

You have the Shell...

 Windows PowerShell

Now where's the

POWER

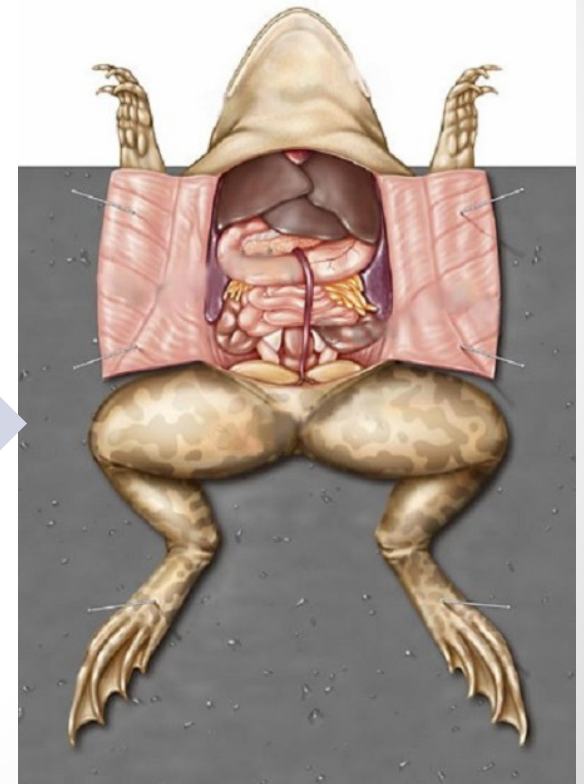
?

About This Class

We are going to discuss how to go from 0-PWNeD by dissecting our tool (script)

I know
nothing

PWNeD



Getting
the
basics

- Oooh gross a dead frog... whose hungry?

More on the Class

- Some code & concept for script comes from Metasploit Minute video.
 - <http://mubix.github.io/blog/2015/01/12/powershell-popups-plus-capture/>
- Reference attack environment
- Code can be found at:
 - git-oaktree
 - <https://goo.gl/HvprVA>

Our Goal...

- Introduce you to PowerShell with plenty of demos/labs to understand the logic behind our reference script and how to go from 0 – code (pwn).
 - We assume you know nothing.
 - Other classes start w/ the precursor that you have the basics.

Our Goal

- Show what PowerShell can do (Red & Blue Team) without importing modules.
- Excite you... Hopefully, to learn more about PowerShell and provide references to resources we have found valuable.
- Give you a program and the skills to build off it... go make it better.

On the Shoulder of Giants

- Windows PowerShell 2.0 for Beginners Training & Overview
 - <https://www.youtube.com/watch?v=3vJvkANKVWA>
- Ben0xA
 - <https://www.youtube.com/watch?v=893NGtZlod8> - Gray Hat PowerShell
 - https://www.youtube.com/watch?v=4X_uBL2YpmA - Practical PowerShell programming for Practical Programmers



- Thanks to everyone that inspired us to get started

On the Shoulder of Giants

- Jared Atkinson @jaredcatkinson
 - PowerForensics
- Matt Graber/ @mattifestation
 - Partial list:
 - Powersploit
 - Exploitmonday.com → blog
- Will Shroeder/ @harmJoy
 - Partial list of projects:
 - Bloodhound
 - powerview
 - <https://www.youtube.com/watch?v=cXWtu-qalSs>
- Carlos Perez @darkoperator



- Thanks to everyone that inspired us to get started

On the Shoulder of Giants

- Lee Holmes @lee_holmes
- Jessica Payne @jepayneMSFT
- Sean Metcalf / @pyrotek3
 - <https://adsecurity.org/?p=2843>
- @enigma0x3
- Casey Smith @subtee
- Chris Cambell @obscuresec
 - <https://www.youtube.com/watch?v=j9HMja>



- Thanks to everyone that inspired us to get started

About Us

- Raymond/@Securemaryland

- Certs I got them – but who cares.
- 20+ years in infosec – man I am old
- This is all mine and not my employers, well kind of. (see name below)
- I have no clue what I am doing.... Just wanted to see if you were reading this 😊

- Octavio/@oaktree__

- Work in IT but hopefully that is relevant without this bullet point.
- This is all mine and not my employers, well kind of. (see name above)
- Very tall
- Avid reader of handyman magazine

• Just two Joe's ... well actually a Ray and an Octavio

What is PowerShell?

IRONIC


“Windows **PowerShell** is a shell developed by Microsoft for purposes of task automation and configuration management. This powerful shell is based on the .NET framework and it includes a command-line shell and a scripting language.”

What is PowerShell?

command-line shell
and
a scripting language

- You don't have to be a scripter to enjoy PowerShell

WMF

- Window Management Framework – is the core that provides PowerShell

The Windows Management Framework Core package provides updated management functionality for IT Professionals. This package includes the following components: Windows PowerShell 2.0 and Windows Remote Management (WinRM) 2.0.

History

- PowerShell runs on pretty much any Windows system to include Windows XP
 - Installed by default on everything Windows 7 and later
 - PowerShell v2 is the most common as it was installed on most systems (e.g. Win 7)
 - The latest current version is PowerShell v5
 - With each new release comes added functionality and add logging/security controls.
- XP just won't die

Extending PowerShell

The base PowerShell console can be extended to include modules (collection of cmdlets and code). Modules allow easier access to:

- Active Directory
- Exchange
- Does it vary?

- But wait there's more!

Today

- The latest current version is PowerShell v5
 - With each new release comes added functionality and add logging/security controls.
- Not just for Windows anymore....
WHAT?
 - Windows has made versions for Linux, MAC OSX, Docker

PowerShell > Security

- Possible uses: Administration of the system, troubleshooting
 - Office 365
 - Management of remote systems
 - Desired State configuration
 - Active Directory
 - Azure automation
- This list goes on and on and on and on...

• Not used for Security only... much more than that

Terminology

Some basic terms:

- **Parameter** – data that can be passed to a command. May be mandatory. “qualifier”
- **Variable** - Storage container for data (strings, int, etc.)
- **Function** – a grouping of similarly functioning code
- | “pipe” – the ability to pass one command/component to another.
- > “redirect” – redirects the output of a command to a file.
- **Alias** – duplicate name for same function

- We have to get this out of the way Sorry we don't like it either.

PowerShell Interface

PowerShell comes with two primary interfaces:

- Console – allows for the direct interaction with commands/scripts
- ISE- The Integrated Scripting Environment (ISE) provides more of a graphical interface and allows you to build scripts before running them
 - The ISE is not installed by default on Server versions.

Built in Variables

PowerShell comes with some “automatic” built-in variables.

- **\$Profile** –full path of PS profile
- **\$PSHome** – full path of PS installation directory
- **\$PSModule** path
- **\$PWD** – full path of current directory
- **\$_** - Contains the current object in pipeline object (we will cover this one a lot)

- If they are variable how are they built in?

Create your own variables

Creating your own variable is easy

- \$ (show me the money)
- **VariableName** - anything you want
- = assigning the cmdlet (anything) to the variable
- Remove-Variable – incase you didn't really want it.

More on variables later...

Labs

We will give you a few minutes to start the first 2 labs and then we will demo them in class.

- LAB 0 – Getting Started... had to start somewhere.
 - Launching PowerShell
- LAB 1 – Some commands.

• Finally I get to make use of this laptop I am hauling around

PowerShell Lies

Isn't convenient that PowerShell allows us to run common commands like `cd` and `dir`?

It is... but PowerShell lies.

Instead of calling the command itself PowerShell aliases the commands to PowerShell cmdlets.

- Lier, lier pants on fire!

Broke it

Because the commands are really aliases to cmdlets they can break.

Let's look at the dir command.

- You break it you bought it.

Did you say Cmdlets?

Predefined “commands” in PowerShell are called cmdlets

- Cmdlets follow a verb-noun naming convention
- Each command can be passed “qualifiers” and further parameters to narrow down what you are looking for
- cmdlets available on a machine vary based on role of system, and version of PowerShell installs
 - `$PSVersionTable.PSVersion`
- Little commands = cmdlets.

How are Cmdlets Written?

“Cmdlets are instances of the .NET framework classes; they are not stand alone executables”

- Two primary classes used:
 - Cmdlet base class
 - Pscmdlet base class – for more complex cmdlets
- Programmers can write their own.

The 1st Cmdlet

Get-Command

VERB



Noun



Windows PowerShell
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\> Get-command

CommandType	Name	Version
Alias	Add-ProvisionedAppxPackage	3.0
Alias	Add-VMToCluster	2.0.0.0
Alias	Add-WindowsFeature	2.0.0.0

-
-
-

Labs

Using Get-Command find some help.

- LAB 2 That help was helpful.

How do I get help?

Lab 2 Walkthrough:

- Get-help is a cmdlet.....
- Category types include:
 - Cmdlets
 - Aliases
 - Help_files
 - Providers

What is a help file?

- Help file explains a concept such as:
 - how to write a for loop, remoting, regex
- Great reference when you do not have internet access
- To view available help files:
 - `Get-help about*`
 - `Get-help -category helpfile`

Get-help Get-Help

```
PS C:\Users\Admin\Desktop> get-help get-help
```

NAME

```
Get-Help
```

SYNOPSIS

```
Displays information about Windows PowerShell commands and concepts.
```

SYNTAX

```
Get-Help [-Full] [[-Name] <string>] [-Category <string[]>] [-Component <string[]>] [-F  
] [<CommonParameters>]
```

```
Get-Help [-Detailed] [[-Name] <string>] [-Category <string[]>] [-Component <string[]>  
g[]>] [<CommonParameters>]
```

```
Get-Help [-Examples] [[-Name] <string>] [-Category <string[]>] [-Component <string[]>  
g[]>] [<CommonParameters>]
```

```
Get-Help [-Parameter <string>] [[-Name] <string>] [-Category <string[]>] [-Component <  
ole <string[]>] [<CommonParameters>]
```

- For when you need double help

Get-help get-help -examples

----- EXAMPLE 15 -----

```
C:\PS>get-help c:\ps-test\MyScript.ps1
```

Description

This command gets help for the MyScript.ps1 script. For information about writing help

- Can you give me an example?

Get-help Get-help -Detailed

PARAMETERS

-Category <string[]>

Displays help for items in the specified category. Valid values are Alias, Category.

Category is a property of the MamlCommandHelpInfo object that Get-Help returns.

-Component <string[]>

Displays a list of tools with the specified component value, such as "Exchange".

Component is a property of the MamlCommandHelpInfo object that Get-Help returns.

-Detailed [<SwitchParameter>]

Adds parameter descriptions and examples to the basic help display.

This parameter has no effect on displays of conceptual ("About_") help.

Get-help get-help -Full

-Parameter <string>

Displays only the detailed descriptions of the specified parameters. Wildcards are permitted.

This parameter has no effect on displays of conceptual ("About_") help.

Required?	false
Position?	named
Default value	
Accept pipeline input?	false
Accept wildcard characters?	false

• I ate too much I am full... see what I did there?

Parameters

“Qualifiers” / Arguments that can be past to the cmdlet to further define what/how it is run. Command line customization.

- **Named** – specifically defined parameters (e.g. online)
- **Positional** – first non-named parameter is assigned to first positional parameter. Args array.

Parameter breakout

-Parameter <string>

Displays only the detailed descriptions of the specified parameters. Wildcards are

This parameter has no effect on displays of conceptual ("About_") help.

Required?	false
Position?	named
Default value	
Accept pipeline input?	false
Accept wildcard characters?	false

- Breakout - a teenage worse nightmare.

Parameter breakout

-Name <string[]>

Specifies one or more processes by process name. You can type multiple process names. The parameter "Name" is optional.

Required? false

Position? 1

Default value

Accept pipeline input? true (ByPropertyName)

Accept wildcard characters? true

Parameter Input

`-Parameter <string>`

Displays only the detailed descriptions of the specified parameters. Wildcards are permitted.

This parameter has no effect on displays of conceptual ("About_") help.

Required?	false
Position?	named
Default value	
Accept pipeline input?	false
Accept wildcard characters?	false

Parameters cont.

Almost all cmdlets have/accept parameters.

- Optional (most) vs. Required
- Required parameters will be marked by `required=true`.
- Required Parameters throw errors when omitted.

Parameter breakout

-Parameter <string>

Displays only the detailed descriptions of the specified parameter

This parameter has no effect on displays of conceptual ("About_

Required?	false
Position?	named
Default value	
Accept pipeline input?	false
Accept wildcard characters?	false

- Splat ... the last noise a fly hears. Funnier when you read next slide.

Parameters Syntax.

-ParamName ParamValue

- Passing more than one
 - Space between each
 - Splat format;
 - *\$myargs = @{ path = \$env:windir }*
 - *Get-childitem \$myargs*

Let's Get Started

In order to run the tool we first need to download it.

- How to download our tool in PowerShell
- Use invoke-webrequest commandlet (damn verb/noun again)
- Download file by passing URL and saved file destination.

Downloading...

- Powershell 3+

*invoke-webrequest -uri "http://link -
outfile "c:\path\to\file"*

- Powershell 2

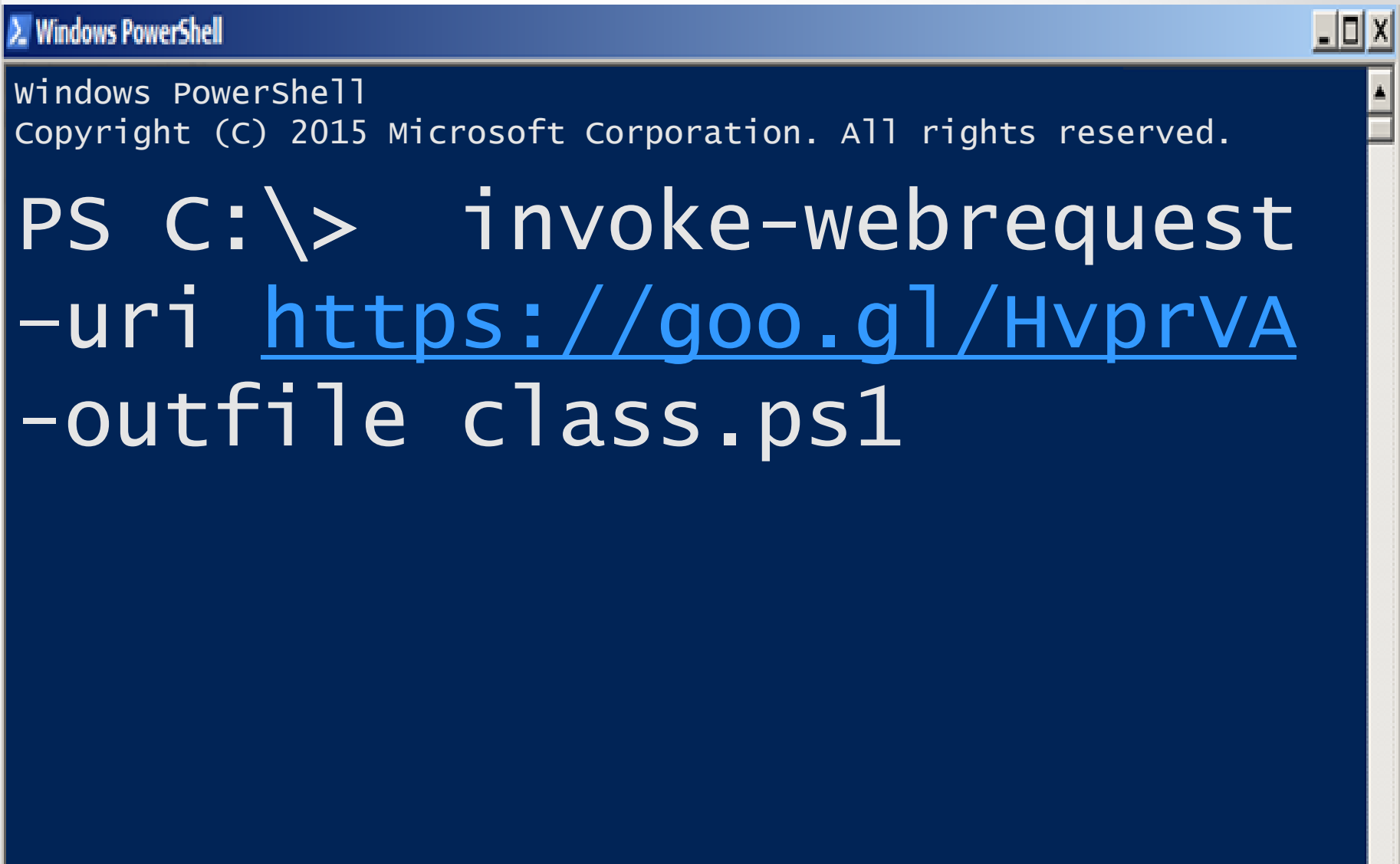
*(new-object
net.webclient).downloadfile("URL", "\$f
ullpathofDestination"*

Now forget what you saw
in the previous slide.



• You sexy beast!

Let's Get Started

A screenshot of a Windows PowerShell window. The title bar reads "Windows PowerShell". The window content shows the following text:

```
Windows PowerShell  
Copyright (C) 2015 Microsoft Corporation. All rights reserved.  
  
PS C:\> invoke-webrequest  
-uri https://goo.gl/HvprVA  
-outfile class.ps1
```

Script Params

Class Activity – looking at the script we just downloaded what parameters are required?

- Time to start looking at the script... wait did I already say that?

Stretch

<break>

- Little break ... now back to work.

Objects

- Some cmdlets contain multiple attributes/properties – these are defined by Objects
- *Sample – get-service.*

Windows PowerShell
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

```
PS C:\> get-service
```

Status	Name	DisplayName
-----	-----	-----
Running	AdobeARMservice	Adobe Acrobat
Stopped	AdobeFlashPlaye...	Adobe Flash Player
Stopped	AJRouter	AllJoyn Router Servic
Stopped	ALG	Application Layer
...		

More on Objects

When you type `get-service` the object returned is an array.

- LAB 3 Playing w/ Objects

Get-Member

The get-member cmdlet lists ALL available objects and their properties.

- Prints all available properties associated w/ cmdlets
- Useful in customizing queries

Pipes

Piping allows you to pass/chain commands together. Technically it allows to perform further actions.

- Linux example: `ls -al | more`
- PowerShell: `get-service | get-member`

```
Windows PowerShell
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\> get-service | get-member

TypeName: System.ServiceProcess.ServiceController

Name                MemberType          Definition
----                -
Name                AliasProperty      Name = ServiceName
RequiredServices    AliasProperty      RequiredServices =
ServicesDependedOn
Disposed            Event
System.EventHandler Disposed(System.Object,
System.EventArgs)
.
.
.
StartType           Property
System.ServiceProcess.ServiceStartMode StartType {get;}

```

Pipe Example Details

Get-Member returns the members, properties and methods for objects. It can get it 2 ways:

- Parameter – InputObject
- Pipe – via Pipe command

Pipe Example Details

The get-service cmdlet pulls information(objects) for all services and passes it to get-member.

Filtering

Allows us to manipulate the output/data returned to our specific needs.

- Good to run get-member first so we know what to look for

Common Filters

- Sort-object – returns a list sorted on your input
- Select-object – allows us to select cmdlet objects we are interested in
- Where-object – allows for pattern matching
- ForEach-object – performs action on each object
- Common – a good rapper

Windows PowerShell

Windows PowerShell

Copyright (C) 2015 Microsoft Corporation. All rights reserved.

```
PS C:\> get-service | select-  
object Name, Status, Starttype
```

Name	Status	StartType
-----	-----	-----
AdobeARMservice	Running	Automatic
AdobeFlashPlayerUpdateSvc	Stopped	Manual
AJRouter	Stopped	Manual
ALG	Stopped	Manual
AppHostSvc	Running	Automatic
AppIDSvc	Stopped	Manual
Appinfo	Running	Manual
Apple Mobile Device Service	Running	Automatic
AppMgmt	Stopped	Manual
AppReadiness	Stopped	Manual

- Just a demo.. Ha you thought I was going to say example 3 didn't you?

Select-Object Details

Allows us to select (duh) which objects we are interested in. We can pull from pretty much anything that comes back from get-member (properties)

- Just the facts (details).

Windows PowerShell
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

```
PS C:\> get-service | select-object Name,  
Status, Starttype |where-object {$_.status -  
eq "running"}
```

Name	Status	StartType
-----	-----	-----
AdobeARMservice	Running	Automatic
AppHostSvc	Running	Automatic
Appinfo	Running	Manual
Apple Mobile Device Service	Running	Automatic
atahost	Running	Automatic
AudioEndpointBuilder	Running	Automatic
Audiosrv	Running	Automatic
BBUpdate	Running	Manual
BFE	Running	Automatic

Where-object

Where-object {\$_ .status -eq
“running”}

- { - indicates that the a code snippet applied to an object
- \$_ - default variable indicating the current instance of the object being evaluated
- **-eq** – conditional operator in this case equals
- **“running”** – condition check.

Labs

Filters & Pipes

- LAB 4 Try on your own.

Coding practices

- If
 - If this then that (IFTT)
- Function
 - Separate code piece that can be called
- While
 - While I sleep nothing gets done
- For
 - For letter in word type letter word.
- Foreach
 - Foreach child = school debt

Background of the Tool

Scenario:

- Have access to a system, and want to obtain a user's credentials
- When a user saves a file to particular network share we want to prompt the user with a password prompt
- Credentials obtained are sent to a Metasploit web listener.
- Idea came from a Metasploit minute video by Rob Fuller (Mubix)

Demo of tool

- <break>

- Please work ...pretty please.

Code review

- Demos demos demos

- Did you just repeat repeat the word demo?

Where else can we look?

- Providers shows various kinds of data storage as a disk drive.
- Powershell supports the ability to browse the registry, certificate store, and
- Get-psprovider will list available what PS has access to

Sample

```
Windows PowerShell
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\> get-psprovider |select
Name, capabilities, Drives

Name                Capabilities                Drives
----                -
Registry            ShouldProcess, Transactions {HKLM, HKCU}
Alias                ShouldProcess                {Alias}
Environment          ShouldProcess                {Env}
FileSystem           Filter, ShouldProcess, Creds. {C, F, D}
Function             ShouldProcess                {Function}
Variable            ShouldProcess                {Variable}
```

- Psprovider – not what you see on the end of a letter.

Sample

```
PS C:\> Get-PSPProvider | select Name, Capabilities, Drives |
```

Name	Capabilities	Drives
WSMan	Credentials	{WSMan}
Alias	ShouldProcess	{Alias}
Environment	ShouldProcess	{Env}
FileSystem	Filter, ShouldProcess	{C, D, Z}
Function	ShouldProcess	{Function}
Registry	ShouldProcess, Transactions	{HKLM, HKCU}
Variable	ShouldProcess	{Variable}
Certificate	ShouldProcess	{cert}

- Look ma, we have drives

Sample

```
PS C:\> Get-PSPProvider | select Name, Capabilities, Drives |
```

Name	Capabilities	Drives
WSMan	Credentials	{WSMan}
Alias	ShouldProcess	{Alias}
Environment	ShouldProcess	{Env}
FileSystem	Filter, ShouldProcess	{C, D, Z}
Function	ShouldProcess	{Function}
Registry	ShouldProcess, Transactions	{HKLM, HKCU}
Variable	ShouldProcess	{Variable}
Certificate	ShouldProcess	{cert}

- Look ma, we have registry

Navigating providers

- Cmdlets with the noun item, or items are usually used for this purpose.
- The noun items is used to retrieve a folder
 - Eg, get-childitem
- The noun itemproperty returns data on a single object/file.

Where to monitor?

GOAL: When a user saves a file to particular network share we want to prompt the user with a password prompt.

DECISION: Where/what do we monitor?

- We can ask the user (person running script not the victim)
- We can look at the registry
- We can “hard code” it
- We can look at drive letters..... And on and on and on...

• No this isn't a test.... Or is it?

Navigating the registry

- The HKEY_Local_Machine (HKLM) & HKEY_Current_User (HKCU) registry keys can be browsed directly through PowerShell.
 - To navigate into these keys use the set-location or its alias ;)
 - Use get-item to query registry keys
 - Use get-itemproperty to read the contents of a value.
- PS allows for easy manipulation of registry – no that's not a joke.

Registry example

- Set-location "HKCU:\software\microsoft\Internet Explorer"
- Get-item -path "HKCU:\software\microsoft\Internet Explorer"
- Get-itemproperty -path "HKCU:\software\microsoft\Internet Explorer" -name "Download Directory"

Lets review the registry code from Invoke-Gimme

GOAL: Grab a file path form the most recently used list (e.g. recent files)

- Two ways of doing similar items:
 - Two individual if statements (line 1 & 2)
 - One compound statement (line 7)

Registry Demo cont.

Let's look at the code.



- Finally the cool stuff

Lab 5

Time to mess around with the registry... you may need admin.
BE CAREFUL.

- LAB 5 Registry

IF Statement Closer Look

If statements are used to determine an action based on the result of statement.

- If statement is \$TRUE everything in { ... } gets executed
- Switch statement could be used in lieu of several if's



• Red light Green light...

If statement continuation

- Let's look at the code.



- If class is good don't fall asleep.

Break

Please make sure you have download and save WMIExplorer onto your systems before we begin the next section.



WMI

WMI – Windows Management Instrumentation

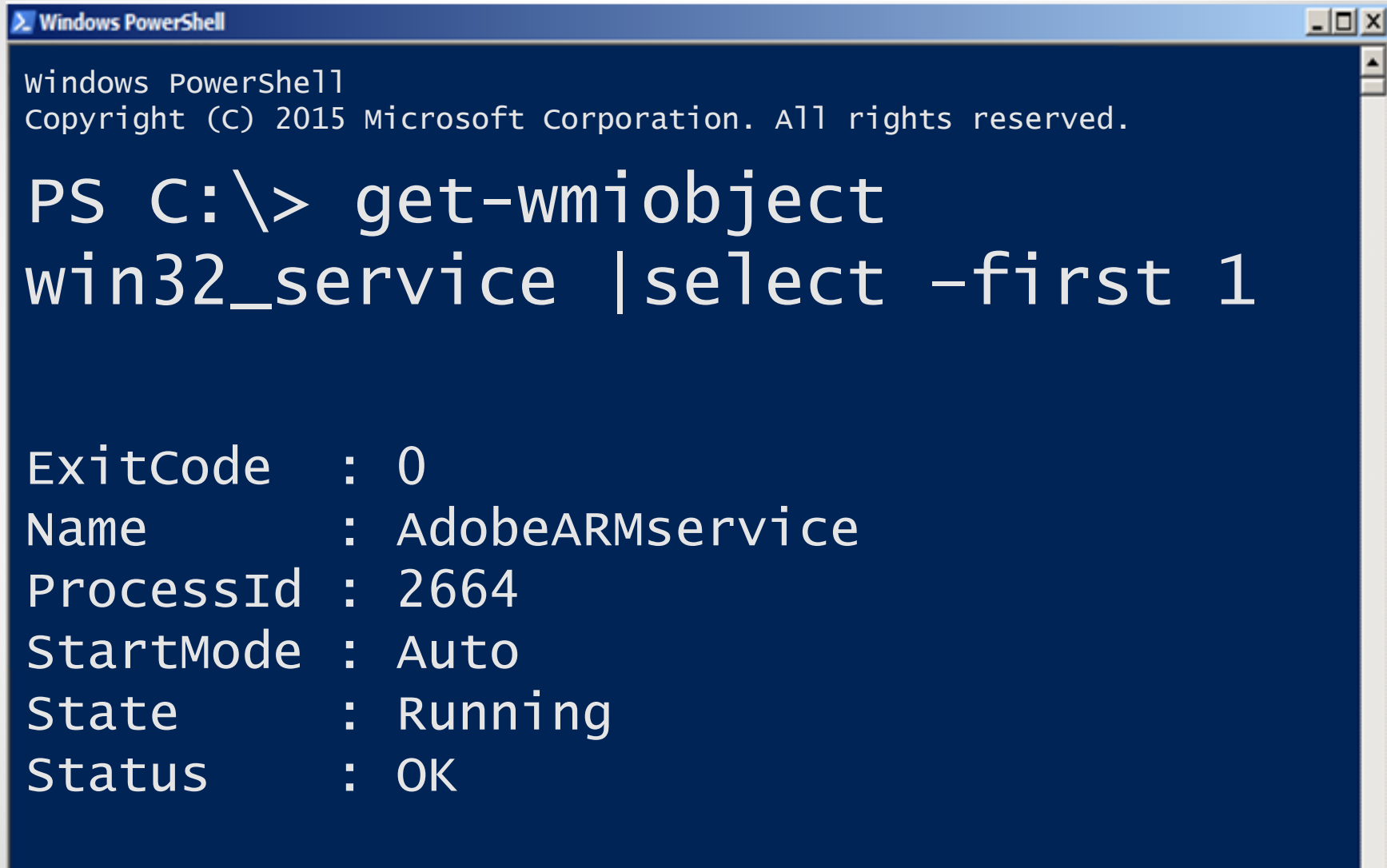
- We can look at WMI as a collection of objects that provide access to different parts of the operating system
- WMI can be used both locally or remotely to obtain information or start process.

<http://www.darkoperator.com/blog/2013/1/31/introduction-to-wmi-basics-with-powershell-part-1-what-it-is.html>

More WMI

- WMI is organized into namespaces. Think of a namespace as sort of like a folder that ties to a specific product or technology.
- WMI is divided into a series of classes. A class represents a management component that WMI knows how to query.

WMI Sample



```
Windows PowerShell
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\> get-wmiobject
win32_service |select -first 1

ExitCode      : 0
Name          : AdobeARMservice
ProcessId     : 2664
StartMode     : Auto
State         : Running
Status       : OK
```

- Get-Service – the “hello world” of PowerShell

Brief explanation of CIM

- First, you have to remember that CIM = WMI = CIM. CIM is an open standard from the [Distributed Management Task Force](#) (DMTF), with the latest version introduced in January 2016. CIM provides a common definition of management information for systems, networks, applications, and services, and it allows for vendor extensions. WMI is the Microsoft implementation of CIM for the Windows platform.

<https://blogs.technet.microsoft.com/heyscriptingguy/2016/02/08/should-i-use-cim-or-wmi-with-windows->

- powershell/

Some notes about CIM

- The new “CIM cmdlets,” such as `Get-CimInstance` and `Invoke-CimMethod`—These are more or less equivalent to the old “WMI cmdlets,” although they communicate over WS-MAN (implemented by the Windows Remote Management service) instead of using RPCs. This is Microsoft’s way forward, and running `Get-Command -noun CIM*` reveals that Microsoft offers a lot more functionality through these commands.

- Not sure where I plagiarized this from but its definitely not my words.

Three Queries same result

Goal is to locate and identify the win32_service class

1. WMIExplorer- Great for helping constructing a query for use with Powershell.
2. `Get-wmiobject -namespace root\CIMV2 -class win32_service`
3. `$query_service = "select * from win32_service"`
 - o `Get-wmiobject -query $query_service`

So why is example three over complicated?

- Example 3
 - (`$query_service = "select * from win32_service"`)
- Example three uses a language referred to as WQL which is SQL for WMI
- In order to configure the event monitoring with WMI, you must know how to use WQL to setup the

•*hint we are going to use event subscribers soon

WMI Explorer

Screenshot below path:

1. Connect to localhost
2. Root\CIMv2
3. Win32_service

The screenshot displays the WMI Explorer interface. On the left, the 'Namespaces' tree shows the path: \\STU-WINDOWS7\ROOT > ROOT\CIMV2 > Win32_service. The central pane shows a table of classes with 'Win32_Service' selected. The right pane shows the 'Instances' list with various service names.

Name	Laz...	Description	Path
Win32_Service	Fal...	The Win32_Servi...	\\STU-WINDOW...
Win32_ServiceCo...	Fal...	Instances of this ...	\\STU-WINDOW...
Win32_ServiceSp...	Fal...	Instances of this ...	\\STU-WINDOW...
Win32_ServiceSp...	Fal...		\\STU-WINDOW...

Instances (155) | Properties (25) | Methods (12) | Query | Script | Logging

Instance Options

Quick Filter: Show Null Values Show System Properties

Instances

- Win32_Service.Name="AeLookupSvc"
- Win32_Service.Name="ALG"
- Win32_Service.Name="AppIDSvc"
- Win32_Service.Name="Appinfo"
- Win32_Service.Name="AppMgmt"
- Win32_Service.Name="aspnet_state"
- Win32_Service.Name="AudioEndpointBuilder"
- Win32_Service.Name="Audiosrv"
- Win32_Service.Name="AVGIDSAgent"
- Win32_Service.Name="avgwd"
- Win32_Service.Name="AxInstSV"

LAB

- Lab <Insert lab number here>
- 6 if you couldn't count.
- Antivirus query or local group administration

String manipulation

What if we have to manipulate input/output, take only a portion, escape characters, et. Al.? Strings to the rescue.

- Upper/Lower
- Split/Join
- Replace/Remove

• No this isn't string theory.

Create a variable

- `$a = 'The quick brown fox'`
- `[string]$a='The quick brown fox'`
- The options available to us can be seen by running `get-member` against a string.
- `$a | get-member -membertype method`

Manipulating text

- `$a.split()` & `$a.split('q')`
- `$a.ToUpper()`
 - THE QUICK BROWN FOX
- `$a.replace('Fox','Lion')`
 - the quick brown fox
- What is `$a` equal to after running the above commands?
- Is it a Fox or Lion?

String Lab

- Lab 7 Similar to our sample

WMI Subscriber

- Explain concept behind event driven wmi
- What are the three components:
 - Query
 - Action
 - Name

Writing a query

- Query is written using WQL
- Backslashes must be commented out. So if we want to capture at path C:\windows, we must use C:\\windows
- The query is comprised of the class to query and then any other components.

Action and ID

- Action is what we want to occur after the query is met.
 - Command(s) are surrounded by hashes (eg, { })
- The ID is the name that will uniquely identify this subscriber.

WMI Subscribers

- Temporary
 - Runs as long as the PowerShell window is open.
- Permanent ---- W00T!!!!!!
 - Run as system
 - Great way to maintain persistence as it survives a reboot
 - or use it as a HID

• Subscribe now for only three easy payments of \$19.95

10/21/2016 • 97

WMI events

- Classes of interest:
 - __instanceCreationEvent
 - __instanceDeletionEvent
 - __InstanceModificationEvent
 - registryKeyChangeEvent
 - RegistryTreeChangeEvent
 - RegistryValueChangeEvent
- Permanent ---- WOOT!!!!!!
 - Run as system
 - Great way to maintain persistence or use it as a HID

Temporary event subscriber

- `$Query = "Select * from RegistryTreeChangeEvent where Hive='HKEY_LOCAL_MACHINE' AND RootPath='Software\\'"`
- `$Action = { $Global:Data = $Event ; Write-Host 'Something happened!' -ForegroundColor Green -BackgroundColor Black }`
- `Register-WmiEvent -SourceIdentifier "test" -Query $query -Action $action`

DEMO

- Lets analyze our code!
 1. Open wmi explorer and
 2. cim_datafile class. It is located under root/cimv2
 3. What Properties can we filter on.
 4. Lets do work!

Our code

```
$query =("select * from  
_instanceCreationEvent within 10  
where targetinstance isa  
'cim_datafile' and  
targetInstance.drive=' $pathToMonitorD  
riveLetter'")
```

*Consolidated filter

- Looks like SQL did I fall asleep and wake up in a DB lecture?

WMI subscriber code

```
Register-WmiEvent -Query $query -  
SourceIdentifier "MonitorFiles3"  
-Debug -Action {  
$Global:MyEVT=$event  
$modFilename=$event.SourceEventArgs.  
newevent.TargetInstance.fileName  
$modExtension=$event.SourceEventA  
gs.newevent.TargetInstance.extens  
ion...
```

- Debug is debomb

Lab

• 8

- Just a plain lab like this plain slide

Combining .NET and Powershell

- .net framework consist of code that other developers can utilize to obtain repeatable and consistent output.
- .Net classes are used when we must provide additional functionality to powershell.

Calling .net libraries

- Words, hand gestures and stuff –two demos in one

- Demo :

```
[Microsoft.VisualBasic.Interaction]
```

```
::MsgBox("Unable to reach  
$modDriveLetter Drive. File  
$modFileNameExtension was not  
saved",
```

```
'okonly, SystemModal, Information',  
"Error saving file")
```

calling VB.interaction library

- Who you going to call?

Example 1

- Goal is to download a file using the `system.net.webclient` class
 1. We must first determine if this class is appropriate for our task.
 1. Can check MSDN:
[https://msdn.microsoft.com/en-us/library/system.net.webclient\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.net.webclient(v=vs.110).aspx)
 2. Assign a variable within Powershell and then run `get-member $classdemo=(new-object system.net.webclient)`

DownloadFile	Downloads data from a resource to a local file.
DownloadFileAsync	Downloads data from a resource to a local file, without blocking the calling thread.
DownloadString	Downloads a String from a resource and returns a String .

Example 2

2. We are going to use the
Downloadfile method.

2. `$classdemo.downloadfile(<URL>,<full
path of destination>`

Examples in our script

- `[Microsoft.VisualBasic.Interaction]::MsgBox()`
 - Purpose: Present the first error box to the user
- `[Environment]::UserDomainName`
 - Purpose: Populate Domain name into credentials box
- `[Environment]::UserName`
 - Purpose: Populate username into credentials box

FIN

- About Damn Time already....

Where to go from here?

We built this script to highlight some PowerShell basics and how they could be used pretty simply to attack a system. It's up to you to improve on this and make it better... don't worry I will take credit 😊

- Demo tool is “red” team focused – how can you use what you learn for “blue” team? Monitoring a directory may be a good thing?
- Make this into a module that can run on it's own or as a module for other PS tools
- Improve where it looks to monitor
 - Why choose one when you can have them all
- Validate the creds you get
 - Multiple pop-ups, back-end validation
 - Got real creds? Good then get the data
- Build your own that has nothing to do with this – hopefully we inspired you and given you the basics to get started.

• The world is your playground – go play

Further PowerShell Goodness

- <https://www.youtube.com/watch?v=3vJvkANKVWA>
- <https://www.youtube.com/watch?v=893NGtZlod8> - Gray Hat Powershell
- https://www.youtube.com/watch?v=4X_uBL2YpmA - Practical Powershell programming for Practical Programmers
- https://dl.mandiant.com/EE/library/MIRcon2014/MIRcon_2014_IR_Track_Investigating_Powershell_Attacks.pdf
- <https://www.youtube.com/watch?v=cXWtu-qalSs>
- <https://adsecurity.org/?p=2843>
- <https://www.youtube.com/watch?v=j9HMjal9qgk>