Porting Binary PowerShell Modules to Linux/OSX

Jared Atkinson - Specter Ops

Open Source PowerShell!!

Agenda

- PowerForensics Introduction
- Integrating .NET Core
- Cross Platform APIs
- Abstracting API calls
- Dealing with multiple assemblies
- Running Binary Modules Remotely

What is PowerForensics

- PowerShell based Forensic Toolkit
- Allows inspection of NTFS & FAT data structures
  - MFT/FAT
  - UsnJrnl
- Create an event timeline based on forensic artifacts
  - File Operations
  - Program Execution
  - User Activity
- Deleted File Recovery
- Upcoming HFS+ and Ext4 support (Open Source PS ☺)

Project Goals

- Transparent User Interaction
- Major File System Support
  - NTFS, FAT32, HFS+, EXT3/4
- Able to be run remotely (Live Response)
  - Does not have to be installed on every endpoint
- Supports Open Source PowerShell
  - Linux/MacOS compatibility (Image Analysis)
  - Nano Server compatibility (Challenged by Mattifestation)
- Compatible with PowerShell Version 2.0+ (Tanium)
  - Windows 7 compatibility

Problem 1: Integrating .NET Core

- Current State:
  - PowerForensics builds on .NET 3.5
- Desired State:
  - PowerForensics runs on all PowerShell versions
    - Windows PowerShell (v2+)
    - Open Source PowerShell (MacOS and Linux)
    - Core PowerShell (Nano Server)
  - One set of source code

Handling Caveats

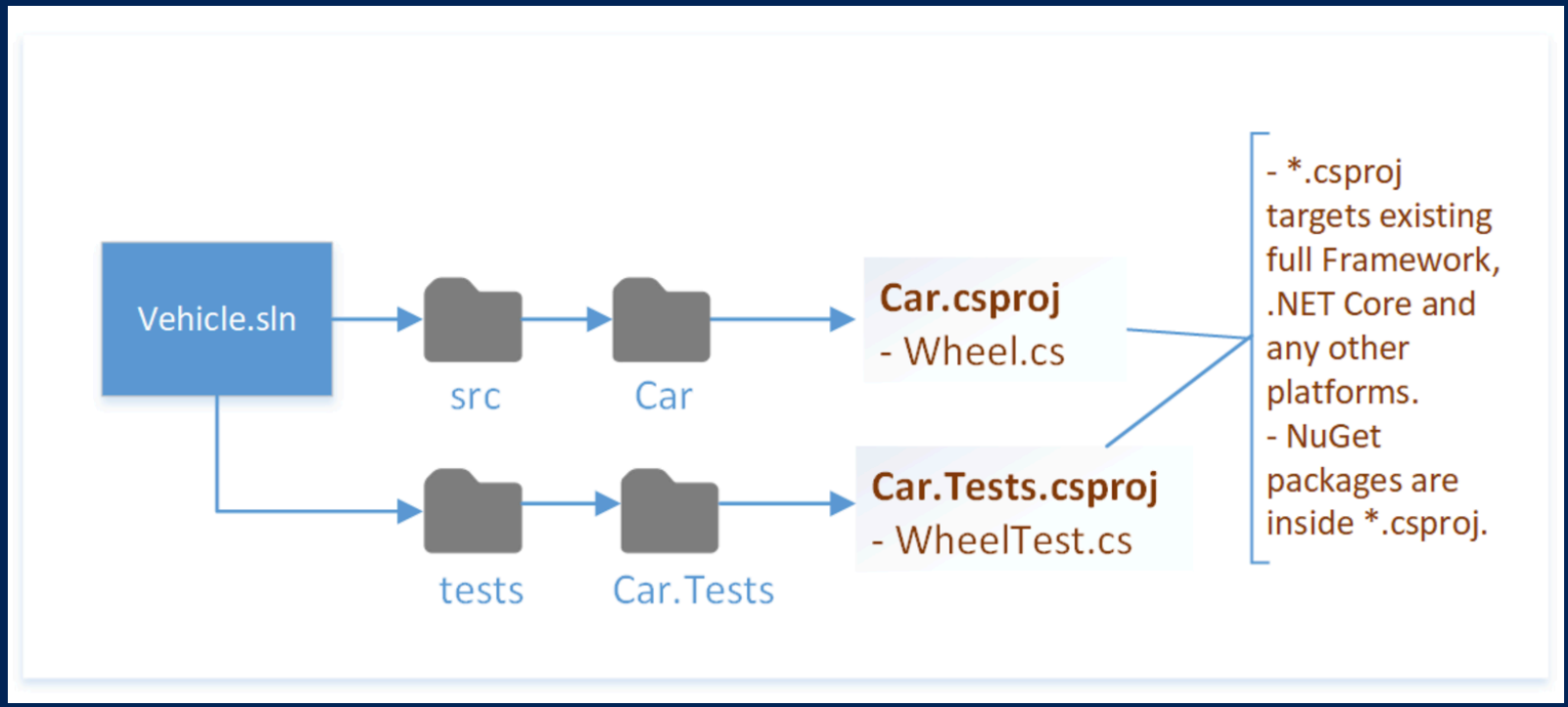
	2.0	3.0	4.0	5.x	Core 5.x	6.x
Windows 7/2008 R2	Default	Supported	Supported	Supported	Not Supported	Not Supported
Windows 8/2012		Default	Not Supported	Supported	Not Supported	Not Supported
Windows 8.1/2012R2			Default	Supported	Not Supported	Supported
Windows 10/2016				Defuault	Not Supported	Supported
Windows Nano					Default	Supported
Windows IoT					Default	Supported
Linux/OS X						Supported

12

Image courtesy of Carlos Perez - @darkoperator

Transitioning from Full CLR to Core CLR

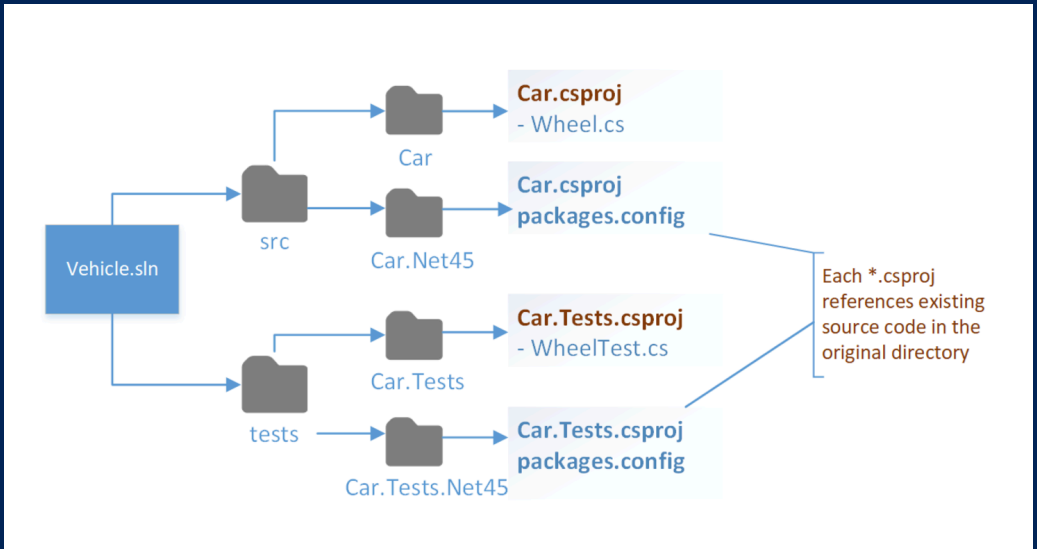
- If you have no need for supporting older versions of PS
  - Replace old project with a .NET Core project



https://docs.microsoft.com/en-us/dotnet/articles/core/porting/project-structure#support-pcl

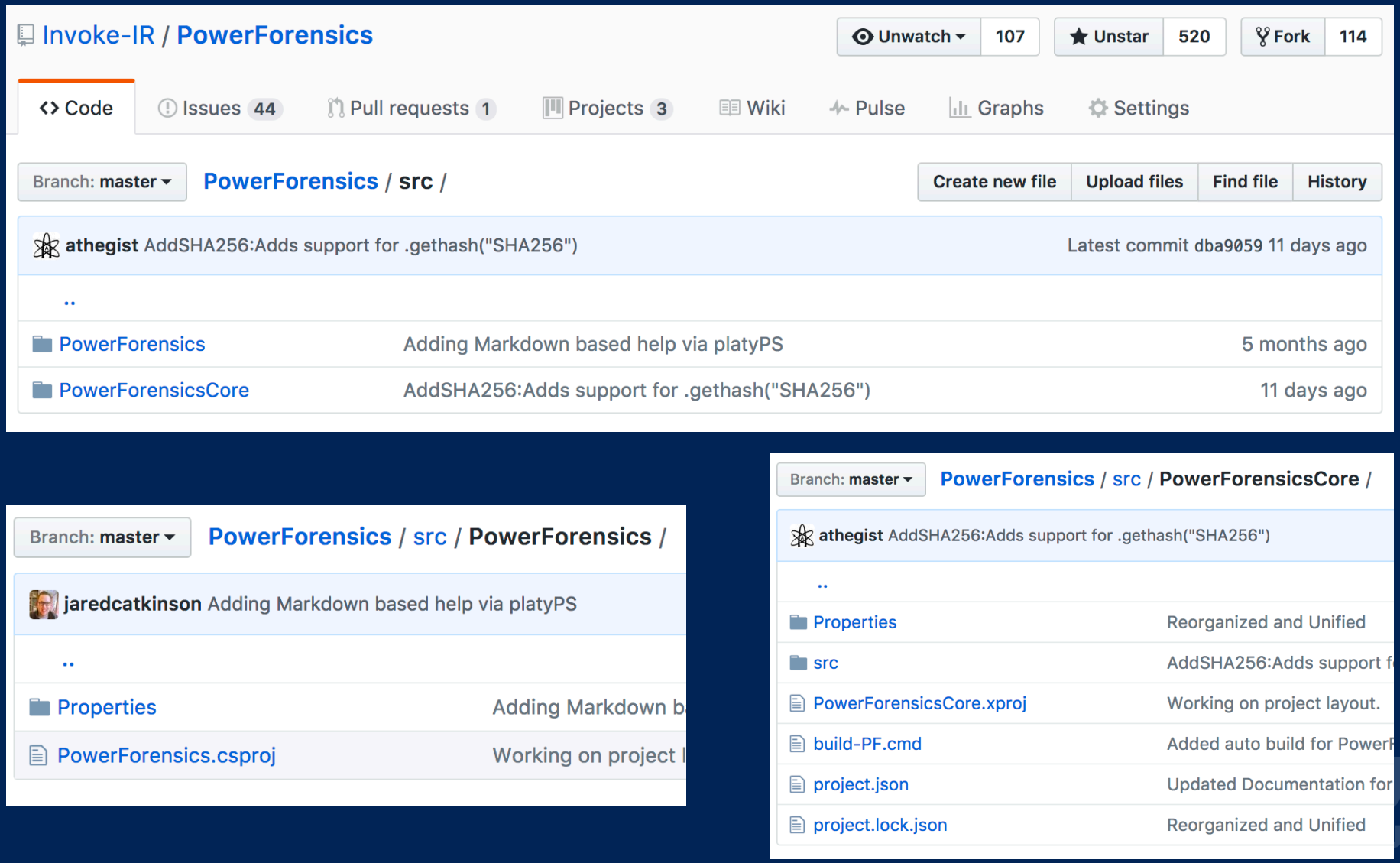
Organizing My Project

- Two projects
  - PowerForensics (.NET Framework 3.5 or PowerShell v2)
  - PowerForensicsCore (.NET Core)
- One set of source code
  - PowerForensicsCore/src/*
  - Make changes to one project, affect all projects



https://docs.microsoft.com/en-us/dotnet/articles/core/porting/project-structure#support-pcl

Solution - PowerForensics Organization



Invoke-IR / PowerForensics

Unwatch 107 | Unstar 520 | Fork 114

Code | Issues 44 | Pull requests 1 | Projects 3 | Wiki | Pulse | Graphs | Settings

Branch: master — PowerForensics / src / — Create new file | Upload files | Find file | History

athegist AddSHA256:Adds support for .gethash("SHA256") — Latest commit dba9059 11 days ago

..		
PowerForensics	Adding Markdown based help via platyPS	5 months ago
PowerForensicsCore	AddSHA256:Adds support for .gethash("SHA256")	11 days ago

Branch: master — PowerForensics / src / PowerForensics /

jaredcatkinson Adding Markdown based help via platyPS

..	
Properties	Adding Markdown b
PowerForensics.csproj	Working on project l

Branch: master — PowerForensics / src / PowerForensicsCore /

athegist AddSHA256:Adds support for .gethash("SHA256")

..	
Properties	Reorganized and Unified
src	AddSHA256:Adds support f
PowerForensicsCore.xproj	Working on project layout.
build-PF.cmd	Added auto build for PowerF
project.json	Updated Documentation for
project.lock.json	Reorganized and Unified

Problem 2:
Finding the *nix “CreateFile"

- Current State:
  - PowerForensics uses CreateFile to access raw devices
  - Know that Open is the *nix equivalent to CreateFile
- Desired State:
  - Create a P/Invoke signature for Open (Linux APIs)
  - PowerForensics can read files on Linux

Calling APIs

C# provides a couple methods for calling Win32 APIs

- Platform Invoke (P/Invoke)
  - Allows managed code to call unmanaged functions
  - Provide the C# compiler with:
    - Declaration of the unmanaged function
    - Description of how to marshal parameters and return value
  - PowerShell allows P/Invoke via the Add-Type cmdlet

Marshalling types - https://msdn.microsoft.com/en-us/library/aa446538.aspx

Platform Invoke (P/Invoke)

- Examples of P/Invoke can be found at pinvoke.net

Sends a string to the debugger for display.

C# Signature:

```
[DllImport("kernel32.dll")]
static extern void OutputDebugString(string lpOutputString);
```

- If no example exists build one based on MSDN docs

Syntax

C++

```
void WINAPI OutputDebugString(
  _In_opt_ LPCTSTR lpOutputString
);
```

Parameters

lpOutputString [in, optional]
The null-terminated string to be displayed.

https://msdn.microsoft.com/en-us/library/windows/desktop/aa363362(v=vs.85).aspx

P/Invoke

https://youtu.be/Dt63hIwAwgE

Chasing Down the Implementation

- Thinking about Open
  - Open is used to read files on *nix
  - PowerShell Core can read file contents on *nix
  - Let's not reinvent the wheel
- Microsoft's Implementation
  - Github
    - https://github.com/dotnet/corefx
    - https://github.com/powershell/powershell
  - dnSpy
    - Decompile C# Assembly
    - System.Reflection.Assembly - Location Property

dnSpy

https://youtu.be/zsMjMmnGNfQ

Open API Implementation

```
using System.Runtime.InteropServices;
using Microsoft.Win32.SafeHandles;

internal static partial class Interop
{
    internal static partial class Sys
    {
        [DllImport(Libraries.SystemNative, EntryPoint = "SystemNative_Open", SetLastError = true)]
        internal static extern SafeFileHandle Open(string filename, OpenFlags flags, int mode);
    }
}
```

Solution - File Handle Example

- CreateFile (Windows)

```
[DllImport("kernel32.dll", CharSet = CharSet.Ansi, SetLastError = true)]
internal static extern SafeFileHandle CreateFile
    (
        string fileName,
        [MarshalAs(UnmanagedType.U4)] FileAccess fileAccess,
        [MarshalAs(UnmanagedType.U4)] FileShare fileShare,
        IntPtr securityAttributes,
        [MarshalAs(UnmanagedType.U4)] FileMode creationDisposition,
        int flags,
        IntPtr template
    );
```

- Open (*nix)

```
[DllImport("System.Native", EntryPoint = "SystemNative_Open", SetLastError = true)]
internal static extern SafeFileHandle Open
    (
        string filename,
        OpenFlags flags,
        int mode
    );
```

CreateFile - https://msdn.microsoft.com/en-us/library/windows/desktop/aa363858(v=vs.85).aspx
Open - http://www.tutorialspoint.com/unix_system_calls/open.htm

Problem 3: Abstracting APIs

- Current State:
  - P/Invoke for CreateFile (Windows) and Open (*nix)
- Desired State:
  - Code selects the correct API based on the environment
    - Cross platform functionality abstracted to the lowest level
  - Users shouldn't worry about what OS API to use
  - Contributors shouldn't worry about code architecture

Preprocessing Directive

- Gives instructions to the compiler to ‘preprocess’ before compilation begins
  - #if - allows testing a symbol
  - #else - allows a compound conditional directive w/ #if
  - #endif - specifies the end of a conditional directive

Branch: master ▾ **PowerForensics** / src / PowerForensicsCore / **project.json**

jaredcatkinson Updated Documentation for ReadTheDocs

1 contributor

21 lines (21 sloc) | 371 Bytes

```
1  {
2    "version": "1.0.0-*",
3    "scripts": {
4      "postcompile": "build-PF.cmd"
5    },
6    "dependencies": {
7      "NETStandard.Library": "1.6.0"
8    },
9    "frameworks": {
10     "netstandard1.6": {
11       "imports": "dnxcore50"
12     "buildOptions": {
13       "define": [ "CORECLR" ],
14       "outputName": "PowerForensics"
15     }
```

```
"buildOptions": {
  "define": [ "CORECLR" ],
  "outputName": "PowerForensics"
}
```

Solution - Letting the code decide…

```
internal static FileStream getFileStream(string fileName)
{
    SafeFileHandle hDevice = null;
```

```
#if CORECLR
    if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
    {
        // Get Handle to specified Volume/File/Directory
        hDevice = NativeMethods.CreateFile(
            fileName,
            FileAccess.Read,
            FileShare.Write | FileShare.Read | FileShare.Delete,
            IntPtr.Zero,
            FileMode.Open,
            NativeMethods.FILE_FLAG_BACKUP_SEMANTICS,
            IntPtr.Zero
        );
    }
    else
    {
        hDevice = NativeMethods.Open(
            fileName,
            NativeMethods.OpenFlags.O_RDONLY,
            3
        );
    }
```

PS Core

```
#else
    // Get Handle to specified Volume/File/Directory
    hDevice = NativeMethods.CreateFile(
        fileName,
        FileAccess.Read,
        FileShare.Write | FileShare.Read | FileShare.Delete,
        IntPtr.Zero,
        FileMode.Open,
        NativeMethods.FILE_FLAG_BACKUP_SEMANTICS,
        IntPtr.Zero
    );
#endif
```

Win PS

Solution – Letting the code decide

Problem 4: Selecting the Correct Assembly

- Current State:
  - Three versions of PowerForensics exist
    - PowerForensics (PSv2)
    - PowerForensicsCore (Core PS)
    - PowerForensics Portable (Remote PowerForensics)
- Desired State:
  - The correct version of PowerForeniscs is automatically selected for the user

Project Properties

Solution - Loading the Correct Assembly

```
# Import the appropriate nested binary module based on the current PowerShell version
$binaryModuleRoot = $PSModuleRoot


if (($PSVersionTable.Keys -contains "PSEdition") -and ($PSVersionTable.PSEdition -ne 'Desktop')) {
    $binaryModuleRoot = Join-Path -Path $PSModuleRoot -ChildPath 'lib\coreclr'
}
else
{
    $binaryModuleRoot = Join-Path -Path $PSModuleRoot -ChildPath 'lib\PSv2'
}

$binaryModulePath = Join-Path -Path $binaryModuleRoot -ChildPath 'PowerForensics.dll'
$binaryModule = Import-Module -Name $binaryModulePath -PassThru
```

Assemblies are Libraries… Act accordingly

- PowerForensics is a C# library
  - Exposes a .NET API (classes, properties, methods)
- Two ways to approach from PowerShell
  - Cmdlets
    - Binary cmdlets included in the PowerForensics DLL
  - Advanced Functions
    - PowerShell scripts that wrap the exposed PF API
    - Easier for supporting multiple versions
- Casual contributors find scripting a .ps1 easier than writing a C# binary cmdlet

powerforensics.readthedocs.io

PowerForensics - PowerShell Digital Forensics

Developed by @jaredcatkinson

Overview

The purpose of PowerForensics is to provide an all inclusive framework for hard drive forensic analysis. PowerForensics currently supports NTFS and FAT file systems, and work has begun on Extended File System and HFS+ support.

Detailed instructions for installing PowerForensics can be found here.

Cmdlets

Boot Sector

PowerForensics Cross Platform

https://youtu.be/bDqZs4a3eTw

Problem 5: Running Binary Modules Remotely

- Current State:
  - Cross platform and works with multiple PS versions
- Desired State:
  - Module functions can run on a Remote Machine
    - Through PowerShell Remoting

Running Binary Modules Remotely

- Load the assembly in memory on the remote system
  - System.Reflection.Assembly class
    - Load(byte[]) method
      - Loads byte[] in memory as a .NET Assembly
      - Exposes the assembly's public functions
  - Add-PowerForensicsType
    - Checks if assembly is already loaded
    - Selects which version of the assembly to load
    - Loads the assembly to expose public APIs
- Pass local functions to a remote system
- Can execute via Invoke-Command

https://msdn.microsoft.com/en-us/library/h538bck7(v=vs.110).aspx

Add-PowerForensicsType

```
function Add-PowerForensicsType
{
    if (('PowerForensics.BootSector.MasterBootRecord' -as [Type]) -eq $null)
    {
        $pscore = '5L0JYFxFGTj+9r3d994eSfOy6e42TbPpkXTZTUOatCU96F2gpUBbCmzSlp4cDdCtb7mX5RBBgbYQDoEIBcqNAoIccsihKAiWQxEUpICKIAgq
        $lengthcore = 173568
        $psv2 = '5L0JYFxFGTj+9u3ue2+PpHnZdHebptm0NOmSTUOaHqQHPWjLUQq0pcAmbenJ0QDd+pZ7WQ4RFGgr4RCIUKCA3CDIjXIol2BBFEFECqgIgoCicq
        $lengthv2 = 172544

        if (($PSVersionTable.Keys -contains "PSEdition") -and ($PSVersionTable.PSEdition -ne 'Desktop'))
        {
            $EncodedCompressedFile = $pscore
            $Length = $lengthcore
        }
        else
        {
            $EncodedCompressedFile = $psv2
            $Length = $lengthv2
        }

        $DeflatedStream = New-Object IO.Compression.DeflateStream([IO.MemoryStream][Convert]::FromBase64String($EncodedCompress
        $UncompressedFileBytes = New-Object Byte[]($Length)
        $DeflatedStream.Read($UncompressedFileBytes, 0, $Length) | Out-Null
        [Reflection.Assembly]::Load($UncompressedFileBytes) | Out-Null
    }
}
```

Add-PowerForensicType

Running PowerForensics in Memory

```
PS> $s = New-PSSession -ComputerName dc01

PS> Invoke-Command -Session $s -Scriptblock
${function:Add-PowerForensicsType}

PS> $usn = Invoke-Command -Session $s -Scriptblock
${function:Get-ForensicUsnJrnl}
```


http://mikefrobbins.com/2014/03/27/run-a-local-powershell-function-against-a-remote-computer-with-powershell-remoting/

Summary

- Integrating .NET Core
- Cross Platform APIs
- Abstracting API calls
- Dealing with multiple assemblies
- Running Binary Modules Remotely

Questions?

About_Author

- Jared Atkinson
  - Technical Director, Adversary Detection – Specter Ops LLC
  - Former
    - U.S. Air Force Hunt (2011 – 2015)
    - Veris Group's Adaptive Threat Division (2015-2017)
  - 2015 Black Hat Minesweeper Champion
  - Microsoft MVP (Cloud and Datacenter Management/PowerShell)
  - Open Source Developer
    - PowerForensics
    - Uproot IDS
    - WmiEvent
  - Researcher of forensic artifact file formats