Cornucopia

These Are *Interesting*

index=wls* EventID=4688
(BaseFileName=powershell*.exe OR BaseFileName=cmd.exe)
(CommandLine="*get-process*" OR CommandLine="*get-service*" OR CommandLine="*get-filehash*" OR CommandLine="*get-hotfix*" OR CommandLine="*cd hk*" OR CommandLine="*get-itemproperty hk*" OR CommandLine="*netfirewallrule*"
Common Injection Methods

INJECTION SON!

PowerShellMafia’s PowerSploit

Dirty Dirty Tricks

- Open Source PowerShell Attack Framework
  - Becoming More and More Common

- We Can Enumerate PowerSploit Modules
  - And Look For Them
  - And yell/cry/smile if we find any

Q: Is Anyone Running PowerSploit?
(BETTER NOT BE!)
<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>CodeExecution</td>
<td>Set all module versions to 3.0</td>
<td>7 months</td>
</tr>
<tr>
<td>Exfiltration</td>
<td>Don't search for SYSTEM token by using hard coded English name for SYSTEM</td>
<td>7 months</td>
</tr>
<tr>
<td>Mayhem</td>
<td>Set all module versions to 3.0</td>
<td>7 months</td>
</tr>
<tr>
<td>Persistence</td>
<td>Added ScheduledTaskHourly to New-UserPersistenceOption</td>
<td>3 months</td>
</tr>
<tr>
<td>Privesc</td>
<td>Set all module versions to 3.0</td>
<td>7 months</td>
</tr>
<tr>
<td>Recon</td>
<td>Set all module versions to 3.0</td>
<td>7 months</td>
</tr>
<tr>
<td>ScriptModification</td>
<td>Set all module versions to 3.0</td>
<td>7 months</td>
</tr>
</tbody>
</table>
CodeExecution

Execute code on a target machine.

- **Invoke-DllInjection**
  Injests a Dll into the process ID of your choosing.

- **Invoke-ReflectivePEInjection**
  Reflectively loads a Windows PE file (DLL/EXE) into the powershell process, or reflectively injects a DLL into a remote process.

- **Invoke-Shellcode**
  Injects shellcode into the process ID of your choosing or within PowerShell locally.

- **Invoke-WmiCommand**
  Executes a PowerShell ScriptBlock on a target computer and returns its formatted output using WMI as a C2 channel.
Example: CodeExecution Modules

Script Transcription Method (index=powershell)

\[
\text{index=powershell std\_out=""*"} \\
(\text{std\_out=""*DllInjection*" OR std\_out=""*ReflectivePEInjection*" OR std\_out=""*Shellcode*" OR std\_out=""*WmiCommand*"})
\]
Recap of 5 Takeaways

‘Member These Things

- PowerShell is **Already in Your Environment**
- PowerShell Logging **Must Be Enabled**
- Event Codes of Interest: **4103, 4104, & 4688**
- Create, Test, and Perfect **Baselines** to Avoid FPs
- **Ongoing Research** Leads to New Search Ideas

@rj_chap

@ltawfall
THANK YOU

QUESTIONS?

Ryan Chapman & Lisa Tawfall
Bechtel Corporation
References

Check ‘Em Out!

Bonus Reference

Highly Recommended


- Malware Archaeology’s Cheat Sheets Include:
  - Windows PowerShell Logging
  - Windows Logging
  - Windows Splunk Logging
  - And more!