PSCONF.EU

2018

The Power of the **Dark** Side: Offensive PowerShell Workshop

Will Schroeder and Jared Atkinson

about_Authors

- **@harmj0y** and **@jaredcatkinson** on Twitter
- Red Team and Hunt at **@SpecterOps**
- Microsoft PowerShell/CDM MVPs
- will [at] harmj0y.net | jared [at] invoke-ir.com

Agenda

- Introduction/setup
- Token Manipulation
  - Get-System to Get-AccessToken
- Mimikatz
  - Passwords, tickets, and detection
- PowerShell without powershell.exe
  - UnmanagedPowerShell/PSInject, attacks and detection
- (If time) Alternate PowerShell Hosts
  - Searching for and abusing alternate hosts
- (If time) Subversive PowerShell Profiles

Workshop Goals

- Hacking is fun! We want you give you hands on with the tools we're familiar with :)
  - Plus you can't defend against what you don't know!
- Expose you to specific security-related PowerShell subject areas that you may not be familiar with
- Give you practical defensive advice and let you play with hands on detection of some of these techniques

DISCLAIMER

- We hope you trust us :)
- We will be running malicious code!
- You have been warned…

Labs

- Download the lab material from the ./workshops/Offensive PowerShell Workshop/ folder from https://github.com/HarmJ0y/2018
  - Will be updated on https://github.com/psconfeu/2018
- Download the materials to an exclusion folder for your antivirus :)
- We recommend temporarily disabling Defender:
  - From an elevated prompt:
  - PS\> **Set-MpPreference -DisableRealtimeMonitoring $True**

PSCONF.EU

Why We Chose **PowerShell** (offense)

- Allows us to automate previously time-consuming/expensive tradecraft

- Provides:
  - Full .NET access
  - Execution through a trusted Microsoft binary
  - Direct access to the Win32 API
  - Ability to assemble malicious binaries in memory
  - Default installation on Win7+

Why We're Diversifying Our Offense

- The PowerShell security team is awesome!
  - PowerShell == the most secure scripting language, *ever*

- Version 5 is phenomenal from a security perspective:
  - Deep script block logging
  - Script block auto logging
  - Enforcement of security protections for ALL PowerShell hosts
  - AMSI hooks for AV
  - Constrained language mode
  - Just-enough-administration (JEA)

Why We Chose **PowerShell** (defense)

- Fits in with our point-in-time sweep approach methodology
- Provides:
  - Full .NET access
  - Execution through a trusted Microsoft binary
  - Direct access to the Win32 API
  - Default installation on Win7+
  - Robust remoting functionality
  - No additional configuration changes necessary!

PowerShell Conference EU 2018

PSCONF.EU

PowerShell Conference EU 2018

PSCONF.EU

WHEN YOU ONLY HAVE 3 HOURS
TO TEACH ALL OF POWERSHELL SECURITY
imgflip.com

2018

Token Manipulation

From **Get-System** to **Get-AccessToken**

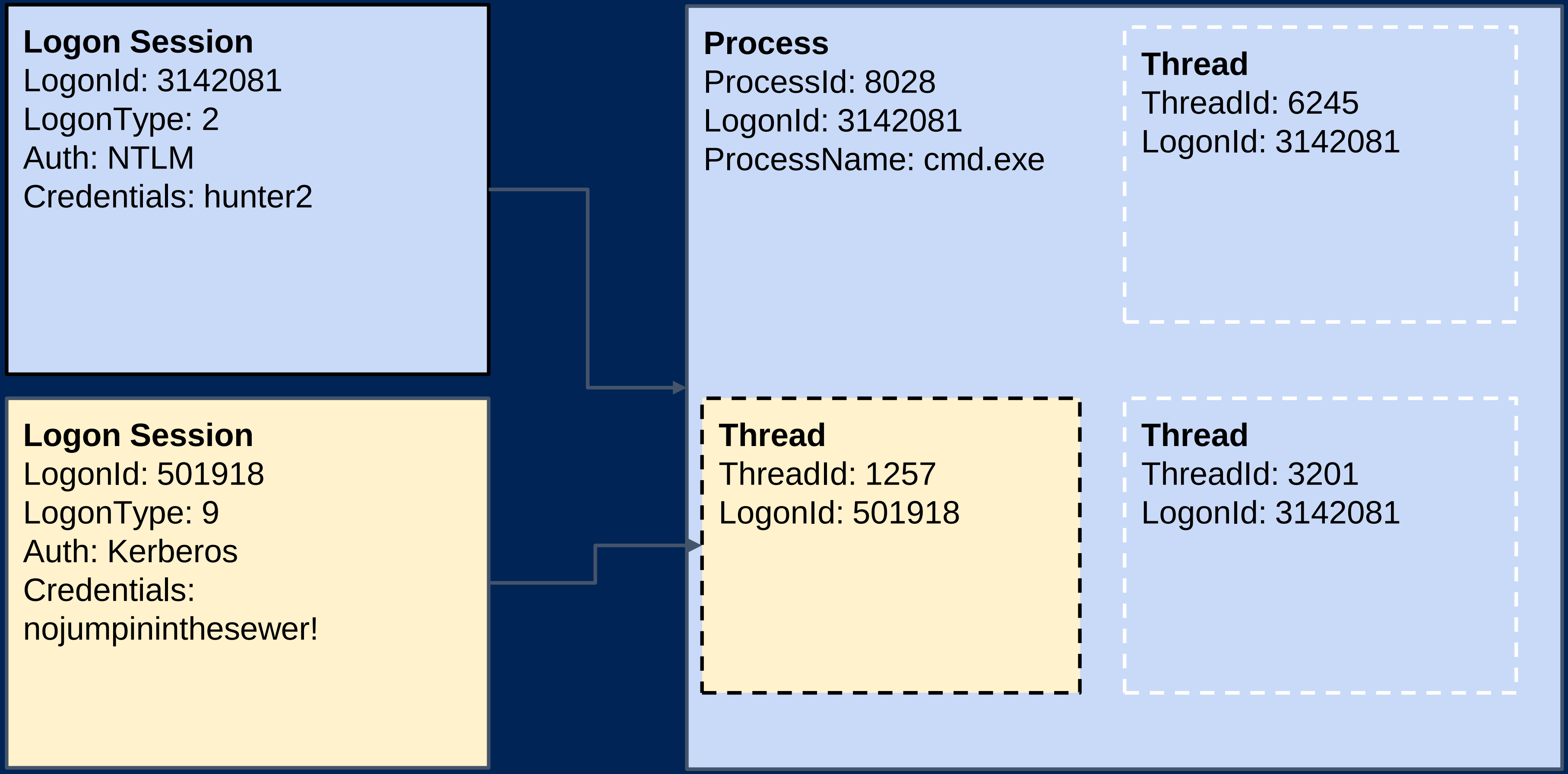
Windows Authentication Overview

- Windows creates a logon session upon successful authentication
  - User credentials (if any) are stored in lsass.exe
  - Credentials may be used later for Single Sign On
- Access tokens define the security context of a process/thread
  - When a process/thread wants to act in a user context it uses a token
- Tokens are tied to logon sessions and determine how the cred is used
  - Credential → Logon Session → Access Token→ Thread/Process

Token Types & Impersonation Levels

- 1) **Primary** - a process token
  - OS uses token's credentials to authenticate remotely.
- 2) **Impersonation** - a thread token
  - Threads use impersonation tokens to impersonate other security contexts
  - OS *might* use token's credentials to authenticate remotely
- Impersonation tokens have impersonations *levels*, but we won't worry about that here
- Impersonation tokens can be "stolen" (cloned) from other processes!

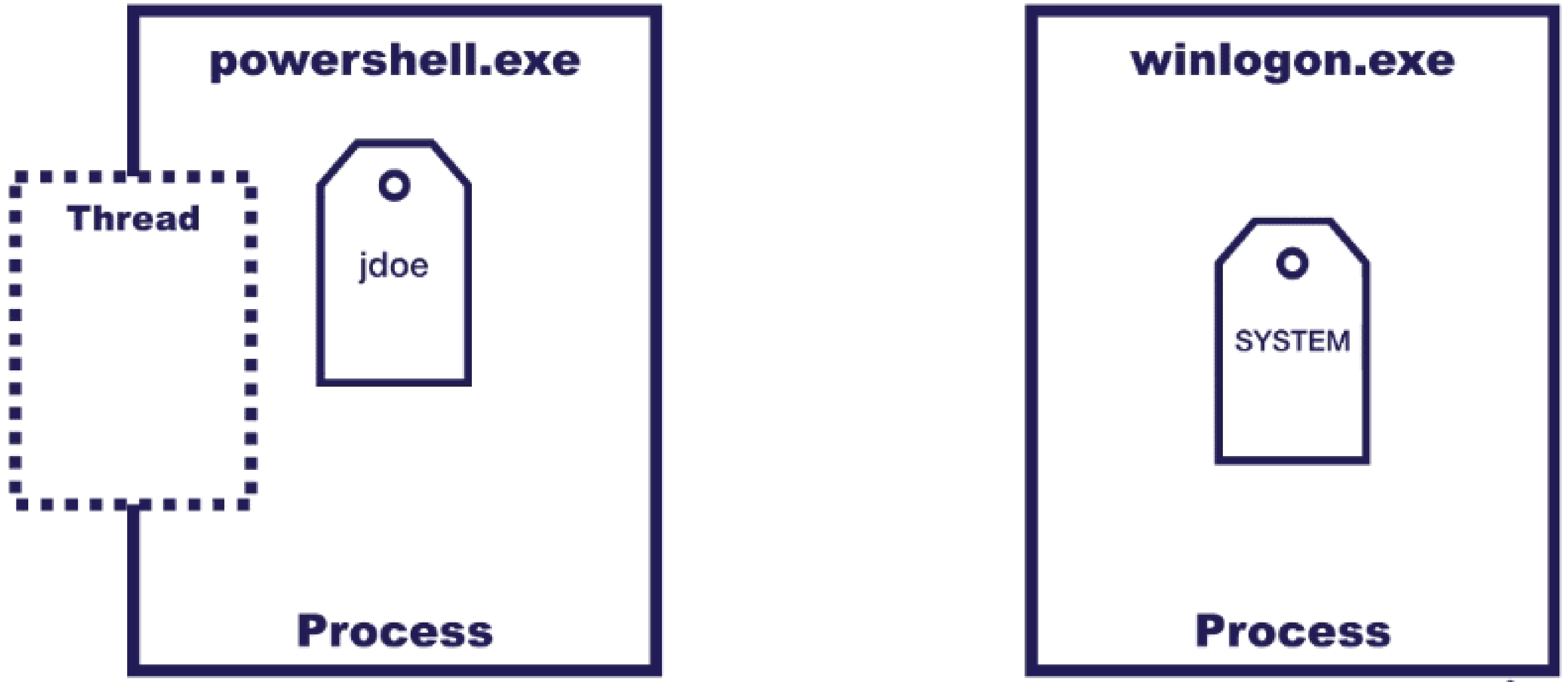
Windows Authentication Overview



Get-System

- One common attacker technique: elevate to a SYSTEM context for specific post-ex actions
- **Method 1:**
  - Enable SeDebugPrivilege, open a handle to a SYSTEM process with OpenProcessToken(), duplication the token with DuplicateToken(), and set the token to your process with SetThreadToken()
- **Method 2:**
  - Create a named pipe, create a service that uses cmd.exe to echo text to the named pipe, use ImpersonateNamedPipeClient() to impersonate SYSTEM

TOKEN IMPERSONATION/THEFT



Demo

Get-System under the hood

Detecting Token Impersonation

- Some attack tools do not clean up duplicate/created tokens!
- Token impersonation applies to a specific thread, so we can compare process token to thread tokens
  - Looking for weird “anomalies”
- We built a tool (**Get-AccessToken**) that will enumerate current access tokens at a very granular level

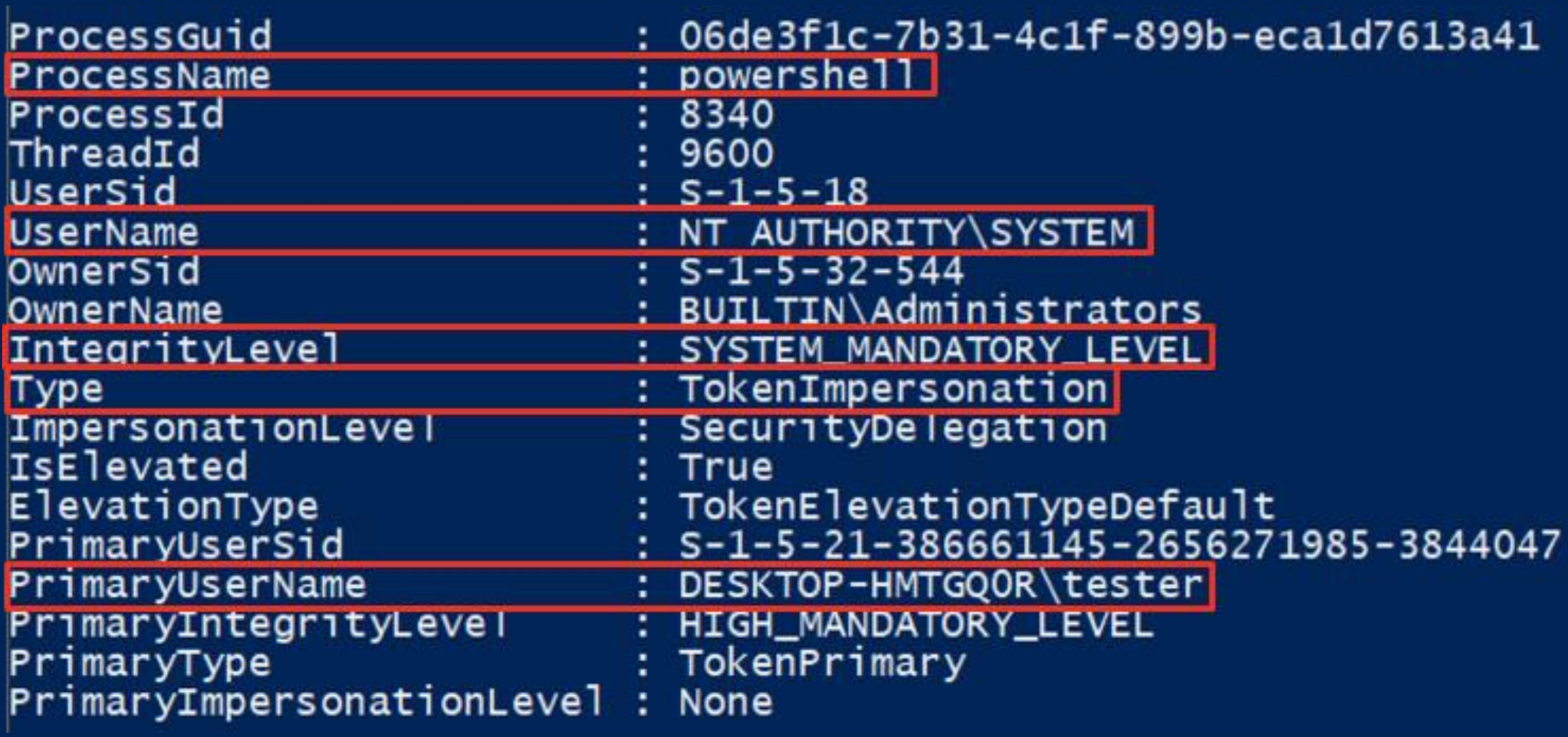
Get-AccessToken

- Enumerate processes/threads (Get-Process)
- **OpenProcess*** - Returns a handle to a process object
- **OpenProcessToken*** - Opens an access token associated with a process
- **OpenThread*** - Returns a handle to a thread object
- **OpenThreadToken*** - Opens an access token associated with a thread
- **GetTokenInformation*** - Retrieves a specified type of information about an access token

* uses PSReflect for Win32 API access!

PowerShell Conference EU 2018

Get-AccessToken vs Get-System



```
ProcessGuid                : 06de3f1c-7b31-4c1f-899b-eca1d7613a41
ProcessName                : powershell
ProcessId                  : 8340
ThreadId                   : 9600
UserSid                    : S-1-5-18
UserName                   : NT AUTHORITY\SYSTEM
OwnerSid                   : S-1-5-32-544
OwnerName                  : BUILTIN\Administrators
IntegrityLevel             : SYSTEM_MANDATORY_LEVEL
Type                       : TokenImpersonation
ImpersonationLevel         : SecurityDelegation
IsElevated                 : True
ElevationType              : TokenElevationTypeDefault
PrimaryUserSid             : S-1-5-21-386661145-2656271985-3844047
PrimaryUserName            : DESKTOP-HMTGQOR\tester
PrimaryIntegrityLevel      : HIGH_MANDATORY_LEVEL
PrimaryType                : TokenPrimary
PrimaryImpersonationLevel  : None
```

Demo

Detection of token impersonation with **Get-AccessToken**

PSCONF.EU

2018

Mimikatz

Defensive Enemy #1 :)

Mimikatz Background

- The current de facto blackhat/whitehat hacking tool
  - Written by Benjamin Delpy (@gentilkiwi)
- Best known for extracting passwords from memory though various credential packages, but WAY more than just that!
  - The current best overall command breakdown is Sean Metcalf's "Unofficial Guide to Mimikatz & Command Reference" (https://adsecurity.org/?page_id=1821)
- Weaponized independently (mimikatz.exe), integrated into most open-source remote access tools, and packaged into Invoke-Mimikatz !

Common Mimikatz Modules (part 1)

- **lsadump -** interact w/ local security authority (LSA)
  - ::lsa - extracts AD passwords (on a DC)
  - ::sam - extracts passwords stored in SAM
  - ::secrets - extracts LSA secrets

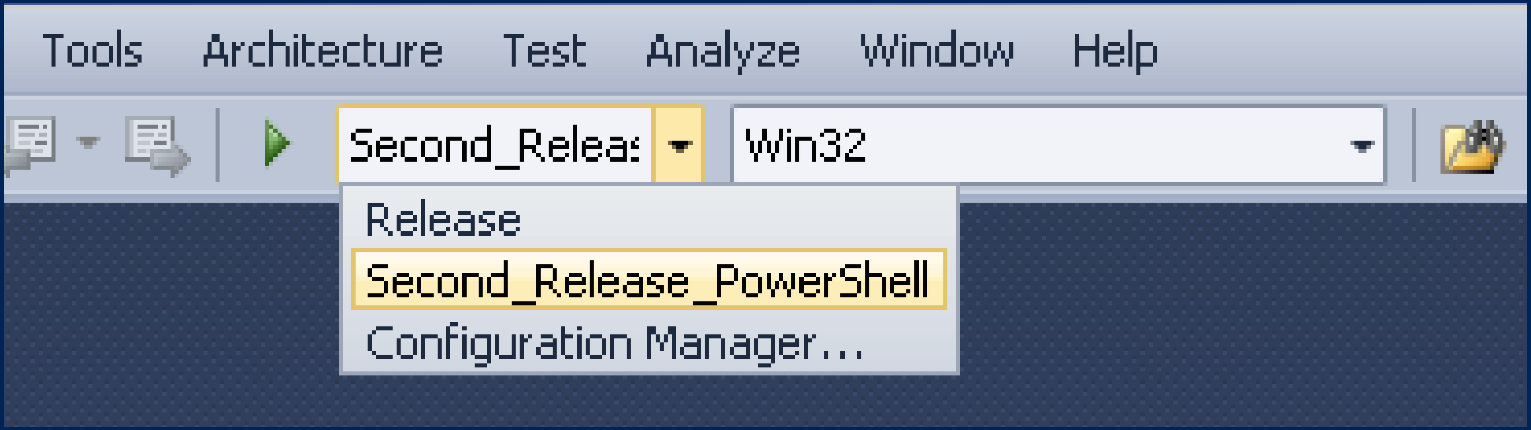
- **sekurlsa** - interacts w/ LSASS
  - ::wdigest - plaintext in-memory creds, fixed as of KB2871997
  - ::tspkg - terminal services credentials
  - ::logonpasswords - pull creds from all available providers
  - ...lots more!

Common Mimikatz Modules (part 2)

- **token** - interact with user tokens
  - ::list - list all available tokens
  - ::elevate - impersonate specific tokens
- **kerberos** - interface with the Kerberos API
  - ::list - list currently registered tickets
  - ::golden - ticket forgery
- **privilege** - enable various user rights
  - ::debug - most common, needed for more “interesting” options

Invoke-Mimikatz

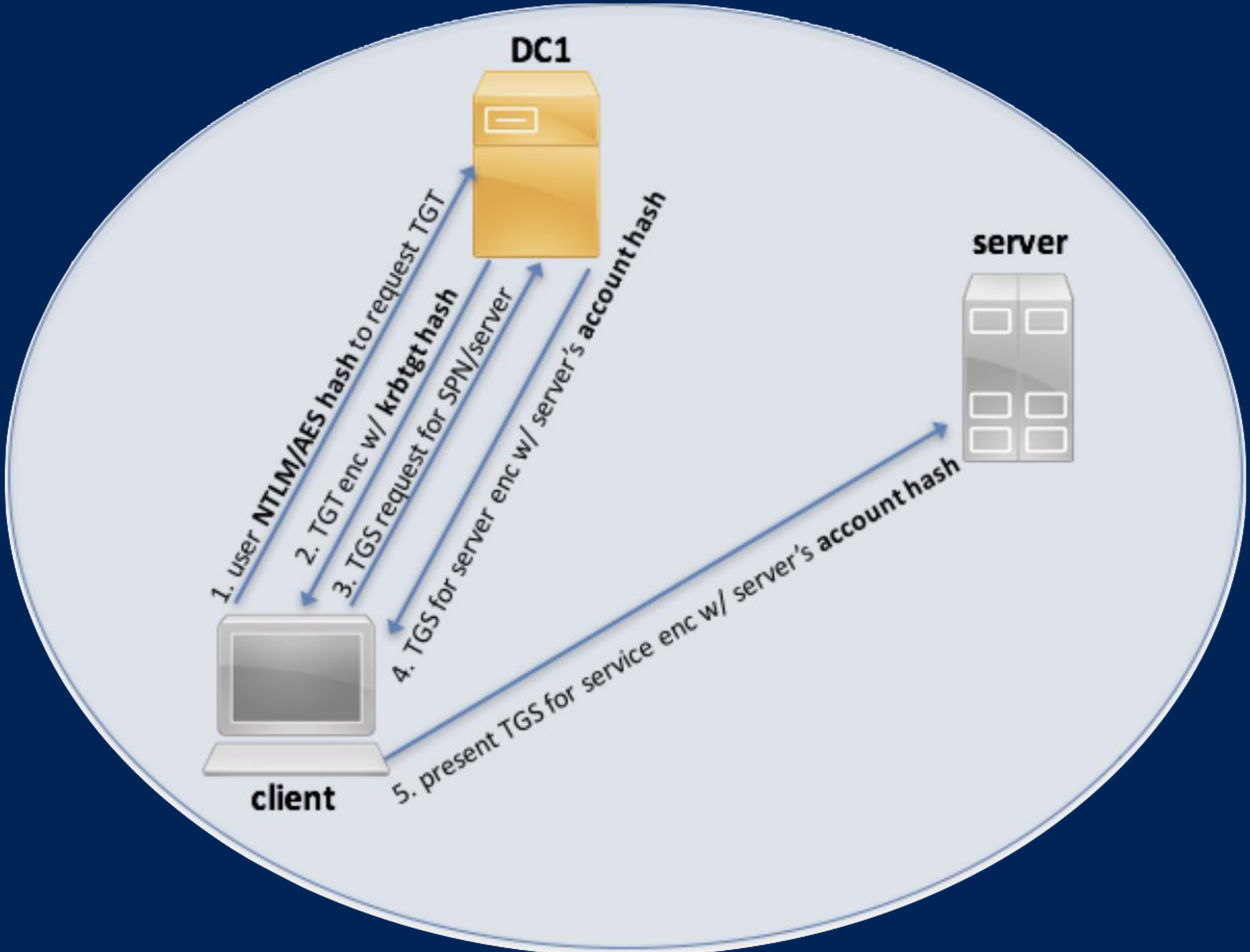
- Written by Joe Bialek while on the Microsoft Office 365 red team
  - Can load Mimikatz completely in memory through a customized PE loader
- The Mimikatz Visual Studio project has an Invoke-Mimikatz build target:



Kerberos: In English (hopefully…)

- A client proves who they are to a domain controller using their password/hash and receives a ticket-granting-ticket (TGT) in return
  - This TGT has signed info that proves the user's identity
- To access a given service in a domain the client requests a service ticket (TGS) from the DC by presenting the TGT
- The ending TGS is presented to the target service (which proves that the DC authenticated the client) and the service decides whether to grant or deny access

Kerberos 101


DC1
server
client
1. user **NTLM/AES hash** to request TGT
2. TGT enc w/ **krbtgt hash**
3. TGS request for SPN/server
4. TGS for server enc w/ server's **account hash**
5. present TGS for service enc w/ server's **account hash**

Forged Kerberos Tickets

- Ticket granting tickets (TGT) are ultimately protected by the hash of the **krbtgt** account
- If we can compromise this hash, we can forge our own Kerberos ticket granting tickets!
  - These are Mimikatz "**golden tickets**"

Kiwi's

GOLDEN TICKET

GREETINGS TO YOU, THE LUCKY FINDER OF THIS GOLDEN TICKET!...

PRESENT THIS TICKET TO THE KEY DISTRIBUTION CENTER WHEN YOU WANT...
YOU CAN BE LATE, AND YOU MAY BRING WITH YOU **mimikatz**!

In your wildest dreams you could not imagine the marvelous RIGTHS that await YOU!

Demo

Invoke-Mimikatz functionality

Extracting passwords, ticket forgery

PowerShell Scriptblock Autologging

- Introduced in PSv5, scriptblock autologging automatically logs any scriptblock execution that contains a predetermined “dirty word” deemed suspicious

- Dirty words can be dumped with the following command:
  - `[ScriptBlock].GetField('signatures', 'NonPublic, Static').GetValue($null)`

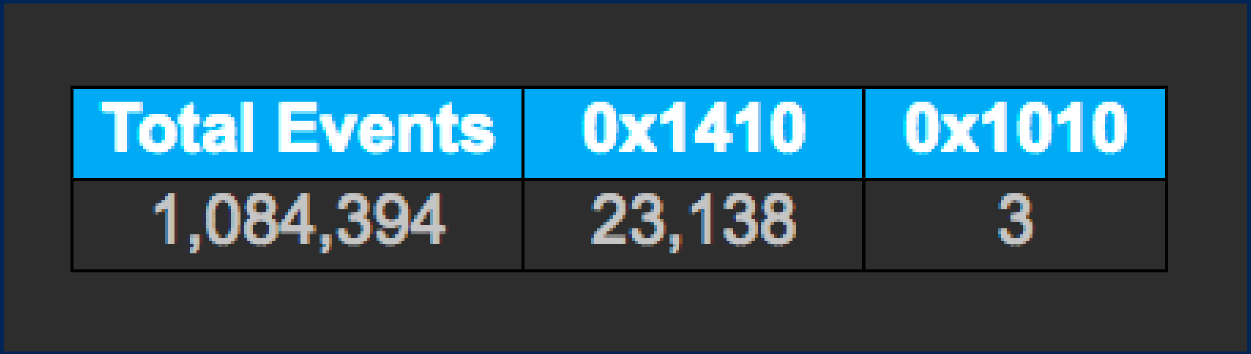
PowerShell Scriptblock Autologging

- Logged to the **Microsoft-Windows-PowerShell/Operational** log under event ID 4104 with the “Warning” error level.

```
Get-WinEvent -LogName Microsoft-Windows-PowerShell/Operational -FilterXPath '*[System[EventID=4104 and Level=3]]'
```

Detecting Credential Dumping

- @Cyb3rWard0g has an excellent blog post* where he analyzes calls to OpenProcess against lsass
- Configure Sysmon to monitor Process Access Events (Event ID 10) for lsass.exe
  - Look for the "GrantedAccess" field of 0x1410, 0x143A, or 0x1010



Total Events	0x1410	0x1010
1,084,394	23,138	3

*https://cyberwardog.blogspot.com/2017/03/chronicles-of-threat-hunter-hunting-for_22.html

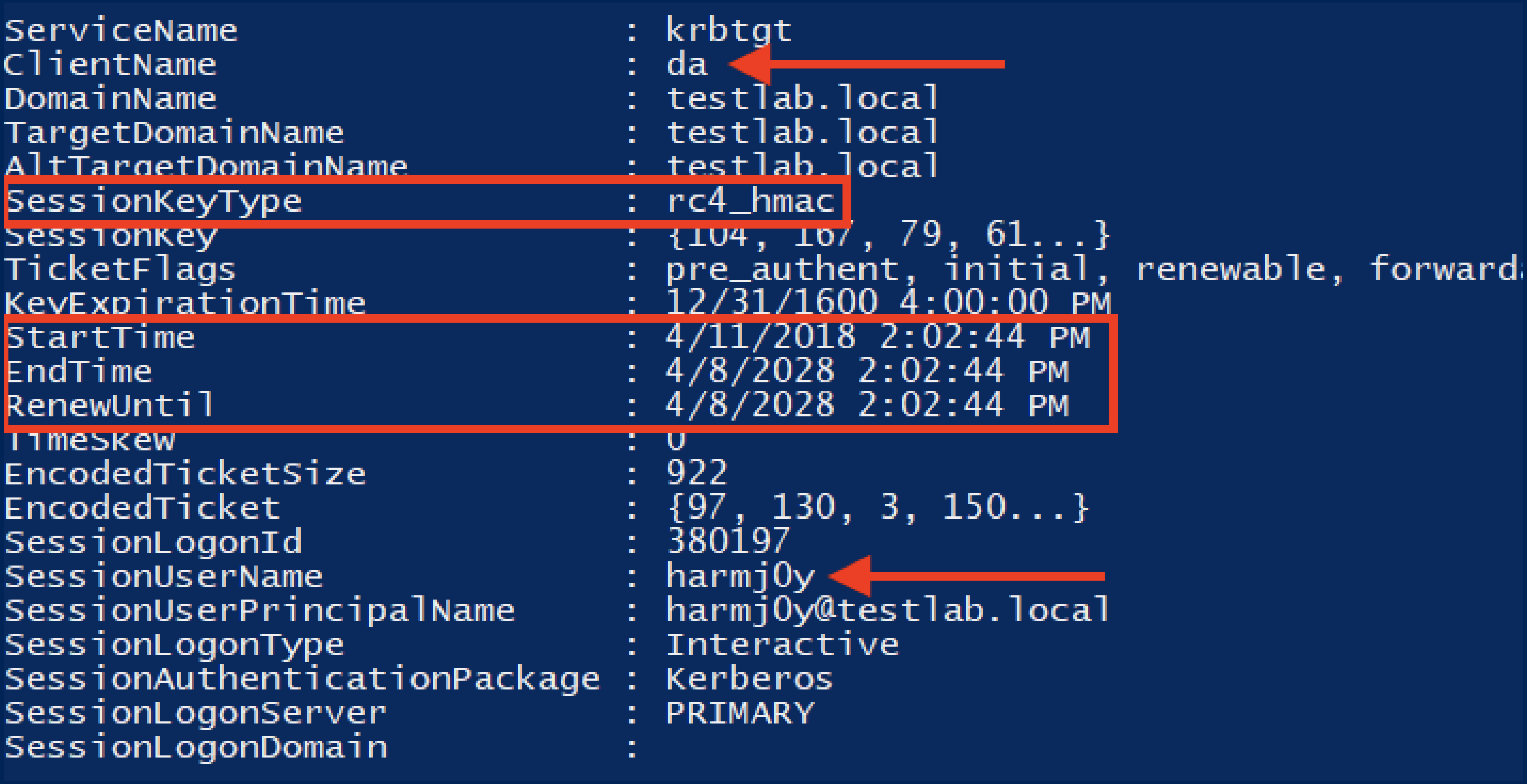
Detecting Ticket Forgery

- We needed a tool that enumerates Kerberos ticket information at a granular level, so we built one!
  - **Get-KerberosTicketGrantingTicket.ps1**
- Stock Mimikatz forged tickets have a few anomalies:
  - 10 year lifetime, RC4/NTLM encryption, others
  - But these can be modified!
- Our “general” heuristic:
  - Check if the session *username* matches the *clientname* pulled from the ticket-granting-ticket

Get-KerberosTicketGrantingTicket

- Enumerate LSA Logon Sessions
  - **LsaEnumerateLogonSessions** - Returns a handle to an array of session data structures.
  - **LsaGetLogonSessionData** - Queries each session handle for its associated information (logon type, user, etc.).
- Request each Logon Session's Ticket Granting Ticket
  - **LsaRegisterLogonProcess** - Establishes a connection to the Local Security Authority Server.
  - **LsaCallAuthenticationPackage** - Calls a specified function implemented by an authentication package (Kerberos).
  - **LsaDeregisterLogonProcess** - Closes the connection to the Local Security Authority Server.

Forged TGT



```
ServiceName                    : krbtgt
ClientName                     : da
DomainName                     : testlab.local
TargetDomainName               : testlab.local
AltTargetDomainName            : testlab.local
SessionKeyType                 : rc4_hmac
SessionKey                     : {104, 167, 79, 61...}
TicketFlags                    : pre_authent, initial, renewable, forward
KeyExpirationTime              : 12/31/1600 4:00:00 PM
StartTime                      : 4/11/2018 2:02:44 PM
EndTime                        : 4/8/2028 2:02:44 PM
RenewUntil                     : 4/8/2028 2:02:44 PM
TimeSkew                       : 0
EncodedTicketSize              : 922
EncodedTicket                  : {97, 130, 3, 150...}
SessionLogonId                 : 380197
SessionUserName                : harmj0y
SessionUserPrincipalName       : harmj0y@testlab.local
SessionLogonType               : Interactive
SessionAuthenticationPackage   : Kerberos
SessionLogonServer             : PRIMARY
SessionLogonDomain             :
```

Demo

Detecting forged Kerberos Tickets

PSCONF.EU

2018

PowerShell Without powershell.exe

PowerShell != powershell.exe !

- PowerShell == System.Management.Automation.(ni.)dll
- C# can easily be used to build a PowerShell pipeline runner in a dozen lines of code
- There are a number of offensive-oriented projects that implement this approach:
  - SharpPick
  - @jaredhaight’s PSAttack project
  - @Cneelis‘s p0wnedShell
  - @ben0xa’s NPS project

Demo

Building a simple C# PowerShell Runner

UnmanagedPowerShell

- @tifkin_'s response to the "can PowerShell run without powershell.exe" problem
- **UnmanagedPowerShell** provides the ability to run PowerShell code in an *unmanaged* (C/C++/non-.NET) process
  - Loads up the .NET Common Language Runtime (CLR) in the current process (needs code injection for a foreign process)
  - Grabs a pointer to the CLR AppDomain
  - Loads a custom C# assembly that runs PowerShell

Invoke-PSInject.ps1

- **Invoke-PSInject.ps1:**
  - takes a PowerShell script block (base64-encoded)
  - patches the decoded logic into the architecture appropriate ReflectivePick.dll
  - injects the result into a specified ProcessID

- Lets you super-easily run PowerShell code in any process you want!
  - This is what Empire’s process injection capability is built on!

Demo

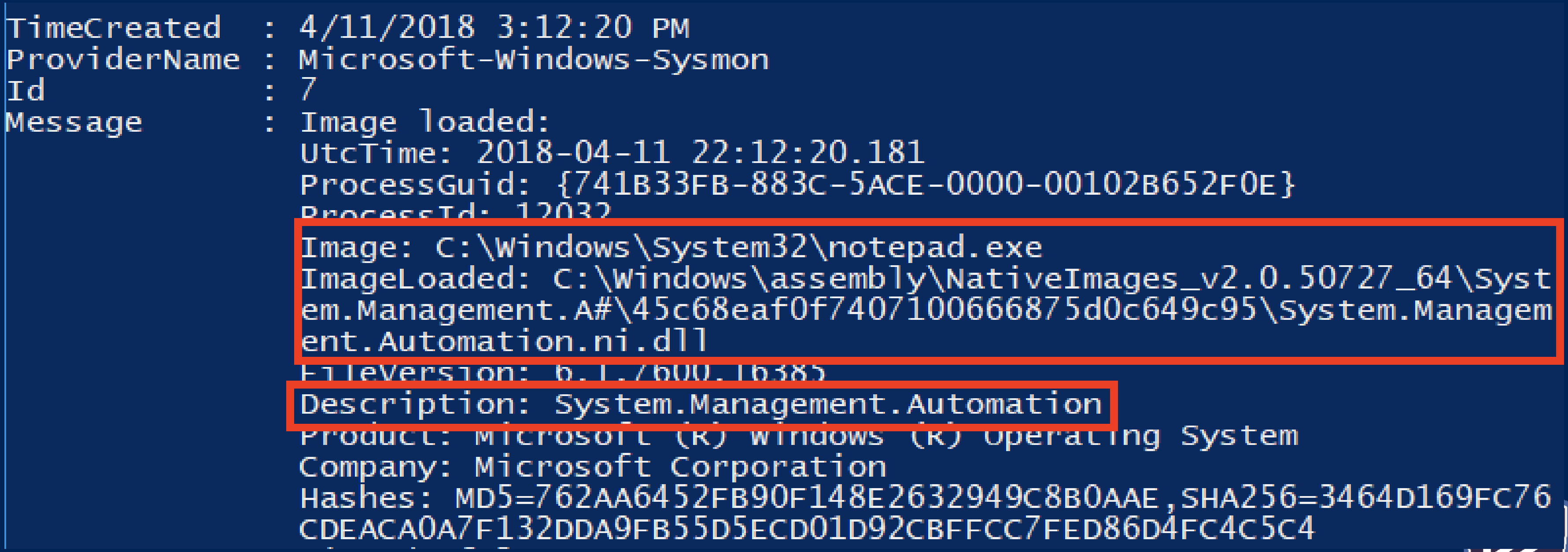
Injecting PowerShell

Detecting PSInject: WMI load events

- WMI event subscription query:
  - `SELECT FileName, ProcessID FROM Win32_ModuleLoadTrace WHERE FileName LIKE "%System.Management.Automation%.dll"`
- Possible actions:
  - **LogFileEventConsumer** with a customized event log
  - **ActiveScriptEventConsumer** to trigger custom .VBS handling code
  - **CommandLineEventConsumer** to do something like auto-dumping the target process' memory

Detecting PSInject: Sysmon

- If you start Sysmon with the **-l** argument, it will log module loads (Sysmon event ID 7)
- Does this look suspicious?

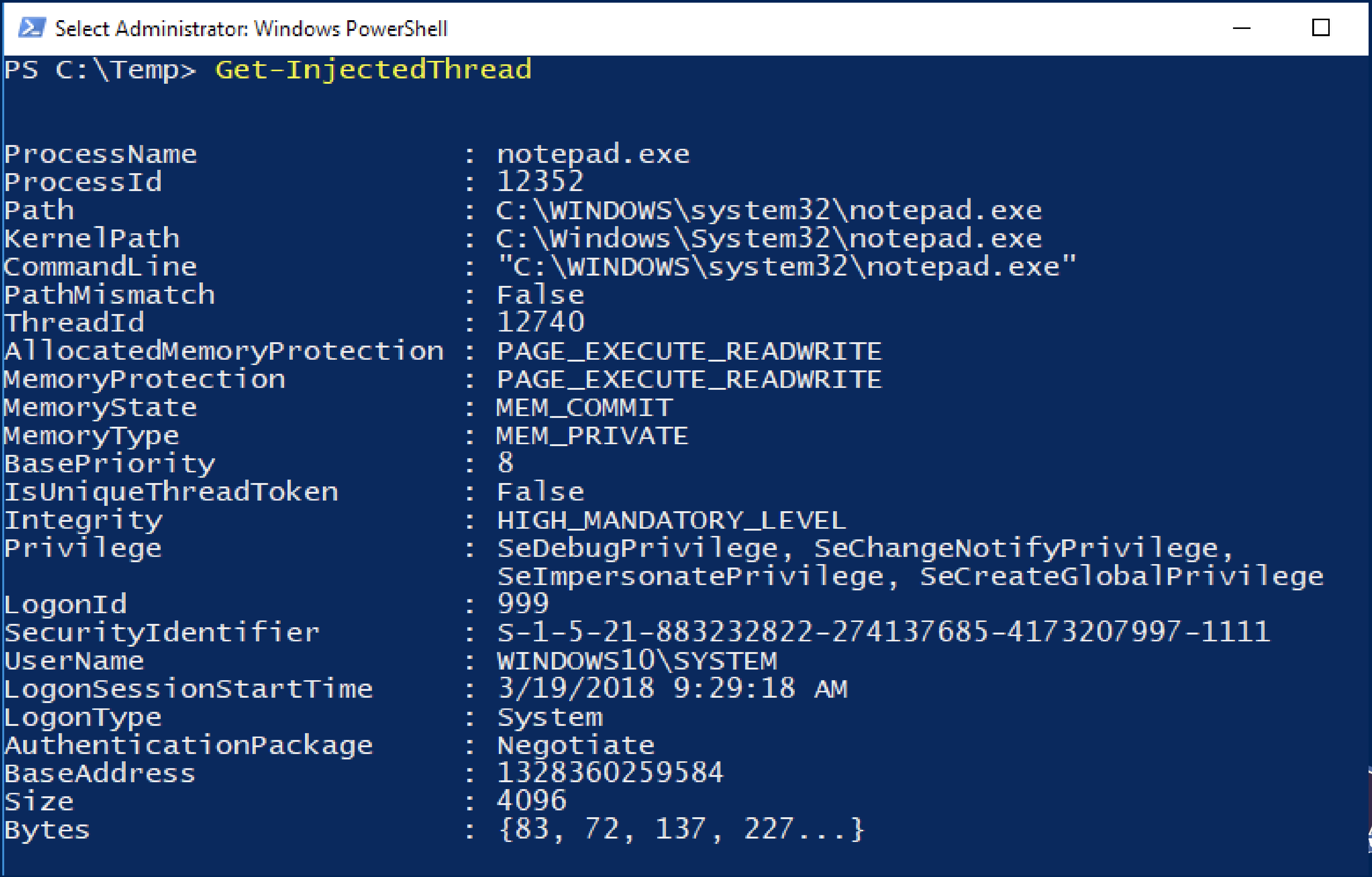


```
TimeCreated  : 4/11/2018 3:12:20 PM
ProviderName : Microsoft-Windows-Sysmon
Id           : 7
Message      : Image loaded:
               UtcTime: 2018-04-11 22:12:20.181
               ProcessGuid: {741B33FB-883C-5ACE-0000-00102B652F0E}
               ProcessId: 12032
               Image: C:\Windows\System32\notepad.exe
               ImageLoaded: C:\Windows\assembly\NativeImages_v2.0.50727_64\Syst
               em.Management.A#\45c68eaf0f7407100668 75d0c649c95\System.Managem
               ent.Automation.ni.dll
               FileVersion: 6.1.7600.16385
               Description: System.Management.Automation
               Product: Microsoft (R) Windows (R) Operating System
               Company: Microsoft Corporation
               Hashes: MD5=762AA6452FB90F148E2632949C8B0AAE,SHA256=3464D169FC76
               CDEACA0A7F132DDA9FB55D5ECD01D92CBFFCC7FED86D4FC4C5C4
```

Detecting Generic Injection

- Our detection relies on certain assumptions:
  - To execute, code must have an associated thread
  - Code executed by a thread **should** live on disk somewhere
- **Get-InjectedThread**’s detection process:
  - Iterate through threads
  - Identify each thread’s base memory address
  - Query the memory page that the base address belongs to
  - Ensure that the memory page is currently committed (MEM_COMMIT)
  - Flag, if memory page contents are not from disk (!MEM_IMAGE)

PSInject vs. Get-InjectedThread



Select Administrator: Windows PowerShell

```
PS C:\Temp> Get-InjectedThread


ProcessName                 : notepad.exe
ProcessId                   : 12352
Path                        : C:\WINDOWS\system32\notepad.exe
KernelPath                  : C:\Windows\System32\notepad.exe
CommandLine                 : "C:\WINDOWS\system32\notepad.exe"
PathMismatch                : False
ThreadId                    : 12740
AllocatedMemoryProtection : PAGE_EXECUTE_READWRITE
MemoryProtection            : PAGE_EXECUTE_READWRITE
MemoryState                 : MEM_COMMIT
MemoryType                  : MEM_PRIVATE
BasePriority                : 8
IsUniqueThreadToken         : False
Integrity                   : HIGH_MANDATORY_LEVEL
Privilege                   : SeDebugPrivilege, SeChangeNotifyPrivilege,
                              SeImpersonatePrivilege, SeCreateGlobalPrivilege
LogonId                     : 999
SecurityIdentifier          : S-1-5-21-883232822-274137685-4173207997-1111
UserName                    : WINDOWS10\SYSTEM
LogonSessionStartTime       : 3/19/2018 9:29:18 AM
LogonType                   : System
AuthenticationPackage       : Negotiate
BaseAddress                 : 1328360259584
Size                        : 4096
Bytes                       : {83, 72, 137, 227...}
```

Demo

Hunting for Injection

PSCONF.EU

2018

Alternate PowerShell Hosts

“Official” Alternate PowerShell Hosts

- There are a number of signed/”official” Microsoft binaries that host System.Management.Automation

- Abuse of any of these binaries can grant you:
  - Avoiding lockdown of powershell.exe
  - Bypass of application whitelisting policies that allow anything signed by Microsoft (most of them)
  - *Might* be a constrained language mode bypass
  - Avoiding of command line logging and some Sysmon logging

Searching for “Official” hosts

- So how can you go about finding these hosts?

- **Characteristic 1:**
  - These binaries are almost always C#/.NET .exes/.dlls

- **Characteristic 2:**
  - These binaries have System.Management.Automation.dll as a referenced assembly

- **Characteristic 3:**
  - These may not always be “built in” binaries

Demo

Searching for and Abusing
Alternate PowerShell Hosts

PSCONF.EU

2018

Bonus: Subversive Profiles

PowerShell Profiles

- Scripts that run every time an “official” PowerShell host (meaning powershell.exe/powershell_ise.exe) starts
  - Meant for shell customization, not loaded with remoting!
- Profiles can be subverted with malicious proxy functionality!
- More information:
  - http://www.exploit-monday.com/2015/11/investigating-subversive-powershell.html

PowerShell Profile Locations

AllUsersAllHosts	`%windir%\System32\WindowsPowerShell\v1.0\profile.ps1`
AllUsersAllHosts (WoW64)	`%windir%\SysWOW64\WindowsPowerShell\v1.0\profile.ps1`
AllUsersCurrentHost	`%windir%\System32\WindowsPowerShell\v1.0\Microsoft.PowerShell_profile.ps1`
AllUsersCurrentHost (ISE)	`%windir%\System32\WindowsPowerShell\v1.0\Microsoft.PowerShellISE_profile.ps1`
AllUsersCurrentHost (WoW64)	`%windir%\SysWOW64\WindowsPowerShell\v1.0\Microsoft.PowerShell_profile.ps1`
AllUsersCurrentHost (ISE - WoW64)	`%windir%\SysWOW64\WindowsPowerShell\v1.0\Microsoft.PowerShellISE_profile.ps1`
CurrentUserAllHosts	`%homedrive%%homepath%\[My ]Documents\WindowsPowerShell\profile.ps1`
CurrentUserCurrentHost	`%homedrive%%homepath%\[My ]Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1`
CurrentUserCurrentHost (ISE)	`%homedrive%%homepath%\[My ]Documents\WindowsPowerShell\Microsoft.PowerShellISE_profile.ps1`

Lab: Subversive Profiles

- Build a subversive profile that hides any powershell.exe instances from **Get-Process**
  - Check out the “call operator”!

- (Bonus) food for thought:
  - How would you write a malicious **Get-Credential** proxy?
  - How would you use a subversive profile for lateral movement?

Summary

- There's lots of offensive PowerShell out there!
- There are ways to detect or mitigate the vast majority of the public offensive toolsets!
- You can't detect what you're not aware of
  - Play with these tools hands on in your environment and work out detections that work for you!

PowerShell Conference EU 2018

Questions?

PSCONF.EU