PowerShell for Audit, Compliance and Security

ISACA North Texas

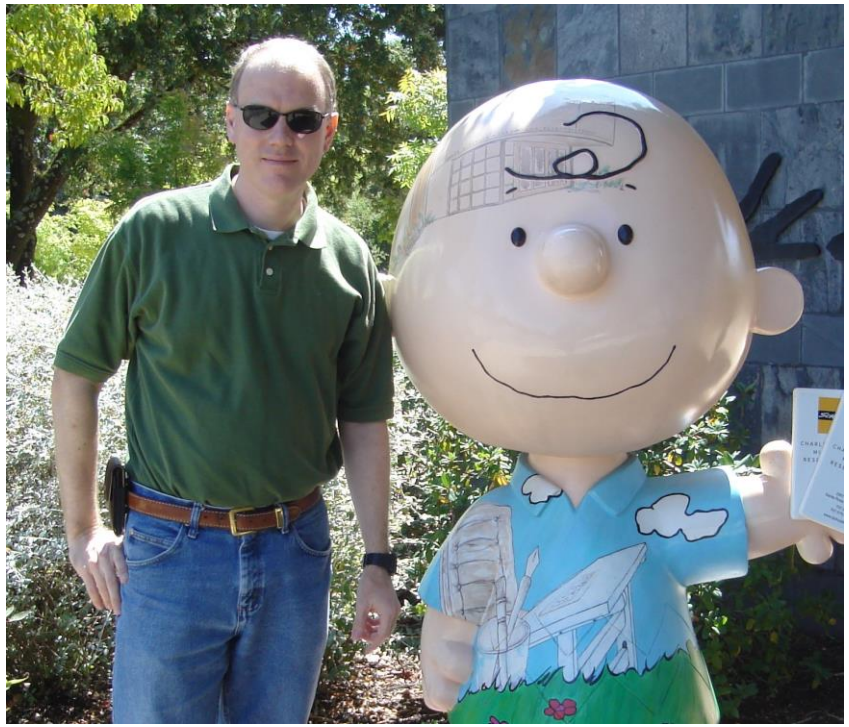
Friday, November 13, 2020

Agenda

- Introductions
- Why automate?
- Why PowerShell?
- Foundational PowerShell commands
- Getting help
- Extracting data
- Handling data
- Exporting data

Introductions

- Clay Risenhoover
  - CPA/CITP, CISA, CISM, CISSP, etc.
  - SANS Principal Instructor
  - Author of SANS AUD507 course



- Today's material is a preview of upcoming SANS SEC557 course

Continuous Automation for Enterprise and Cloud Compliance

What This Session Is(n't)

ISN'T

- Non-technical
- Slow-paced
- Theoretical
- Tutorial
- "Click through and read the slides to you"

IS

- Technical
- Fast paced
- Practical
- Demonstration
  - 150+ PowerShell commands during demos

Downloads

- Get all of today's demos and PDFs of the slide decks from:

https://github.com/AuditClay/AuditScripts

GitHub

Why Automate? (1)

Then

Now

```yaml
AWSTemplateFormatVersion: "2010-09-09"
Description: AMI web server template
Resources:
  MyEC2Instance:
    Type: "AWS::EC2::Instance"
    Properties:
      ImageId: "ami-0ff8a91507f77f867"
      InstanceType: t2.micro
      KeyName: TestKey
      BlockDeviceMappings:
        -
          DeviceName: /dev/sdm
          Ebs:
            VolumeType: io1
            Iops: 200
            DeleteOnTermination: false
            VolumeSize: 20
```

Why Automate? (2)

Then

- Discrete servers
- Long development cycles
- Infrequent changes
- Physical network devices

Now

- Cloud
- Virtualization
- Agile
- DevOps
- Infrastructure as code

"Red Queen Syndrome"

‘Well, in our country,’ said Alice, still panting a little, ‘you’d generally get to somewhere else—if you ran very fast for a long time, as we’ve been doing.’

‘A slow sort of country!’ said the Queen. ‘Now, here, you see, **it takes all the running you can do, to keep in the same place. If you want to get somewhere else, you must run at least twice as fast as that!’**

Why PowerShell?

- Auditors/infosec/compliance professionals need to "live off the land"
- Your admins are probably already using PowerShell for daily administration
- Native to Windows systems and easy to add to others
- Large community developing scripts and modules

PowerShell Overview

- Cross-platform scripting language
- Fully object-oriented
- Windows-native and cross-platform versions
- .Net (Core) integration

PowerShell Editions

- Windows PowerShell
  - Powershell.exe
  - Windows native
  - Full .NET framework exposed
  - Stuck in version 5.1
  - No new features - security/stability updates only
- PowerShell Core
  - Pwsh.exe on Windows
  - Cross-platform
  - .NET Core framework
  - Version 7 LTS
  - New features added regularly

Cmdlets

- Cmdlets: Pre-compiled fully-functional commands

```
PS C:\> Get-Command -Type Cmdlet

CommandType     Name                                       Version    Source
-----------     ----                                       -------    ------
Cmdlet          Add-AppvClientConnectionGroup              1.0.0.0    AppvClient
Cmdlet          Add-AppvClientPackage                      1.0.0.0    AppvClient
Cmdlet          Add-AppvPublishingServer                   1.0.0.0    AppvClient
Cmdlet          Add-AppxPackage                            2.0.0.0    Appx
Cmdlet          Add-AppxProvisionedPackage                 3.0        Dism
Cmdlet          Add-AppxVolume                             2.0.0.0    Appx
Cmdlet          Add-BitsFile                               2.0.0.0    BitsTransfer
Cmdlet          Add-CertificateEnrollmentPolicyServer      1.0.0.0    PKI
```

Functions

- Usually written in PowerShell and not pre-compiled
- Work like cmdlets for all practical purposes

```
PS C:\> Get-Command -Type Function

CommandType     Name                                       Version    Source
-----------     ----                                       -------    ------
Function        A:
Function        Add-BCDataCacheExtension                   1.0.0.0    BranchCache
Function        Add-BitLockerKeyProtector                  1.0.0.0    BitLocker
Function        Add-DnsClientNrptRule                      1.0.0.0    DnsClient
Function        Add-DtcClusterTMMapping                    1.0.0.0    MsDtc
Function        Add-EtwTraceProvider                       1.0.0.0    EventTracingManagement
Function        Add-InitiatorIdToMaskingSet                2.0.0.0    Storage
Function        Add-MpPreference                           1.0        Defender
```

Aliases

- Short names for other cmdlets or functions
- Recommendation: Use only in interactive console; use full names in scripts for readability

```
PS C:\> Get-Alias

CommandType     Name                                       Definition
-----------     ----                                       ----------
Alias           %                                          ForEach-Object
Alias           ?                                          Where-Object
Alias           ac                                         Add-Content
Alias           asnp                                       Add-PSSnapIn
Alias           cat                                        Get-Content
Alias           cd                                         Set-Location
Alias           chdir                                      Set-Location
```

Command Name Format

- Cmdlets and functions are commonly named in a “Verb-Noun” format
- Noun is always singular
- Examples:
  - **`Get-ADUser`**
  - **`Set-FileShare`**
  - **`Write-Host`**
  - **`Remove-Job`**
  - **`New-Object`**

```
PS C:\> Get-Verb

Verb        Group
----        -----
Add         Common
Clear       Common
Close       Common
Copy        Common
Enter       Common
Exit        Common
...
PS C:\> Get-Verb | Measure-Object
Count    : 98
```

Special Mention – Quotation Marks

- Convention is to use single quotes (')
- Double-quotes (") are used for:
  - Including a variable's contents in a string
  - **`$cmd="Get-ChildItem $path"`**
- Including single quotes inside a string
  - **`$name="O'Brian"`**
- Using escaped characters
  - **`Write-Host "Col1`tCol2"`**
  - ` is the "grave accent," and is used as the escape character (similar to "\t" in other languages)
- Smart quotes ("") no longer break PowerShell

Case Sensitivity

- PowerShell is not case-sensitive
- **`Get-aduser`** is equivalent to **`gEt-ADuSeR`**
- Good idea to use PascalCase for readability

```
PS C:\> GeT-aLIas pWd
CommandType     Name
-----------     ----
Alias           pwd -> Get-Location

PS C:\> Get-Location
Path
----
C:\
```

Getting Help

- Use the **`Get-Help`** command
- Shows the built-in help text for the command
- Most-current help file will be online:
  - **`Get-Help Get-ADUser –Online`**

```
PS C:\> Get-Help Get-ADUser

NAME
    Get-ADUser

SYNTAX
    Get-ADUser -Filter <string> [-AuthType {Negotiate | Basic}] [-Credential <ps
    [-SearchBase <string>] [-SearchScope {Base | OneLevel | Subtree}] [-Server <
```

Updating Help Files

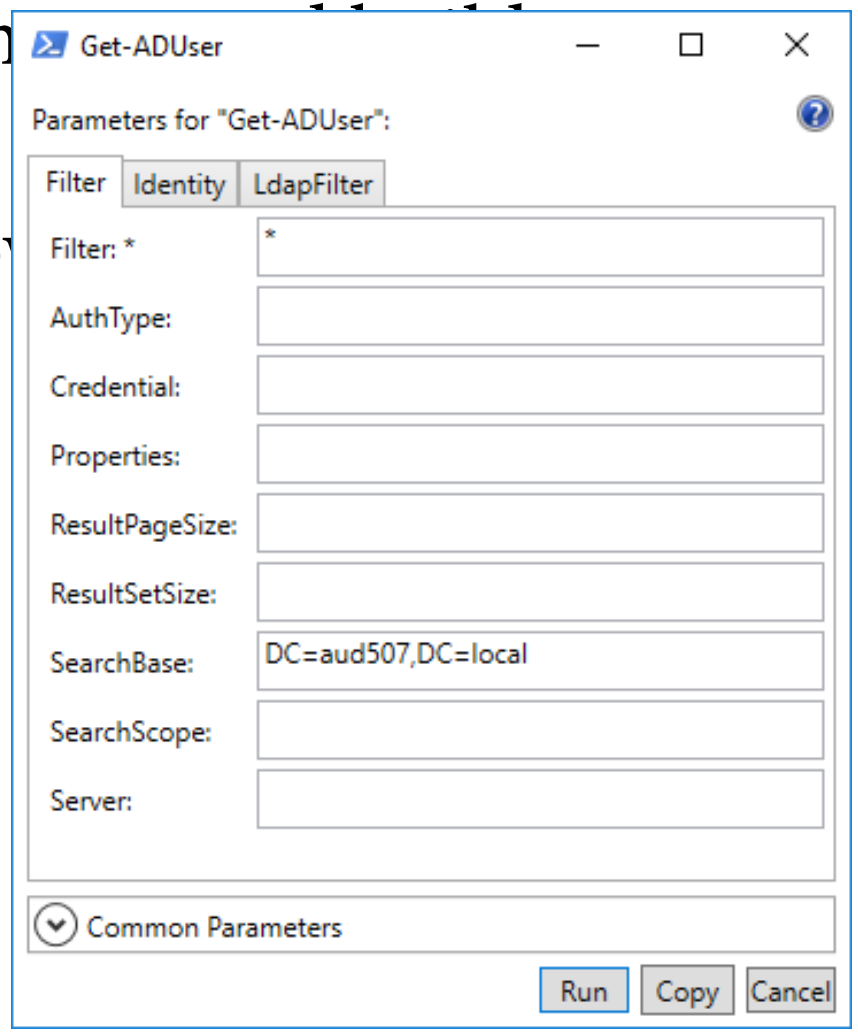
- **`Update-Help`** cmdlet downloads current copies of all help files
  - May take a while to run
- **`Save-Help`** cmdlet saves help files for offline updating
  - Good for non-internet connected hosts

```
PS C:\> Update-Help

 Updating Help for module AppBackgroundTask
    Connecting to Help Content...
    [oooooooooooooooooooooooooooo
```

GUI Command Editor

- **`Show-Command`** allows you to see param
  command visually
- The “Run” button pastes command into Po



Formatting Output (1)

- **`Format-Table`** and **`Format-List`** commonly used
- Others available:

```
PS C:\> Get-Command Format-* -Module Microsoft.PowerShell.Utility

CommandType     Name              Version    Source
-----------     ----              -------    ------
Function        Format-Hex        3.1.0.0    Microsoft.PowerShell.Utility
Cmdlet          Format-Custom     3.1.0.0    Microsoft.PowerShell.Utility
Cmdlet          Format-List       3.1.0.0    Microsoft.PowerShell.Utility
Cmdlet          Format-Table      3.1.0.0    Microsoft.PowerShell.Utility
Cmdlet          Format-Wide       3.1.0.0    Microsoft.PowerShell.Utility
```

Formatting Output (2)

- **`ConvertTo-*`** functions change objects to specified output formats

```
PS C:\> Get-Command ConvertTo*  -Module Microsoft.PowerShell.Utility

CommandType    Name               Version    Source
-----------    ----               -------    ------
Cmdlet         ConvertTo-Csv      3.1.0.0    Microsoft.PowerShell.Utility
Cmdlet         ConvertTo-Html     3.1.0.0    Microsoft.PowerShell.Utility
Cmdlet         ConvertTo-Json     3.1.0.0    Microsoft.PowerShell.Utility
Cmdlet         ConvertTo-Xml      3.1.0.0    Microsoft.PowerShell.Utility
```

Redirecting Output

- Redirect with **`Out-File`** (equivalent of “>” redirection operator)
- Tee output with **`Tee-Object`**

```
Get-ADUser -Filter * | Out-File -FilePath c:\users.txt –Append

Get-ADUser -Filter * | Tee-Object -FilePath c:\users.txt -Append |
       Sort-Object -Property SamAccountName
```

Select-Object

- Allows selection of only specified properties for an object
- Similar to SELECT clause in SQL
- Aliased to “select”

```
Get-WmiObject win32_NetworkAdapterConfiguration |
Select-Object Description, MACAddress, DHCPEnabled, IPAddress
```

Where-Object

- Filter results based on the results of a comparison
- Similar to SQL WHERE clause

  - Comparison statement format:

```
Get-Service | Where-Object Status -eq 'stopped'
```

  - Script-block format:
    - **$_** represents the current object being evaluated

```
Get-Service | Where-Object { $_.Status -eq 'stopped' }
```

Comparison Operators: Equality

-eq equals

-ne not equals

-gt greater than

-ge greater than or equal

-lt less than

-le less than or equal

```
PS C:\> 1 -lt 3
True
PS C:\> 3 -le 2
False
```

Comparison Operators : Contains/NotContains

-contains	Returns true when reference value contained in a collection
-notcontains	Returns true when reference value not contained in a collection

```
PS C:\> 1,2,3,4 -contains 3
True
PS C:\> 1,2,3,4 -notcontains 5
True
```

Comparison Operators : In/NotIn

-in	Returns true when test value contained in a collection
-notin	Returns true when test value not contained in a collection

```
PS C:\> 3 -in 1,2,3,4
True
PS C:\> 5 -notin 1,2,3,4
True
```

Comparison Operators : Like/Notlike

-like	Returns true when test string exists in another string
-notlike	Returns true when test string does not exist in another string

- Both allow the use of the ‘*’ wildcard

```
PS C:\> 'AUD507' -like '*50*'
True
PS C:\> 'AUD507' -notlike '*40*'
True
```

Common Data Formats

- JSON - JavaScript object notation
- XML - extensible markup language
- HTML - hypertext markup language
- CSV - comma-separated values
- Excel spreadsheets

JSON in PowerShell

- Convert between PowerShell objects and JSON with
  - ConvertFrom-Json
  - ConvertTo-Json
- Test JSON validity with Test-Json
  - PS Core only
  - Single-line JSON only

XML in PowerShell

- ConvertTo-Xml
- Select-Xml - uses Xpath queries
- Work with Common Language Infrastructure (CLI) XML using
  - Export-CliXml
  - Import-CliXml
- Export-CliXml can be used to saved encrypted credentials on Windows using the Data Protection API

CSV files in PowerShell

- CSV is a common output format for LOTS of tools
- Many security APIs give results as CSV
- PowerShell has native cmdlets for CSV handling:
  - ConvertFrom-Csv
  - ConvertTo-Csv
  - Import-Csv (handles CSV files with line feeds in cells)
  - Export-Csv

Excel Documents in PowerShell

- Source files are often Excel spreadsheets
- PowerShell can interact with the Excel application (sort of) through Microsoft's Office Interop assemblies
  - A bit heavy-handed for our purposes
- Import-Excel third-party module is ideal for this
  - Provides Import-Excel and Export-Excel functions
  - Doesn't require Excel to be installed on the system

Iteration

- The For or the Foreach loop:
  - Enables you to run through a list of things
  - Typically a list of numbers
  - Could also be the content of a file
- For:
  - `for(initialize; test; repeat) { }`
- Foreach:
  - `foreach($item in list object) { }`

For Statement

- Iterate through values:
  - `for ($x=1;$x -lt 255; $x++){}`
  - Start at 1
  - Count up to 255
  - Increment $x by one each time

```
for($x=1;$x -lt 255;$x++) {
  Invoke-Expression "ping -n 1 10.50.7.$x"
}
```

Foreach Loop

- `ForEach($item in list){}`

```
ForEach($host in (Get-Content hosts.txt)) {
  Get-Service -ComputerName $host |
    Out-File -FilePath Services.txt -Append
}
```

Shameless Plugs!

- Interested in the 3-day version of this material? I'll have a beta run of the SANS class early in 2021.
- Sign up for more info at:

https://www.sans.org/new-sans-courses

- Check the box for SEC557

Shameless Plugs!

- Can't get enough of hearing Clay talking through your computer speakers?
- I'm teaching a 40-hour CISSP review course for the chapter next month.
- Sign up at:

https://engage.isaca.org/northtexaschapter