```
 ____    ____________________________________________________________________________________________________
//¯¯\\__//¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\\___
\\__//¯¯¯ New-VmController [~]                                                                                 ___//¯¯\\
 ¯¯¯\\____________________________________________________________________________________________________//¯¯\\__//
     ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯    ¯¯¯¯
```

Introduction

This particular function is specifically meant to manage the production of virtual servers for Active Directory.
I just wrote up a document yesterday that features the function below named “Initialize-FeAdInstance”.
https://github.com/mcc85s/FightingEntropy/blob/main/Docs/2023_0124-(Initialize-AdFeInstance).pdf

All (3) of these functions are being worked on...

Get-FEDCPromo - For promoting a server to an Active Directory Domain Controller via (4) modes
https://github.com/mcc85s/FightingEntropy/blob/main/Version/2022.12.0/Functions/Get-FEDCPromo.ps1
New-VmController - For building a lab environment of preconfigured virtual machines (Not in the module yet)
https://github.com/mcc85s/FightingEntropy/blob/main/Scripts/New-VmController.ps1
Initialize-FeAdInstance - For populating an Active Directory Domain Controller with OU's, Groups, and Users
https://github.com/mcc85s/FightingEntropy/blob/main/Version/2022.12.0/Functions/Initialize-FeAdInstance.ps1

In this particular document, I'm only going to cover the function “New-VmController”, which is not part of the module quite yet. As stated in the above linked PDF file, this function was showcased in the following video:

01/12/23	2023_0112-(PowerShell | Virtualization Lab + FEDCPromo)	https://youtu.be/9v7uJHF-cGQ

In that particular video, I was working with the GUI for “Get-FEDCPromo”.

```
# Last edited : 2023-01-24 19:19:01
# Purpose     : Automatically installs a Windows Server 2016 instance for configuration

# [Objective]: Get (3) virtual servers to work together as an Active Directory domain controller
# cluster by using [FightingEntropy(π)] Get-FEDCPromo.

# Be really clever about it, too.
# Use a mixture of virtualization, graphic design, networking, and programming...
# ...to show everybody and their mother...
# ...that you're an expert.

# Even the guys at Microsoft will think this shit is WICKED cool...
# https://github.com/mcc85s/FightingEntropy/blob/main/Docs/2023_0103-(Get-FEDCPromo).pdf

# Gonna have to update this file up above ^
# Because I've replaced so many aspects of it within the last week.
```

In order to test the additions to Get-FEDCPromo, an available Active Directory domain must exist.

Before, the utility divided multiple aspects of the (scanning/login) process, and there are still a couple of additional things left to implement before that process is ready to (use/test).

At this juncture, I'm going to cover the function “New-VmController”.
First, I will paste the function wrapper below without the embedded classes.
Then, I will cover each individual class.

The way that I wrote my code is so that essentially every individual line follows the sacred rules of guys like Kevlin Hinney, “no more than 80 characters across, bro.”

But, sometimes that causes me to have to write the code in a manner that is a lot more complicated, especially when it concerns PowerShell code/functions/classes with a LOT of parameters, or switches.

Still, I do strive to keep the code to 100 characters or less in width, so there will be very minimal text wrapping between lines in the code below.
Here we go...

Class [VmAdminCredential]

```powershell
# // ======================================================
# // | Generates a random password for security purposes |
# // ======================================================

Class VmAdminCredential
{
    [String]         $UserName
    [PSCredential] $Credential
    VmAdminCredential([String]$Username)
    {
        $This.Username   = $Username
        $This.Credential = $This.SetCredential($This.Generate())
    }
    VmAdminCredential([Object]$File)
    {
        $This.Username   = "Administrator"
        $This.Credential = $This.SetCredential($This.Content($File.Fullname))
    }
    [PSCredential] SetCredential([String]$String)
    {
        Return [PSCredential]::New($This.Username,$This.Secure($String))
    }
    [SecureString] Secure([String]$In)
    {
        Return $In | ConvertTo-SecureString -AsPlainText -Force
    }
    [String] Generate()
    {
        Do
        {
            $Length          = $This.Random(10,16)
            $Bytes           = [Byte[]]::New($Length)
            ForEach ($X in 0..($Length-1))
            {
                $Bytes[$X]   = $This.Random(32,126)
            }

            $Pass            = [Char[]]$Bytes -join ''
        }
        Until ($Pass -match $This.Pattern())

        Return $Pass
    }
    [String] Content([String]$Path)
    {
        Return [System.IO.File]::ReadAllLines($Path)
    }
    [String] Pattern()
    {
        Return "(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[:punct:]).{10}"
    }
    [UInt32] Random([UInt32]$Min,[UInt32]$Max)
    {
        Return Get-Random -Min $Min -Max $Max
    }
    [String] Password()
    {
        Return $This.Credential.GetNetworkCredential().Password
    }
    [String] ToString()
    {
        Return "<FEVirtual.VmAdminCredential>"
    }
}
```

Class [VmAdminCredential]

Class [VmByteSize]

```powershell
# // ================================================================
# // | Used to convert the byte size of a drive or partition into a string |
# // ================================================================

Class VmByteSize
{
    [String]   $Name
    [UInt64]  $Bytes
    [String]   $Unit
    [String]   $Size
    VmByteSize([String]$Name,[UInt64]$Bytes)
    {
        $This.Name   = $Name
        $This.Bytes  = $Bytes
        $This.GetUnit()
        $This.GetSize()
    }
    GetUnit()
    {
        $This.Unit   = Switch ($This.Bytes)
        {
            {$_ -lt 1KB}                {     "Byte" }
            {$_ -ge 1KB -and $_ -lt 1MB} { "Kilobyte" }
            {$_ -ge 1MB -and $_ -lt 1GB} { "Megabyte" }
            {$_ -ge 1GB -and $_ -lt 1TB} { "Gigabyte" }
            {$_ -ge 1TB}                { "Terabyte" }
        }
    }
    GetSize()
    {
        $This.Size   = Switch -Regex ($This.Unit)
        {
            ^Byte     {     "{0} B" -f  $This.Bytes/1    }
            ^Kilobyte { "{0:n2} KB" -f ($This.Bytes/1KB) }
            ^Megabyte { "{0:n2} MB" -f ($This.Bytes/1MB) }
            ^Gigabyte { "{0:n2} GB" -f ($This.Bytes/1GB) }
            ^Terabyte { "{0:n2} TB" -f ($This.Bytes/1TB) }
        }
    }
    [String] ToString()
    {
        Return $This.Size
    }
}
```

Class [VmNetworkV4Ping]

```powershell
# // ============================================
# // | Object returned from a ping (sweep/scan) |
# // ============================================

Class VmNetworkV4Ping
{
    [UInt32]         $Index
    [UInt32]        $Status
    [String]          $Type = "Host"
    [String]     $IpAddress
    [String]      $Hostname
    [String[]]     $Aliases
    [String[]] $AddressList
    VmNetworkV4Ping([UInt32]$Index,[String]$IpAddress,[Object]$Reply)
    {
        $This.Index          = $Index
        $This.Status         = $Reply.Result.Status -match "Success"
        $This.IpAddress      = $IpAddress
    }
    Resolve()
    {
        $Item                = [System.Net.Dns]::Resolve($This.IpAddress)
        $This.Hostname       = $Item.Hostname
        $This.Aliases        = $Item.Aliases
        $This.AddressList    = $Item.AddressList
    }
    [String] ToString()
    {
        Return $This.IPAddress
    }
}
```

Class [VmNetworkV4Ping]

Class [VmNetworkNode]

```
# // ================================================================================
# // | Information for the network adapter in the virtual machine guest operating system |
# // ================================================================================

Class VmNetworkNode
{
    [UInt32]     $Index
    [String]      $Name
    [String] $IpAddress
    [String]    $Domain
    [String]   $NetBios
    [String]   $Trusted
    [UInt32]    $Prefix
    [String]   $Netmask
    [String]   $Gateway
    [String[]]     $Dns
    [Object]      $Dhcp
    VmNetworkNode([UInt32]$Index,[String]$Name,[String]$IpAddress,[Object]$Hive)
    {
        $This.Index     = $Index
        $This.Name      = $Name
        $This.IpAddress = $IpAddress
        $This.Domain    = $Hive.Domain
        $This.NetBios   = $Hive.NetBios
        $This.Trusted   = $Hive.Trusted
        $This.Prefix    = $Hive.Prefix
        $This.Netmask   = $Hive.Netmask
        $This.Gateway   = $Hive.Gateway
        $This.Dns       = $Hive.Dns
        $This.Dhcp      = $Hive.Dhcp
    }
    VmNetworkNode([Object]$File)
    {
        $This.Index     = $File.Index
        $This.Name      = $File.Name
        $This.IpAddress = $File.IpAddress
        $This.Domain    = $File.Domain
        $This.NetBios   = $File.NetBios
        $This.Trusted   = $File.Trusted
        $This.Prefix    = $File.Prefix
        $This.Netmask   = $File.Netmask
        $This.Gateway   = $File.Gateway
        $This.Dns       = $File.Dns
        $This.Dhcp      = $File.Dhcp
    }
    [String] ToString()
    {
        Return "<FEVirtual.VmNetwork[Node]>"
    }
}
```

Class [VmNetworkNode]

Class [VmNetworkList]

```powershell
# // ================================================================================================
# // | Creates a model for an entire single network to asynchronously ping + resolve hostnames |
# // ================================================================================================

Class VmNetworkList
{
    [UInt32]     $Index
    [String]     $Count
    [String]   $Netmask
    [String]  $Notation
    [Object]    $Output
    VmNetworkList([UInt32]$Index,[String]$Netmask,[UInt32]$Count,[String]$Notation)
    {
        $This.Index    = $Index
        $This.Count    = $Count
        $This.Netmask  = $Netmask
        $This.Notation = $Notation
        $This.Output   = @( )
    }
    Expand()
    {
        $Split     = $This.Notation.Split("/")
        $HostRange = @{ }
        ForEach ($0 in $Split[0] | Invoke-Expression)
        {
            ForEach ($1 in $Split[1] | Invoke-Expression)
            {
                ForEach ($2 in $Split[2] | Invoke-Expression)
                {
                    ForEach ($3 in $Split[3] | Invoke-Expression)
                    {
                        $HostRange.Add($HostRange.Count,"$0.$1.$2.$3")
                    }
                }
            }
        }

        $This.Output    = $HostRange[0..($HostRange.Count-1)]
    }
    [String] ToString()
    {
        Return "<FEVirtual.VmNetwork[List]>"
    }
}
```

Class [VmNetworkList]

Class [VmNetworkDhcp]

```powershell
# // ============================================================
# // | Casts entire range of Dhcp (hosts/addresses/exclusions) |
# // ============================================================

Class VmNetworkDhcp
{
    [String]          $Name
    [String]    $SubnetMask
    [String]       $Network
    [String]    $StartRange
    [String]      $EndRange
    [String]     $Broadcast
    [String[]]   $Exclusion
    VmNetworkDhcp([Object]$Network)
    {
        $This.Name        = "{0}/{1}" -f ($Network.Hosts | ? Type -eq Network), $Network.Prefix
        $This.SubnetMask  = $Network.Netmask
        $This.Network     = $Network.Hosts | ? Type -eq Network
        $Range            = $Network.Hosts | ? Type -eq Host
        $This.StartRange  = $Range[0].IpAddress
        $This.EndRange    = $Range[-1].IpAddress
        $This.Broadcast   = $Network.Hosts | ? Type -eq Broadcast
        $This.Exclusion   = $Range | ? Status | % IpAddress
    }
    [String] ToString()
    {
        Return "<FEVirtual.VmNetwork[Dhcp]>"
    }
}
```

Class [VmNetworkDhcp]

Class [VmNetworkController] /----------\

```
# // ================================================================================
# // | Creates a hive of possible VM network nodes based on prefix length + network address |
# // ================================================================================

Class VmNetworkController
{
    Hidden [Object]   $Config
    [String]          $Domain
    [String]         $NetBios
    [String]         $Trusted
    [UInt32]          $Prefix
    [String]         $Netmask
    [String]        $Wildcard
    [String]         $Gateway
    [String[]]           $Dns
    [Object]            $Dhcp
    [Object]        $Networks
    [Object]           $Hosts
    [Object]           $Nodes
    VmNetworkController([Object]$Config,[String]$Domain,[String]$NetBios)
    {
        $This.Config    = $Config
        $This.Domain    = $Domain
        $This.NetBios   = $NetBios
        $This.Trusted   = $This.Config.IPV4Address.IpAddress.ToString()
        $This.Prefix    = $This.Config.IPv4Address.PrefixLength

        $This.GetConversion()

        $This.Gateway   = $This.Config.IPV4DefaultGateway.NextHop
        $This.Dns       = $This.Config.DnsServer | ? AddressFamily -eq 2 | % ServerAddresses

        $This.Networks  = @( )
        $This.Hosts     = @( )
        $This.Nodes     = @( )

        $This.GetNetworkLists()

        $This.Dhcp      = $This.VmNetworkDhcp()
    }
    [Object] VmNetworkNode([UInt32]$Index,[String]$Name,[String]$IpAddress,[Object]$Hive)
    {
        Return [VmNetworkNode]::New($Index,$Name,$IpAddress,$Hive)
    }
    [Object] VmNetworkList([UInt32]$Index,[String]$Netmask,[UInt32]$Count,[String]$Notation)
    {
        Return [VmNetworkList]::New($Index,$Netmask,$Count,$Notation)
    }
    [Object] VmNetworkDhcp()
    {
        Return [VmNetworkDhcp]::New($This)
    }
    AddNode([String]$Name)
    {
        $Item              = ($This.Hosts | ? Type -eq Host | ? Status -eq 0)[0]
        $This.Nodes       += $This.VmNetworkNode($This.Nodes.Count,$Name,$Item.IpAddress,$This)
        If ($Name -in $This.Nodes.Name)
        {
            $Item          = $This.Hosts[$Item.Index]
            $Item.Status   = 1
            $Item.Hostname = $Name
            [Console]::WriteLine("[+] Node [$Name] added")
        }
    }
    AddList([UInt32]$Count,[String]$Notation)
    {
        $This.Networks += $This.VmNetworkList($This.Networks.Count,$This.Netmask,$Count,$Notation)
    }
```

```
        GetConversion()
        {
            # Convert IP and PrefixLength into binary, netmask, and wildcard
            $xBinary       = 0..3 | % { (($_*8)..(($_*8)+7) | % { @(0,1)[$_ -lt $This.Prefix] }) -join '' }
            $This.Netmask  = ($xBinary | % { [Convert]::ToInt32($_,2 ) }) -join "."
            $This.Wildcard = ($This.Netmask.Split(".") | % { (256-$_) }) -join "."
        }
        GetNetworkLists()
        {
            $Address      = $This.Trusted.Split(".")

            $xNetmask     = $This.Netmask  -split "\."
            $xWildCard    = $This.Wildcard -split "\."
            $Total        = $xWildcard -join "*" | Invoke-Expression

            # Convert wildcard into total host range
            $Hash         = @{ }
            ForEach ($X in 0..3)
            {
                $Value = Switch ($xWildcard[$X])
                {
                    1
                    {
                        $Address[$X]
                    }
                    Default
                    {
                        ForEach ($Item in 0..255 | ? { $_ % $xWildcard[$X] -eq 0 })
                        {
                            "{0}..{1}" -f $Item, ($Item+($xWildcard[$X]-1))
                        }
                    }
                    255
                    {
                        "{0}..{1}" -f $xNetmask[$X],($xNetmask[$X]+$xWildcard[$X])
                    }
                }

                $Hash.Add($X,$Value)
            }

            # Build host range
            $xRange   = @{ }
            ForEach ($0 in $Hash[0])
            {
                ForEach ($1 in $Hash[1])
                {
                    ForEach ($2 in $Hash[2])
                    {
                        ForEach ($3 in $Hash[3])
                        {
                            $xRange.Add($xRange.Count,"$0/$1/$2/$3")
                        }
                    }
                }
            }

            Switch ($xRange.Count)
            {
                0
                {
                    "Error"
                }
                1
                {
                    $This.AddList($Total,$xRange[0])
                }
                Default
                {
                    ForEach ($X in 0..($xRange.Count-1))
                    {
                        $This.AddList($Total,$xRange[$X])
```

```
                    }
                }
            }

            # Subtract network + broadcast addresses
            ForEach ($Network in $This.Networks)
            {
                $Network.Expand()
                If ($This.Trusted -in $Network.Output)
                {
                    $This.Hosts          = $This.V4PingSweep($Network)
                    $This.Hosts[ 0].Type = "Network"
                    $This.Hosts[-1].Type = "Broadcast"
                }
                Else
                {
                    $Network.Output = $Null
                }
            }
        }
        Resolve()
        {
            If ($This.Output.Count -gt 2)
            {
                ForEach ($Item in $This.Hosts | ? Status)
                {
                    $Item.Resolve()
                }
            }
        }
        [Object] V4PingOptions()
        {
            Return [System.Net.NetworkInformation.PingOptions]::New()
        }
        [Object] V4PingBuffer()
        {
            Return 97..119 + 97..105 | % { "0x{0:X}" -f $_ }
        }
        [Object] V4Ping([String]$Ip)
        {
            $Item = [System.Net.NetworkInformation.Ping]::New()
            Return $Item.SendPingAsync($Ip,100,$This.V4PingBuffer(),$This.V4PingOptions())
        }
        [Object] V4PingResponse([UInt32]$Index,[Object]$Ip,[Object]$Ping)
        {
            Return [VmNetworkV4Ping]::New($Index,$Ip,$Ping)
        }
        [Object[]] V4PingSweep([Object]$Network)
        {
            $Ping                = @{ }
            $Response            = @{ }
            ForEach ($X in 0..($Network.Output.Count-1))
            {
                $Ping.Add($Ping.Count,$This.V4Ping($Network.Output[$X]))
            }

            ForEach ($X in 0..($Ping.Count-1))
            {
                $Response.Add($X,$This.V4PingResponse($X,$Network.Output[$X],$Ping[$X]))
            }

            Return $Response[0..($Response.Count-1)]
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNetwork[Controller]>"
        }
    }
```

_____/ Class [VmNetworkController]

Class [VmNodeItem]

```
# // ========================================
# // | Holds information for a virtual machine |
# // ========================================

Class VmNodeItem
{
    [UInt32]      $Index
    [Object]       $Name
    [Object]     $Memory
    [Object]       $Path
    [Object]        $Vhd
    [Object]    $VhdSize
    [Object] $Generation
    [UInt32]       $Core
    [Object] $SwitchName
    [Object]    $Network
    VmNodeItem([Object]$Node,[Object]$Hive)
    {
        $This.Index      = $Node.Index
        $This.Name       = $Node.Name
        $This.Memory     = $Hive.Memory
        $This.Path       = $Hive.Base, $This.Name -join '\'
        $This.Vhd        = "{0}\{1}\{1}.vhdx" -f $Hive.Base, $This.Name
        $This.VhdSize    = $This.Size("HDD",$Hive.HDD)
    }
    [String] ToString()
    {
        Return "<FEVirtual.VmNode[Item]>"
    }
}
```

Class [VmNodeItem]

Class [VmNodeTemplate]

```
    # // ===================================================================
    # // | Carries standard template information for a new virtual machine |
    # // ===================================================================

    Class VmNodeTemplate
    {
        [String]     $Base
        [UInt64]   $Memory
        [UInt64]      $Hdd
        [UInt32]      $Gen
        [UInt32]     $Core
        [String] $SwitchId
        [String]    $Image
        VmNodeTemplate([String]$Path,[UInt64]$Ram,[UInt64]$Hdd,[UInt32]$Gen,[UInt32]$Core,[String]$Switch,
[String]$Img)
        {
            $This.Base     = $Path
            $This.Memory   = $Ram
            $This.Hdd      = $Hdd
            $This.Gen      = $Gen
            $This.Core     = $Core
            $This.SwitchId = $Switch
            $This.Image    = $Img
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNode[Template]>"
        }
    }
```

Class [VmNodeTemplate]

Class [VmNodeFile]

```
# // ================================================================
# // | Writes the network and virtual machine node information to a file |
# // ================================================================

Class VmNodeFile
{
    [UInt32]     $Index
    [String]      $Name
    [String] $IpAddress
    [String]    $Domain
    [String]   $NetBios
    [String]   $Trusted
    [UInt32]    $Prefix
    [String]   $Netmask
    [String]   $Gateway
    [String[]]     $Dns
    [Object]      $Dhcp
    [String]      $Base
    [UInt64]    $Memory
    [UInt64]       $Hdd
    [UInt32]       $Gen
    [UInt32]      $Core
    [String]  $SwitchId
    [String]     $Image
    VmNodeFile([Object]$Node,[Object]$Template)
    {
        $This.Index     = $Node.Index
        $This.Name      = $Node.Name
        $This.IpAddress = $Node.IpAddress
        $This.Domain    = $Node.Domain
        $This.NetBios   = $Node.NetBios
        $This.Trusted   = $Node.Trusted
        $This.Prefix    = $Node.Prefix
        $This.Netmask   = $Node.Netmask
        $This.Gateway   = $Node.Gateway
        $This.Dns       = $Node.Dns
        $This.Dhcp      = $Node.Dhcp
        $This.Base      = $Template.Base
        $This.Memory    = $Template.Memory
        $This.Hdd       = $Template.Hdd
        $This.Gen       = $Template.Gen
        $This.Core      = $Template.Core
        $This.SwitchId  = $Template.SwitchId
        $This.Image     = $Template.Image
    }
    [String] ToString()
    {
        Return "<FEVirtual.VmNode[File]>"
    }
}
```

Class [VmNodeFile]

Class [VmNodeController] /----------\
/----------

```
    # // ================================================================================
    # // | Spins up all of the necessary information for a range of VM's w/ network info |
    # // ================================================================================

    Class VmNodeController
    {
        [String]     $Path
        [String]   $Domain
        [String]  $NetBios
        [Object]    $Admin
        [Object]   $Config
        [Object]  $Network
        [Object] $Template
        VmNodeController([String]$Path,[String]$Domain,[String]$NetBios)
        {
            If (![System.IO.Directory]::Exists($Path))
            {
                [System.IO.Directory]::CreateDirectory($Path)
            }

            $This.Path     = $Path
            $This.Domain   = $Domain
            $This.NetBios  = $NetBios
            $This.Admin    = $This.NewVmAdminCredential()
            $This.Config   = $This.GetNetIPConfiguration()
            $This.Network  = $This.NewVmNetworkController()
        }
        VmNodeController([String]$Path,[String]$IpAddress,[UInt32]$Prefix,[String]$Gateway,[String[]]$Dns,
[String]$Domain,[String]$NetBios)
        {
            $This.Path     = $Path
            $This.Domain   = $Domain
            $This.NetBios  = $NetBios
            $This.Admin    = $This.NewVmAdminCredential()
            $This.Config   = $null
            $This.Network  = $This.NewVmNetworkController($IpAddress,$Prefix,$Gateway,$Dns,$Domain,$NetBios)
        }
        [Object] NewVmAdminCredential()
        {
            Return [VmAdminCredential]::New("Administrator")
        }
        [Object] GetNetIPConfiguration()
        {
            Return Get-NetIPConfiguration -Detailed | ? IPV4DefaultGateway | Select-Object -First 1
        }
        [Object] NewVmNetworkController()
        {
            Return [VmNetworkController]::New($This.Config,$This.Domain,$This.NetBios)
        }
        [Object] NewVmNetworkController([String]$IpAddress,[UInt32]$Prefix,[String]$Gateway,[String[]]$Dns,
[String]$Domain,[String]$NetBios)
        {
            Return [VmNetworkController]::New($IpAddress,$Prefix,$Gateway,$Dns,$Domain,$NetBios)
        }
        [Object] NewVmTemplate([String]$Base,[UInt64]$Ram,[UInt64]$Hdd,[Uint32]$Generation,[UInt32]$Core,
[String]$VMSwitch,[String]$Path)
        {
            Return [VmNodeTemplate]::New($Base,$Ram,$Hdd,$Generation,$Core,$VmSwitch,$Path)
        }
        SetTemplate([String]$Base,[UInt64]$Ram,[UInt64]$Hdd,[Uint32]$Generation,[UInt32]$Core,[String]$VMSwitch,
[String]$Path)
        {
            $This.Template = $This.NewVmTemplate($Base,$Ram,$Hdd,$Generation,$Core,$VmSwitch,$Path)
        }
        [Object] NewVmObjectFile([Object]$Node)
        {
            Return [VmNodeFile]::New($Node,$This.Template)
        }
```

```
        AddNode([String]$Name)
        {
            If ($Name -notin $This.Network.Nodes)
            {
                $This.Network.AddNode($Name)
            }
        }
        Export()
        {
            ForEach ($Node in $This.Network.Nodes)
            {
                $FilePath = "{0}\{1}.txt" -f $This.Path, $Node.Name
                $Value    = $This.NewVmObjectFile($Node) | ConvertTo-Json

                [System.IO.File]::WriteAllLines($FilePath,$Value)

                If ([System.IO.File]::Exists($FilePath))
                {
                    [Console]::WriteLine("Exported  [+] File: [$FilePath]")
                }
                Else
                {
                    Throw "Something failed... bye."
                }
            }
        }
        WriteAdmin()
        {
            $FilePath = "{0}\admin.txt" -f $This.Path
            $Value    = $This.Admin.Credential.GetNetworkCredential().Password
            [System.IO.File]::WriteAllLines($FilePath,$Value)
            If ([System.IO.File]::Exists($FilePath))
            {
                [Console]::WriteLine("Exported  [+] File: [$FilePath]")
            }
            Else
            {
                Throw "Something failed... bye."
            }
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmNode[Controller]>"
        }
    }
```

_____/ Class [VmNodeController]

Class [VmNodeInputObject]

```
# // ================================================================================
# // | Instructions for each of these machines to be able to turn output file into VM |
# // ================================================================================

Class VmNodeInputObject
{
    [String]   $Path
    [Object] $Object
    [Object]  $Admin
    VmNodeInputObject([String]$Token,[String]$Path)
    {
        $This.Path   = $Path
        $This.Object = $This.SetObject($Token)
        $This.Admin  = $This.SetAdmin()
    }
    [String] GetChildItem([String]$Name)
    {
        $File = Get-ChildItem $This.Path | ? Name -eq $Name

        If (!$File)
        {
            Throw "Invalid entry"
        }

        Return $File.Fullname
    }
    [Object] SetObject([String]$Token)
    {
        $File        = $This.GetChildItem($Token)
        If (!$File)
        {
            Throw "Invalid token"
        }

        Return [System.IO.File]::ReadAllLines($File) | ConvertFrom-Json
    }
    [PSCredential] SetAdmin()
    {
        $File        = $This.GetChildItem("admin.txt")
        If (!$File)
        {
            Throw "No password detected"
        }

        Return [PSCredential]::New("Administrator",$This.GetPassword($File))
    }
    [SecureString] GetPassword([String]$File)
    {
        Return [System.IO.File]::ReadAllLines($File) | ConvertTo-SecureString -AsPlainText -Force
    }
    [String] ToString()
    {
        Return "<FEVirtual.VmNode[InputObject]>"
    }
}
```

Class [VmScriptBlockLine]

```
# // ================================
# // | Virtual machine script line |
# // ================================

Class VmScriptBlockLine
{
    [UInt32] $Index
    [String]  $Line
    VmScriptBlockLine([UInt32]$Index,[String]$Line)
    {
        $This.Index = $Index
        $This.Line  = $Line
    }
    [String] ToString()
    {
        Return $This.Line
    }
}
```

Class [VmScriptBlockLine]

Class [VmScriptBlockItem]

```
# // ================================
# // | Virtual machine script block |
# // ================================

Class VmScriptBlockItem
{
    [UInt32]       $Index
    [UInt32]       $Phase
    [String]        $Name
    [String] $DisplayName
    [Object]     $Content
    [UInt32]    $Complete
    VmScriptBlockItem([UInt32]$Index,[UInt32]$Phase,[String]$Name,[String]$DisplayName,[String[]]$Content)
    {
        $This.Index       = $Index
        $This.Phase       = $Phase
        $This.Name        = $Name
        $This.DisplayName = $DisplayName

        $This.Load($Content)
    }
    Clear()
    {
        $This.Content     = @( )
    }
    Load([String[]]$Content)
    {
        $This.Clear()
        $This.Add("# $($This.DisplayName)")

        ForEach ($Line in $Content)
        {
            $This.Add($Line)
        }

        $This.Add('')
    }
    [Object] VmScriptBlockLine([UInt32]$Index,[String]$Line)
    {
        Return [VmScriptBlockLine]::New($Index,$Line)
    }
    Add([String]$Line)
    {
        $This.Content += $This.VmScriptBlockLine($This.Content.Count,$Line)
    }
    [String] ToString()
    {
        Return "<FEVirtual.VmScriptBlock[Item]>"
    }
}
```

Class [VmScriptBlockItem]

Class [VmScriptBlockController]

```
    # // ========================================
    # // | Virtual machine script block controller |
    # // ========================================

    Class VmScriptBlockController
    {
        [String]     $Name
        [UInt32] $Selected
        [UInt32]    $Count
        [Object]   $Output
        VmScriptBlockController()
        {
            $This.Name = "ScriptBlock[Controller]"
            $This.Clear()
        }
        Clear()
        {
            $This.Output = @( )
            $This.Count  = 0
        }
        [Object] VmScriptBlockItem([UInt32]$Index,[UInt32]$Phase,[String]$Name,[String]$DisplayName,[String[]]
$Content)
        {
            Return [VmScriptBlockItem]::New($Index,$Phase,$Name,$DisplayName,$Content)
        }
        Add([String]$Phase,[String]$Name,[String]$DisplayName,[String[]]$Content)
        {
            $This.Output += $This.VmScriptBlockItem($This.Output.Count,$Phase,$Name,$DisplayName,$Content)
            $This.Count   = $This.Output.Count
        }
        Select([UInt32]$Index)
        {
            If ($Index -gt $This.Count)
            {
                Throw "Invalid index"
            }

            $This.Selected = $Index
        }
        [Object] Current()
        {
            Return $This.Output[$This.Selected]
        }
        [Object] Get([String]$Name)
        {
            Return $This.Output | ? Name -eq $Name
        }
        [Object] Get([UInt32]$Index)
        {
            Return $This.Output | ? Index -eq $Index
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmScriptBlock[Controller]>"
        }
    }
```

Class [VmScriptBlockController]

Class [VmPropertyItem]

```powershell
# // ====================================
# // | Virtual machine "Get-VM" property |
# // ====================================

Class VmPropertyItem
{
    [UInt32] $Index
    [String] $Name
    [Object] $Value
    VmPropertyItem([UInt32]$Index,[Object]$Property)
    {
        $This.Index = $Index
        $This.Name  = $Property.Name
        $This.Value = $Property.Value
    }
    [String] ToString()
    {
        Return "<FEVirtual.VmProperty[Item]>"
    }
}
```

Class [VmPropertyItem]

Class [VmPropertyList]

```powershell
# // ====================================
# // | Virtual machine "Get-VM" property |
# // ====================================

Class VmPropertyList
{
    [String] $Name
    [UInt32] $Count
    [Object] $Output
    VmPropertyList()
    {
        $This.Name = "VmProperty[List]"
    }
    Clear()
    {
        $This.Output = @( )
    }
    [Object] VmPropertyItem([UInt32]$Index,[Object]$Property)
    {
        Return [VmPropertyItem]::($Index,$Property)
    }
    Add([Object]$Property)
    {
        $This.Output += $This.VmPropertyItem($This.Output.Count,$Property)
        $This.Count   = $This.Output.Count
    }
    [String] ToString()
    {
        Return "({0}) <FEVirtual.VmProperty[List]>" -f $This.Count
    }
}
```

Class [VmPropertyList]

Class [VmObject] /----------\
/----------

```
# // ================================================================================
# // | Virtual Machine controller for Hyper-V, end result of the above classes |
# // ================================================================================

Class VmObject
{
    Hidden [UInt32]     $Mode
    [Object]         $Console
    [Object]            $Name
    [Object]          $Memory
    [Object]            $Path
    [Object]             $Vhd
    [Object]         $VhdSize
    [Object]      $Generation
    [UInt32]            $Core
    [Object]          $Switch
    [Object]        $Firmware
    [UInt32]          $Exists
    [Object]            $Guid
    [Object]         $Network
    [String]             $Iso
    [Object]          $Script
    Hidden [Object] $Property
    Hidden [Object]  $Control
    Hidden [Object] $Keyboard
    VmObject([Switch]$Flags,[Object]$Vm)
    {
        # Meant for removal if found
        $This.Mode       = 1
        $This.StartConsole()

        $This.Name       = $Vm.Name
        $Item            = $This.Get()
        If (!$Item)
        {
            Throw "Vm does not exist"
        }

        $This.Memory     = $This.Size("Ram",$Item.MemoryStartup)
        $This.Path       = $Item.Path | Split-Path
        $This.Vhd        = $Item.HardDrives[0].Path
        $This.VhdSize    = $This.Size("Hdd",(Get-Vhd $This.Vhd).Size)
        $This.Generation = $Item.Generation
        $This.Core       = $Item.ProcessorCount
        $This.Switch     = @($Item.NetworkAdapters[0].SwitchName)
        $This.Firmware   = $This.GetVmFirmware()
    }
    VmObject([Object]$File)
    {
        # Meant to build a new VM
        $This.Mode       = 1
        $This.StartConsole()

        $This.Name       = $File.Name
        If ($This.Get())
        {
            Throw "Vm already exists"
        }

        $This.Memory     = $This.Size("Ram",$File.Memory)
        $This.Path       = "{0}\{1}" -f $File.Base, $This.Name
        $This.Vhd        = "{0}\{1}\{1}.vhdx" -f $File.Base, $This.Name
        $This.VhdSize    = $This.Size("Hdd",$File.HDD)
        $This.Generation = $File.Gen
        $This.Core       = $File.Core
        $This.Switch     = @($File.SwitchId)
        $This.Network    = $This.GetNetworkNode($File)
        $This.Iso        = $File.Image
```

```
        }
        StartConsole()
        {
            # Instantiates and initializes the console
            $This.Console = New-FEConsole
            $This.Console.Initialize()
            $This.Status()
        }
        [Void] Status()
        {
            # If enabled, shows the last item added to the console
            If ($This.Mode -gt 0)
            {
                [Console]::WriteLine($This.Console.Last())
            }
        }
        [Void] Update([Int32]$State,[String]$Status)
        {
            # Updates the console
            $This.Console.Update($State,$Status)
            $This.Status()
        }
        Error([UInt32]$State,[String]$Status)
        {
            $This.Console.Update($State,$Status)
            Throw $This.Console.Last().Status
        }
        [Object] Get()
        {
            $Virtual           = Get-VM -Name $This.Name -EA 0
            $This.Exists       = $Virtual.Count -gt 0
            $This.Guid         = @($Null,$Virtual.Id)[$This.Exists]

            Return @($Null,$Virtual)[$This.Exists]
        }
        [Object] Size([String]$Name,[UInt64]$SizeBytes)
        {
            Return [VmByteSize]::New($Name,$SizeBytes)
        }
        [String] Hostname()
        {
            Return [Environment]::MachineName
        }
        Connect()
        {
            $This.Update(0,"[~] Connecting : $($This.Name)")
            $Splat           = @{

                Filepath     = "vmconnect"
                ArgumentList = @($This.Hostname(),$This.Name)
                Verbose      = $True
                PassThru     = $True
            }

            Start-Process @Splat
        }
        New()
        {
            $Null = $This.Get()
            If ($This.Exists -ne 0)
            {
                $This.Error(-1,"[!] Exists : $($This.Name)")
            }

            $Object                = @{

                Name               = $This.Name
                MemoryStartupBytes = $This.Memory.Bytes
                Path               = $This.Path
                NewVhdPath         = $This.Vhd
                NewVhdSizeBytes    = $This.VhdSize.Bytes
                Generation         = $This.Generation
```

```
            SwitchName          = $This.Switch[0]
        }

        $This.Update(0,"[~] Creating : $($This.Name)")

        # Verbosity level
        Switch ($This.Mode)
        {
            Default { New-VM @Object }
            2       { New-VM @Object -Verbose }
        }

        # Verbosity level
        Switch ($This.Mode)
        {
            Default { Enable-VmResourceMetering -VmName $This.Name }
            2       { Enable-VmResourceMetering -VmName $This.Name -Verbose }
        }

        $Item                   = $This.Get()
        $This.Firmware          = $This.GetVmFirmware()
        $This.SetVMProcessor()

        $This.Script            = $This.NewVmScriptBlockController()
        $This.Property          = $This.NewVmPropertyList()

        ForEach ($Property in $Item.PSObject.Properties)
        {
            $This.Property.Add($Property)
        }
    }
    Start()
    {
        $Vm = $This.Get()
        If (!$Vm)
        {
            $This.Error(-1,"[!] Exception : $($This.Name) [does not exist]")
        }

        ElseIf ($Vm.State -eq "Running")
        {
            $This.Error(-1,"[!] Exception : $($This.Name) [already started]")
        }

        Else
        {
            $This.Update(1,"[~] Starting : $($This.Name)")

            # Verbosity level
            Switch ($This.Mode)
            {
                Default { $Vm | Start-VM }
                2       { $Vm | Start-VM -Verbose }
            }
        }
    }
    Stop()
    {
        $Vm = $This.Get()
        If (!$Vm)
        {
            $This.Error(-1,"[!] Exception : $($This.Name) [does not exist]")
        }

        ElseIf ($Vm.State -ne "Running")
        {
            $This.Error(-1,"[!] Exception : $($This.Name) [not running]")
        }

        Else
        {
            $This.Update(0,"[~] Stopping : $($This.Name)")
```

```
            # Verbosity level
            Switch ($This.Mode)
            {
                Default { $This.Get() | ? State -ne Off | Stop-VM -Force }
                2       { $This.Get() | ? State -ne Off | Stop-VM -Force -Verbose }
            }
        }
    }
    Reset()
    {
        $Vm = $This.Get()
        If (!$Vm)
        {
            $This.Error(-1,"[!] Exception : $($This.Name) [does not exist]")
        }

        ElseIf ($Vm.State -ne "Running")
        {
            $This.Error(-1,"[!] Exception : $($This.Name) [not running]")
        }

        Else
        {
            $This.Update(0,"[~] Restarting : $($This.Name)")
            $This.Stop()
            $This.Start()
            $This.Idle(5,5)
        }
    }
    Remove()
    {
        $Vm = $This.Get()
        If (!$Vm)
        {
            $This.Error(-1,"[!] Exception : $($This.Name) [does not exist]")
        }

        $This.Update(0,"[~] Removing : $($This.Name)")

        If ($Vm.State -ne "Off")
        {
            $This.Update(0,"[~] State : $($This.Name) [attempting shutdown]")
            Switch -Regex ($Vm.State)
            {
                "(^Paused$|^Saved$)"
                {
                    $This.Start()
                    Do
                    {
                        Start-Sleep 1
                    }
                    Until ($This.Get().State -eq "Running")
                }
            }

            $This.Stop()
            Do
            {
                Start-Sleep 1
            }
            Until ($This.Get().State -eq "Off")
        }

        # Verbosity level
        Switch ($This.Mode)
        {
            Default { $This.Get() | Remove-VM -Confirm:$False -Force -EA 0 }
            2       { $This.Get() | Remove-VM -Confirm:$False -Force -Verbose -EA 0 }
        }

        $This.Firmware         = $Null
```

```
            $This.Exists           = 0

            $This.Update(0,"[~] Vhd  : [$($This.Vhd)]")

            # Verbosity level
            Switch ($This.Mode)
            {
                Default { Remove-Item $This.Vhd -Confirm:$False -Force -EA 0 }
                2       { Remove-Item $This.Vhd -Confirm:$False -Force -Verbose -EA 0 }
            }

            $This.Update(0,"[~] Path : [$($This.Path)]")
            ForEach ($Item in Get-ChildItem $This.Path -Recurse | Sort-Object -Descending)
            {
                $This.Update(0,"[~] $($Item.Fullname)")

                # Verbosity level
                Switch ($This.Mode)
                {
                    Default { Remove-Item $Item.Fullname -Confirm:$False -EA 0 }
                    2       { Remove-Item $Item.Fullname -Confirm:$False -Verbose -EA 0 }
                }
            }

            $This.Update(1,"[ ] Removed : $($Item.Fullname)")

            $This.DumpConsole()
        }
        [Object] Measure()
        {
            If (!$This.Exists)
            {
                Throw "Cannot measure a virtual machine when it does not exist"
            }

            Return Measure-Vm -Name $This.Name
        }
        [Object] Wmi([String]$Type)
        {
            Return Get-WmiObject $Type -NS Root\Virtualization\V2
        }
        [Object] NewVmPropertyList()
        {
            Return [VmPropertyList]::New()
        }
        [Object] NewVmScriptBlockController()
        {
            Return [VmScriptBlockController]::New()
        }
        [Object] GetVmFirmware()
        {
            $This.Update(0,"[~] Getting VmFirmware : $($This.Name)")
            $Item = Switch ($This.Generation)
            {
                1
                {
                    # Verbosity level
                    Switch ($This.Mode)
                    {
                        Default { Get-VmBios -VmName $This.Name }
                        2       { Get-VmBios -VmName $This.Name -Verbose }
                    }
                }
                2
                {
                    # Verbosity level
                    Switch ($This.Mode)
                    {
                        Default { Get-VmFirmware -VmName $This.Name }
                        2       { Get-VmFirmware -VmName $This.Name -Verbose }
                    }
                }
```

```
        }

        Return $Item
    }
    SetVmProcessor()
    {
        $This.Update(0,"[~] Setting VmProcessor (Count): [$($This.Core)]")

        # Verbosity level
        Switch ($This.Mode)
        {
            Default { Set-VmProcessor -VMName $This.Name -Count $This.Core }
            2       { Set-VmProcessor -VMName $This.Name -Count $This.Core -Verbose }
        }
    }
    SetVmDvdDrive([String]$Path)
    {
        If (![System.IO.File]::Exists($Path))
        {
            $This.Error(-1,"[!] Invalid path : [$Path]")
        }

        $This.Update(0,"[~] Setting VmDvdDrive (Path): [$Path]")

        # Verbosity level
        Switch ($This.Mode)
        {
            Default { Set-VmDvdDrive -VMName $This.Name -Path $Path }
            2       { Set-VmDvdDrive -VMName $This.Name -Path $Path -Verbose }
        }
    }
    SetVmBootOrder([UInt32]$1,[UInt32]$2,[UInt32]$3)
    {
        $This.Update(0,"[~] Setting VmFirmware (Boot order) : [$1,$2,$3]")

        $Fw = $This.GetVmFirmware()

        # Verbosity level
        Switch ($This.Mode)
        {
            Default { Set-VMFirmware -VMName $This.Name -BootOrder $Fw.BootOrder[$1,$2,$3] }
            2       { Set-VMFirmware -VMName $This.Name -BootOrder $Fw.BootOrder[$1,$2,$3] -Verbose }
        }
    }
    AddVmDvdDrive()
    {
        $This.Update(0,"[+] Adding VmDvdDrive")

        # Verbosity level
        Switch ($This.Mode)
        {
            Default { Add-VmDvdDrive -VMName $This.Name }
            2       { Add-VmDvdDrive -VMName $This.Name -Verbose }
        }
    }
    LoadIso([String]$Path)
    {
        If (![System.IO.File]::Exists($Path))
        {
            $This.Error(-1,"[!] Invalid ISO path : [$Path]")
        }

        Else
        {
            $This.Iso = $Path
            $This.SetVmDvdDrive($This.Iso)
        }
    }
    UnloadIso()
    {
        $This.Update(0,"[+] Unloading ISO")
```

```
            # Verbosity level
            Switch ($This.Mode)
            {
                Default { Set-VmDvdDrive -VMName $This.Name -Path $Null }
                2       { Set-VmDvdDrive -VMName $This.Name -Path $Null -Verbose }
            }
        }
        SetIsoBoot()
        {
            If (!$This.Iso)
            {
                $This.Error(-1,"[!] No (*.iso) file loaded")
            }

            ElseIf ($This.Generation -eq 2)
            {
                $This.SetVmBootOrder(2,0,1)
            }
        }
        TypeChain([UInt32[]]$Array)
        {
            ForEach ($Key in $Array)
            {
                $This.TypeKey($Key)
                Start-Sleep -Milliseconds 125
            }
        }
        TypeKey([UInt32]$Index)
        {
            $This.Update(0,"[+] Typing key : [$Index]")
            $This.Keyboard.TypeKey($Index)
            Start-Sleep -Milliseconds 125
        }
        TypeText([String]$String)
        {
            $This.Update(0,"[+] Typing text : [$String]")
            $This.Keyboard.TypeText($String)
            Start-Sleep -Milliseconds 125
        }
        TypePassword([String]$Pass)
        {
            $This.Update(0,"[+] Typing password : [ActualPassword]")
            $This.Keyboard.TypeText($Pass)
            Start-Sleep -Milliseconds 125
        }
        PressKey([UInt32]$Index)
        {
            $This.Update(0,"[+] Pressing key : [$Index]")
            $This.Keyboard.PressKey($Index)
        }
        ReleaseKey([UInt32]$Index)
        {
            $This.Update(0,"[+] Releasing key : [$Index]")
            $This.Keyboard.ReleaseKey($Index)
        }
        SpecialKey([UInt32]$Index)
        {
            $This.Keyboard.PressKey(18)
            $This.Keyboard.TypeKey($Index)
            $This.Keyboard.ReleaseKey(18)
        }
        TypeCtrlAltDel()
        {
            $This.Update(0,"[+] Typing (CTRL + ALT + DEL)")
            $This.Keyboard.TypeCtrlAltDel()
        }
        Idle([UInt32]$Percent,[UInt32]$Seconds)
        {
            $This.Update(0,"[~] Idle : $($This.Name) [CPU <= $Percent% for $Seconds second(s)]")

            $C = 0
            Do
```

```
            {
                Switch ([UInt32]($This.Get().CpuUsage -le $Percent))
                {
                    0 { $C = 0 } 1 { $C ++ }
                }

                Start-Sleep -Seconds 1
            }
            Until ($C -ge $Seconds)

            $This.Update(1,"[+] Idle complete")
        }
        Uptime([UInt32]$Mode,[UInt32]$Seconds)
        {
            $Mark = @("<=",">=")[$Mode]
            $Flag = 0
            $This.Update(0,"[~] Uptime : $($This.Name) [Uptime $Mark $Seconds second(s)]")
            Do
            {
                Start-Sleep -Seconds 1
                $Uptime        = $This.Get().Uptime.TotalSeconds
                [UInt32] $Flag = Switch ($Mode) { 0 { $Uptime -le $Seconds } 1 { $Uptime -ge $Seconds } }
            }
            Until ($Flag)
            $This.Update(1,"[+] Uptime complete")
        }
        Timer([UInt32]$Seconds)
        {
            $This.Update(0,"[~] Timer : $($This.Name) [Span = $Seconds]")

            $C = 0
            Do
            {
                Start-Sleep -Seconds 1
                $C ++
            }
            Until ($C -ge $Seconds)

            $This.Update(1,"[+] Timer")
        }
        Connection()
        {
            $This.Update(0,"[~] Connection : $($This.Name) [Await response]")

            Do
            {
                Start-Sleep 1
            }
            Until (Test-Connection $This.Network.IpAddress -EA 0)

            $This.Update(1,"[+] Connection")
        }
        SetAdmin([Object]$Admin)
        {
            $This.Update(0,"[~] Setting : Administrator password")
            ForEach ($X in 0..1)
            {
                $This.TypePassword($Admin.Password())
                $This.TypeKey(9)
                Start-Sleep -Milliseconds 125
            }

            $This.TypeKey(9)
            Start-Sleep -Milliseconds 125
            $This.TypeKey(13)
        }
        Login([Object]$Admin)
        {
            If ($Admin.GetType().Name -ne "VmAdminCredential")
            {
                $This.Error("[!] Invalid input object")
            }
```

```
            $This.Update(0,"[~] Login : Administrator")
            $This.TypeCtrlAltDel()
            $This.Timer(5)
            $This.TypePassword($Admin.Password())
            Start-Sleep -Milliseconds 125
            $This.TypeKey(13)
        }
        LaunchPs()
        {
            # Open Start Menu
            $This.PressKey(91)
            $This.TypeKey(88)
            $This.ReleaseKey(91)
            $This.Timer(1)

            # Open Command Prompt
            $This.TypeKey(65)
            $This.Timer(1)

            # Maximize window
            $This.PressKey(91)
            $This.TypeKey(38)
            $This.ReleaseKey(91)
            $This.Timer(1)

            # Open PowerShell
            $This.TypeText("PowerShell")
            $This.TypeKey(13)
            $This.Timer(1)

            # Wait for PowerShell engine to get ready for input
            $This.Idle(5,5)
        }
        [Void] AddScript([UInt32]$Phase,[String]$Name,[String]$DisplayName,[String[]]$Content)
        {
            $This.Script.Add($Phase,$Name,$DisplayName,$Content)
            $This.Update(0,"[+] Added (Script) : $Name")
        }
        [Object] GetScript([UInt32]$Index)
        {
            $Item = $This.Script.Get($Index)
            If (!$Item)
            {
                $This.Error("[!] Invalid index")
            }

            Return $Item
        }
        [Object] GetScript([String]$Name)
        {
            $Item = $This.Script.Get($Name)
            If (!$Item)
            {
                $This.Error(-1,"[!] Invalid name")
            }

            Return $Item
        }
        [Void] RunScript()
        {
            $Item = $This.Script.Current()

            If ($Item.Complete -eq 1)
            {
                $This.Error(-1,"[!] Exception (Script) : [$($Item.Name)] already completed")
            }

            $This.Update(0,"[~] Running (Script) : [$($Item.Name)]")
            ForEach ($Line in $Item.Content)
            {
                Switch -Regex ($Line)
```

```
            {
                "^\<Pause\[\d+\]\>$"
                {
                    $Line -match "\d+"
                    $This.Timer($Matches[0])
                }
                "^$"
                {
                    $This.Idle(5,2)
                }
                Default
                {
                    $This.TypeText($Line)
                    $This.TypeKey(13)
                }
            }
        }

        $This.Update(1,"[+] Complete (Script) : [$($Item.Name)]")

        $Item.Complete = 1
        $This.Script.Selected ++
    }
    [Object] GetNetworkNode([Object]$File)
    {
        Return [VmNetworkNode]::New($File)
    }
    [String] GetRegistryPath()
    {
        Return "HKLM:\Software\Policies\Secure Digits Plus LLC"
    }
    SetPersistentInfo()
    {
        # [Phase 1] Set persistent information
        $This.Script.Add(1,"SetPersistentInfo","Set persistent information",@(
        '$Root      = "{0}"' -f $This.GetRegistryPath();
        '$Name      = "{0}"' -f $This.Name;
        '$Path      = "$Root\ComputerInfo"';
        'Rename-Computer $Name -Force';
        'If (!(Test-Path $Root))';
        '{';
        '    New-Item -Path $Root -Verbose';
        '}';
        'New-Item -Path $Path -Verbose';
        ForEach ($Prop in @($This.Network.PSObject.Properties))
        {
            $Line = 'Set-ItemProperty -Path $Path -Name {0} -Value {1}'
            Switch ($Prop.Name)
            {
                Default
                {
                    $Line -f $Prop.Name, $Prop.Value;
                }
                Dns
                {
                    $Value = "([String[]]@(`"{0}`"))" -f ($Prop.Value -join "`",`"")
                    $Line -f $Prop.Name, $Value
                }
                Dhcp
                {
                    '$Dhcp = "$Path\Dhcp"'
                    'New-Item $Dhcp'
                    'Set-ItemProperty -Path $Path -Name Dhcp -Value $Dhcp'
                    $Line = 'Set-ItemProperty -Path $Dhcp -Name {0} -Value {1}'
                    ForEach ($Item in $Prop.Value.PSObject.Properties)
                    {
                        If ($Item.Value.Count -eq 1)
                        {
                            $Line -f $Item.Name, $Item.Value
                        }
                        Else
                        {
```

```
                                $Value = "([String[]]@(`"{0}`"))" -f ($Item.Value -join "`",`"")
                                $Line -f $Item.Name, $Value
                            }
                        }
                    }
                }
            }))
        }
        SetTimeZone()
        {
            # [Phase 2] Set time zone
            $This.Script.Add(2,"SetTimeZone","Set time zone",@('Set-Timezone -Name "{0}"' -f (Get-Timezone).Id))
        }
        SetComputerInfo()
        {
            # [Phase 3] Set computer info
            $This.Script.Add(3,"SetComputerInfo","Set computer info",@(
            '$Item           = Get-ItemProperty "{0}\ComputerInfo"' -f $This.GetRegistryPath()
            '$TrustedHost    = $Item.Trusted';
            '$IPAddress      = $Item.IpAddress';
            '$PrefixLength   = $Item.Prefix';
            '$DefaultGateway = $Item.Gateway';
            '$Dns            = $Item.Dns'))
        }
        SetIcmpFirewall()
        {
            # [Phase 4] Enable IcmpV4
            $This.Script.Add(4,"SetIcmpFirewall","Enable IcmpV4",@(
            'Get-NetFirewallRule | ? DisplayName -match "(Printer.+IcmpV4)" | Enable-NetFirewallRule -Verbose'))
        }
        SetInterfaceNull()
        {
            # [Phase 5] Get InterfaceIndex, get/remove current (IP address + Net Route)
            $This.Script.Add(5,"SetInterfaceNull","Get InterfaceIndex, get/remove current (IP address + Net
Route)",@(
            '$Index              = Get-NetAdapter | ? Status -eq Up | % InterfaceIndex';
            '$Interface          = Get-NetIPAddress    -AddressFamily IPv4 -InterfaceIndex $Index';
            '$Interface          | Remove-NetIPAddress -AddressFamily IPv4 -Confirm:$False -Verbose';
            '$Interface          | Remove-NetRoute     -AddressFamily IPv4 -Confirm:$False -Verbose'))
        }
        SetStaticIp()
        {
            # [Phase 6] Set static IP Address
            $This.Script.Add(6,"SetStaticIp","Set (static IP Address + Dns server)",@(
            '$Splat              = @{';
            ' ';
            '    InterfaceIndex  = $Index';
            '    AddressFamily   = "IPv4"';
            '    PrefixLength    = $Item.Prefix';
            '    ValidLifetime   = [Timespan]::MaxValue';
            '    IPAddress       = $Item.IPAddress';
            '    DefaultGateway  = $Item.Gateway';
            '}';
            'New-NetIPAddress @Splat';
            'Set-DnsClientServerAddress -InterfaceIndex $Index -ServerAddresses $Item.Dns'))
        }
        SetWinRm()
        {
            # [Phase 7] Set (WinRM Config/Self-Signed Certificate/HTTPS Listener)
            $This.Script.Add(7,"SetWinRm","Set (WinRM Config/Self-Signed Certificate/HTTPS Listener)",@(
            'winrm quickconfig';
            '<Pause[2]>';
            'y';
            '<Pause[3]>';
            'Set-Item WSMan:\localhost\Client\TrustedHosts -Value $Item.Trusted';
            '<Pause[4]>';
            'y';
            '$Cert       = New-SelfSignedCertificate -DnsName $Item.IpAddress -CertStoreLocation
Cert:\LocalMachine\My';
            '$Thumbprint = $Cert.Thumbprint';
            '$Hash       = "@{Hostname=`"$IPAddress`";CertificateThumbprint=`"$Thumbprint`"}"';
            "`$Str         = `"winrm create winrm/config/Listener?Address=*+Transport=HTTPS '{0}'`"";
```

```
            'Invoke-Expression ($Str -f $Hash)'))
        }
        SetWinRmFirewall()
        {
            # [Phase 8] Set WinRm Firewall
            $This.Script.Add(8,"SetWinRmFirewall",'Set WinRm Firewall',@(
            '$Splat          = @{';
            ' ';
            '    Name        = "WinRM/HTTPS"';
            '    DisplayName = "Windows Remote Management (HTTPS-In)"';
            '    Direction   = "In"';
            '    Action      = "Allow"';
            '    Protocol    = "TCP"';
            '    LocalPort   = 5986';
            '}';
            'New-NetFirewallRule @Splat -Verbose'))
        }
        SetRemoteDesktop()
        {
            # [Phase 9] Set Remote Desktop
            $This.Script.Add(9,"SetRemoteDesktop",'Set Remote Desktop',@(
            'Set-ItemProperty "HKLM:\System\CurrentControlSet\Control\Terminal Server" -Name fDenyTSConnections
-Value 0';
            'Enable-NetFirewallRule -DisplayGroup "Remote Desktop"'))
        }
        InstallFeModule()
        {
            # [Phase 10] Install [FightingEntropy()]
            $This.Script.Add(10,"InstallFeModule","Install [FightingEntropy()]",@(
            '[Net.ServicePointManager]::SecurityProtocol = 3072'
            'Set-ExecutionPolicy Bypass -Scope Process -Force'
            '$Install = "https://github.com/mcc85s/FightingEntropy"'
            '$Full    = "$Install/blob/main/Version/2022.12.0/FightingEntropy.ps1?raw=true"'
            'Invoke-RestMethod $Full | Invoke-Expression'
            '$Module.Install()'
            'Import-Module FightingEntropy'))
        }
        InstallChoco()
        {
            # [Phase 11] Install Chocolatey
            $This.Script.Add(11,"InstallChoco","Install Chocolatey",@(
            "Invoke-RestMethod chocolatey.org/install.ps1 | Invoke-Expression"))
        }
        InstallVsCode()
        {
            # [Phase 12] Install Visual Studio Code
            $This.Script.Add(12,"InstallVsCode","Install Visual Studio Code",@("choco install vscode -y"))
        }
        InstallBossMode()
        {
            # [Phase 13] Install BossMode (vscode color theme)
            $This.Script.Add(13,"InstallBossMode","Install BossMode (vscode color theme)",@("Install-BossMode"))
        }
        InstallPsExtension()
        {
            # [Phase 14] Install Visual Studio Code (PowerShell Extension)
            $This.Script.Add(14,"InstallPsExtension","Install Visual Studio Code (PowerShell Extension)",@(
            '$FilePath     = "$Env:ProgramFiles\Microsoft VS Code\bin\code.cmd"';
            '$ArgumentList = "--install-extension ms-vscode.PowerShell"';
            'Start-Process -FilePath $FilePath -ArgumentList $ArgumentList -NoNewWindow | Wait-Process'))
        }
        RestartComputer()
        {
            # [Phase 15] Restart computer
            $This.Script.Add(15,'Restart','Restart computer',@('Restart-Computer'))
        }
        ConfigureDhcp()
        {
            # [Phase 16] Configure Dhcp
            $This.Script.Add(16,'ConfigureDhcp','Configure Dhcp',@(
            '$Root           = "{0}"' -f $This.GetRegistryPath()
            '$Path           = "$Root\ComputerInfo"'
```

```
            '$Item           = Get-ItemProperty $Path'
            '$Item.Dhcp      = Get-ItemProperty $Item.Dhcp';
            ' ';
            '$Splat = @{ ';
            '    ';
            '    StartRange = $Item.Dhcp.StartRange';
            '    EndRange   = $Item.Dhcp.EndRange';
            '    Name       = $Item.Dhcp.Name';
            '    SubnetMask = $Item.Dhcp.SubnetMask';
            '}';
            '';
            'Add-DhcpServerV4Scope @Splat -Verbose';
            'Add-DhcpServerInDc -Verbose';
            ' ';
            'ForEach ($Value in $Item.Dhcp.Exclusion)';
            '{';
            '    $Splat         = @{ ';
            ' ';
            '        ScopeId    = $Item.Dhcp.Network';
            '        StartRange = $Value';
            '        EndRange   = $Value';
            '    }';
            ' ';
            '    Add-DhcpServerV4ExclusionRange @Splat -Verbose';
            ' ';
            '   (3,$Item.Gateway),';
            '   (6,$Item.Dns),';
            '   (15,$Item.Domain),';
            '   (28,$Item.Dhcp.Broadcast) | % {';
            '    ';
            '      Set-DhcpServerV4OptionValue -OptionId $_[0] -Value $_[1] -Verbose'
            '   }';
            '}';
            'netsh dhcp add securitygroups';
            'Restart-Service dhcpserver';
            ' ';
            '$Splat    = @{ ';
            ' ';
            '    Path  = "HKLM:\SOFTWARE\Microsoft\ServerManager\Roles\12"';
            '    Name  = "ConfigurationState"';
            '    Value = 2';
            '}';
            ' ';
            'Set-ItemProperty @Splat -Verbose'))
        }
        InitializeFeAd([String]$Pass)
        {
            $This.Script.Add(17,'InitializeAd','Initialize [FightingEntropy()] AdInstance',@(
            '$Password = Read-Host "Enter password" -AsSecureString';
            '<Pause[2]>';
            '{0}' -f $Pass;
            '$Ctrl = Initialize-FeAdInstance';
            ' ';
            '# Set location';
            '$Ctrl.SetLocation("1718 US-9","Clifton Park","NY",12065,"US")';
            ' ';
            '# Add Organizational Unit';
            '$Ctrl.AddAdOrganizationalUnit("DevOps","Developer(s)/Operator(s)")';
            ' ';
            '# Get Organizational Unit';
            '$Ou     = $Ctrl.GetAdOrganizationalUnit("DevOps")';
            ' ';
            '# Add Group';
            '$Ctrl.AddAdGroup("Engineering","Security","Global","Secure Digits Plus LLC",
$Ou.DistinguishedName)';
            ' ';
            '# Get Group';
            '$Group  = $Ctrl.GetAdGroup("Engineering")';
            ' ';
            '# Add-AdPrincipalGroupMembership';
            '$Ctrl.AddAdPrincipalGroupMembership($Group.Name,@("Administrators","Domain Admins"))';
            ' ';
```

```
            '# Add User';
            '$Ctrl.AddAdUser("Michael","C","Cook","mcook85",$Ou.DistinguishedName)';
            ' ';
            '# Get User';
            '$User   = $Ctrl.GetAdUser("Michael","C","Cook")';
            ' ';
            '# Set [User.General (Description, Office, Email, Homepage)]';
            '$User.SetGeneral("Beginning the fight against ID theft and cybercrime",';
            '                 "<Unspecified>",';
            '                 "michael.c.cook.85@gmail.com",';
            '                 "https://github.com/mcc85s/FightingEntropy")';
            ' ';
            '# Set [User.Address (StreetAddress, City, State, PostalCode, Country)] ';
            '$User.SetLocation($Ctrl.Location)';
            ' ';
            '# Set [User.Profile (ProfilePath, ScriptPath, HomeDirectory, HomeDrive)]';
            '$User.SetProfile("","","","")';
            ' ';
            '# Set [User.Telephone (HomePhone, OfficePhone, MobilePhone, Fax)]';
            '$User.SetTelephone("","518-406-8569","518-406-8569","")';
            ' ';
            '# Set [User.Organization (Title, Department, Company)]';
            '$User.SetOrganization("CEO/Security Engineer","Engineering","Secure Digits Plus LLC")';
            ' ';
            '# Set [User.AccountPassword]';
            '$User.SetAccountPassword($Password)';
            ' ';
            '# Add user to group';
            '$Ctrl.AddAdGroupMember($Group,$User)';
            ' ';
            '# Set user primary group';
            '$User.SetPrimaryGroup($Group)'))
        }
        Load()
        {
            $This.SetPersistentInfo()
            $This.SetTimeZone()
            $This.SetComputerInfo()
            $This.SetIcmpFirewall()
            $This.SetInterfaceNull()
            $This.SetStaticIp()
            $This.SetWinRm()
            $This.SetWinRmFirewall()
            $This.SetRemoteDesktop()
            $This.InstallFeModule()
            $This.InstallChoco()
            $This.InstallVsCode()
            $This.InstallBossMode()
            $This.InstallPsExtension()
            $This.RestartComputer()
            $This.ConfigureDhcp()
        }
        [String] ProgramData()
        {
            Return [Environment]::GetEnvironmentVariable("ProgramData")
        }
        [String] Author()
        {
            Return "Secure Digits Plus LLC"
        }
        [Object] Now()
        {
            Return [DateTime]::Now.ToString("yyyy-MMdd_HHmmss")
        }
        [String] LogPath()
        {
            $xPath = $This.ProgramData()

            ForEach ($Folder in $This.Author(), "Logs")
            {
                $xPath = $xPath, $Folder -join "\"
                If (![System.IO.Directory]::Exists($xPath))
```

```
                {
                    [System.IO.Directory]::CreateDirectory($xPath)
                }
            }

            Return $xPath
        }
        [Object] PSSession([Object]$Admin)
        {
            # Attempt login
            $This.Update(0,"[~] PSSession")
            $Splat = @{

                ComputerName  = $This.Network.IpAddress
                Port          = 5986
                Credential    = $Admin.Credential
                SessionOption = New-PSSessionOption -SkipCACheck
                UseSSL        = $True
            }

            Return $Splat
        }
        DumpConsole()
        {
            $xPath = "{0}\{1}-{2}.log" -f $This.LogPath(), $This.Now(), $This.Name
            $This.Update(100,"[+] Dumping console: [$xPath]")
            $This.Console.Finalize()

            $Value = $This.Console.Output | % ToString

            [System.IO.File]::WriteAllLines($xPath,$Value)
        }
        [String] ToString()
        {
            Return $This.Name
        }
    }
```

Class [VmController] /----------\
/----------

```
    # // ================================================
    # // | Class meant to control all of the above classes |
    # // ================================================

    Class VmController
    {
        [String]     $Path
        [String]   $Domain
        [String]  $NetBios
        [Object]     $Node
        [Object]    $Admin
        [UInt32] $Selected
        [Object[]]   $File
        VmController([String]$Path,[String]$Domain,[String]$NetBios)
        {
            $This.Path    = $Path
            $This.Domain  = $Domain
            $This.NetBios = $NetBios
            $This.File    = @( )
        }
        [Object] VmNodeController()
        {
            Return [VmNodeController]::New($This.Path,$This.Domain,$This.NetBios)
        }
        [Object] VmNodeInputObject([Object]$Token)
        {
            Return [VmNodeInputObject]::New($Token,$This.Path)
        }
        [Object] VmAdminCredential()
        {
            $Item = Get-ChildItem $This.Path | ? Name -eq admin.txt
            If (!$Item)
            {
                Throw "Admin file not found"
            }

            Return [VmAdminCredential]::New($Item)
        }
        Select([UInt32]$Index)
        {
            If ($Index -gt $This.File.Count)
            {
                Throw "Index is too large"
            }

            $This.Selected = $Index
        }
        [Object] Current()
        {
            Return $This.File[$This.Selected]
        }
        [Object] VmObject()
        {
            Return [VmObject]::New($This.Current().Object)
        }
        [Object] VmObject([Switch]$Flags,[Object]$Item)
        {
            Return [VmObject]::New([Switch]$True,$Item)
        }
        GetNodeController()
        {
            $This.Node    = $This.VmNodeController()
        }
        GetNodeInputObject([String]$Token)
        {
            If ($Token -notin (Get-ChildItem $This.Path).Name)
            {
                Throw "Invalid file"
```

```
            }

            $This.File    += $This.VmNodeInputObject($Token)
        }
        GetNodeAdminCredential()
        {
            $This.Admin    = $This.VmAdminCredential()
        }
        Prime()
        {
            $Item          = Get-VM -Name $This.Current().Object.Name -EA 0
            If ($Item)
            {
                $Vm        = $This.VmObject([Switch]$True,$Item)
                $Vm.Update(1,"[_] Removing $($Vm.Name)")
                ForEach ($Property in $Vm.PSObject.Properties)
                {
                    $Line = "[_] {0} : {1}" -f $Property.Name.PadRight(10," "), $Property.Value
                    $Vm.Update(1,$Line)
                }
                $Vm.Remove()
            }
        }
        [String] ToString()
        {
            Return "<FEVirtual.VmController>"
        }
    }
```

_____/ Class [VmController]

```
  Script /----------------------------------------------------------------------------------------------------------------------\
/--------
```

Since this is only a SCRIPT right now, and not a full-blown function...?
I'm going to cover the various aspects of the above work, by showing the customized script process below.

```
#    ____    ____________________________________________________________________________________________________
#   //¯¯\\__//¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\\___
#   \\__//¯¯¯ Creation Area [~]                                                                             ___//¯¯\\
#    ¯¯¯\\__________________________________________________________________________________________________//¯¯\\__//
#        ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯    ¯¯¯¯

    # // ==================================================================================
    # // | Spawn up a hive controller to create file system objects for each Hyper-V host |
    # // | to automate all of the stuff each VM node will be reproducing                  |
    # // ==================================================================================

    # // Generates the factory class
    $Hive = [VmController]::New("C:\FileVm","securedigitsplus.com","SECURED")
    $Hive.GetNodeController()

    $Hive.Node.SetTemplate("C:\VDI",2048MB,64GB,2,2,"External",
                           "C:\Images\Windows_Server_2016_Datacenter_EVAL_en-us_14393_refresh.ISO")

    # // Populates the factory class with (3) nodes (0, 1, 2)
    0..2 | % { $Hive.Node.AddNode("server0$_") }

    # // Exports the file system objects
    $Hive.Node.Export()
    $Hive.Node.WriteAdmin()

#    ____    ____________________________________________________________________________________________________
#   //¯¯\\__//¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\\___
#   \\__//¯¯¯ Action Area [~]                                                                               ___//¯¯\\
#    ¯¯¯\\__________________________________________________________________________________________________//¯¯\\__//
#        ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯    ¯¯¯¯

    # // Reinstantiates the file system information
    $Token       = Switch ([Environment]::MachineName)
    {
        l420-x64    { "server00.txt" }
        lobby-comp1 { "server01.txt" }
        lobby-comp2 { "server02.txt" }
    }

    $Hive.GetNodeAdminCredential()
    $Hive.GetNodeInputObject($Token)

    # // Checks for existence of virtual machine by that name
    $Hive.Prime()

    # // Object instantiation
    $Vm          = $Hive.VmObject()
    $Vm.New()
    $Vm.AddVmDvdDrive()
    $Vm.LoadIso($Vm.Iso)
    $Vm.SetIsoBoot()
    $Vm.Connect()

    # // Start Machine
    $Vm.Start()
    $Vm.Control  = $Vm.Wmi("Msvm_ComputerSystem") | ? ElementName -eq $Vm.Name
    $Vm.Keyboard = $Vm.Wmi("Msvm_Keyboard") | ? Path -match $Vm.Control.Name

    # // Wait for "Press enter to boot from CD/DVD", then press enter
    $Vm.Timer(2)
    $Vm.TypeKey(13)

    # // Wait for "Install Windows" menu
    $Vm.Idle(5,2)

    # // Enter menu
    $Vm.TypeKey(13)
    $Vm.Timer(5)
    $Vm.TypeKey(13)
```

```
# // Wait to select installation
$Vm.Idle(5,5)

# // Select installation
$Vm.TypeChain(@(40,40,40,13))

# // Wait to accept license terms
$Vm.Idle(5,2)

# // Accept license terms
$Vm.TypeChain(@(32,9,9,9,9,13))

# // Wait Windows Setup
$Vm.Idle(5,2)

# // Windows Setup
$Vm.SpecialKey(67)

# // Wait partition
$Vm.Idle(5,2)

# // Set partition
$Vm.SpecialKey(78)

# // Wait until Windows installation completes
$Vm.Idle(5,5)

# // Catch and release ISO upon reboot
$Vm.Uptime(0,5)
$Vm.UnloadIso()

# // Wait for the login screen
$Vm.Idle(5,5)

# // Establish administrator account
$Vm.SetAdmin($Hive.Admin)

# Wait for actual login
$Vm.Uptime(1,60)
$Vm.Idle(20,5)

# Enter (CTRL + ALT + DEL) to sign into Windows
$Vm.Login($Hive.Admin)

# Wait for operating system to do [FirstRun/FirstLogin] stuff
$Vm.Timer(30)
$Vm.Idle(5,5)

# Press enter for Network to allow pc to be discoverable
$Vm.TypeKey(13)

# Launch PowerShell
$Vm.LaunchPs()

# Loads all scripts
$Vm.Load()

# Set persistent info
$Vm.RunScript()
$Vm.Timer(5)

# Set time zone
$Vm.RunScript()
$Vm.Timer(1)

# Set computer info
$Vm.RunScript()
$Vm.Timer(3)

# Set Icmp Firewall
$Vm.RunScript()
$Vm.Timer(5)

# Set network interface to null
$Vm.RunScript()
$Vm.Timer(5)

# Set static IP
```

```
    $Vm.RunScript()
    $Vm.Connection()

    # Set WinRm
    $Vm.RunScript()
    $Vm.Timer(5)

    # Set WinRmFirewall
    $Vm.RunScript()
    $Vm.Timer(5)

    # Set Remote Desktop
    $Vm.RunScript()
    $Vm.Timer(5)

    # Install FightingEntropy
    $Vm.RunScript()
    $Vm.Idle(0,5)

    # Install Chocolatey
    $Vm.RunScript()
    $Vm.Idle(0,5)

    # Install VsCode | (Timer + Idle) Network Metering needed here...
    $Vm.RunScript()
    $Vm.Idle(0,5)

    # Install BossMode
    $Vm.RunScript()
    $Vm.Idle(0,5)

    # Install PsExtension
    $Vm.RunScript()
    $Vm.Idle(0,5)

    # Restart computer
    $Vm.RunScript()
    $Vm.Uptime(0,5)
    $Vm.Uptime(1,40)
    $Vm.Idle(5,5)

#     ____    ____________________________________________________________________________________________________
#    //--\\__//                                                                                            \\___
#    \\__//--- Launch PowerShell and Get-FEDCPromo -Mode [~]                                                ___//--\\
#     ---\\_____________________________________________________________________________________________//--\\__//
#         ------------------------------------------------------------------------------------------    ----

    # Login
    $Vm.Login($Hive.Admin)

    # Wait idle
    $Vm.Idle(5,5)

    # Launch PowerShell
    $Vm.LaunchPs()

    # Kill Server Manager
    $Vm.TypeText("Get-Process -Name ServerManager -EA 0 | Stop-Process -Force -Confirm:`$False")
    $Vm.TypeKey(13)

    # Launch FEDCPromo
    $Vm.TypeText("Get-FEDCPromo -Mode 1")
    $Vm.TypeKey(13)

    # Wait idle
    $Vm.Idle(5,5)

    # Switches to FEDCPromo GUI
    $Vm.PressKey(18)
    $Vm.TypeKey(9)
    $Vm.ReleaseKey(18)
    $Vm.Timer(1)

    # Command Tab | [Forest] Already selected
    $Vm.TypeKey(9)
    $Vm.Timer(1)

    # Cycle to tab control, tab over to Names, tab into Domain name
```

```
    $Vm.TypeChain(@(9,9,9,39,39,39,9))
    $Vm.Timer(1)

    # Type domain name
    $Vm.TypeText($Vm.Network.Domain)
    $Vm.Timer(1)

    # Tab into Netbios
    $Vm.TypeKey(9)

    # Type NetBIOS name
    $Vm.TypeText($Vm.Network.NetBios)
    $Vm.Timer(1)

    # Back up to tab control
    $Vm.PressKey(16)
    $Vm.TypeKey(9)
    $Vm.TypeKey(9)
    $Vm.ReleaseKey(16)
    $Vm.Timer(1)

    # Cycle over to (Connection/Dsrm) tab
    $Vm.TypeKey(39)
    $Vm.TypeKey(39)
    $Vm.Timer(1)

    # (Tab into/type) Dsrm password
    $Vm.TypeKey(9)
    $Vm.TypePassword($Hive.Admin.Password())
    $Vm.Timer(1)

    # (Tab into/type) Confirm password
    $Vm.TypeKey(9)
    $Vm.TypePassword($Hive.Admin.Password())
    $Vm.Timer(1)

    # (Tab into/press) Start
    $Vm.TypeKey(9)
    $Vm.TypeKey(13)

#    ____    ____________________________________________________________________________________________________
#   //¯¯\\__//¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\\___
#   \\__//¯¯¯ Installing [~] [FightingEntropy()] Domain Controller                                        ___//¯¯\\
#    ¯¯¯\\__________________________________________________________________________________________________//¯¯\\__//
#        ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯    ¯¯¯¯
#

    # Wait for (reboot/idle)
    $Vm.Uptime(0,5)
    $Vm.Idle(5,10)

    # Login
    $Vm.Login($Hive.Admin)

    # Wait for reboot
    $Vm.Uptime(0,5)

    # Wait for Active Directory to finish installing
    $Vm.Uptime(1,60)
    $Vm.Idle(1,10)

    # Login
    $Vm.Login($Hive.Admin)

    # Wait idle
    $Vm.Idle(5,5)

    # Launch PowerShell, then configure Dhcp
    $Vm.LaunchPs()

    # Configure Dhcp
    $Vm.RunScript()
    $Vm.Idle(5,5)

    # Configure Dns (what's left)...
    # - Sign the zones

    # Populate Active Directory with (Organizational Unit, Group, User)
    $Vm.InitializeFeAd($Hive.Admin.Password())
```

```
    $Vm.RunScript()
    $Vm.Idle(0,5)

    # Configure Wds
    # Install Microsoft Deployment Toolkit/PowerShell Deployment
    # Obtain Iso
    # Generate an environment key

#    ____    ____________________________________________________________________________________________________
#   //¯¯\\__//¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\\___
#   \\__//¯¯¯ Development [~] Launch VSCode and set initial settings                                 ___//¯¯\\
#    ¯¯¯\\__________________________________________________________________________________________//¯¯\\__//
#        ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯    ¯¯¯¯

    <# Start menu
    $Vm.TypeChain(@(91,40,40,93,40,39,40,40,13))
    $Vm.Idle(5,5)

    # Color Theme ->
    $Vm.PressKey(17)
    $Vm.TypeKey(75)
    $Vm.Timer(1)
    $Vm.TypeKey(84)
    $Vm.ReleaseKey(17)
    $Vm.TypeText("BossMode")
    $Vm.TypeKey(13)
    $Vm.Timer(1)

    # [F]ile ->
    $Vm.SpecialKey(70)
    $Vm.Timer(1)

    ##########################>
```

```
                                                                                                    ________/
\__________________________________________________________________________________________________/ Script
  Output /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯
```

This first section here instantiates the main class, [VmController].
The parameters are feeding a local location for the files to go, and that location has already been configured with Samba file share properties which have been propagated to additional systems running Hyper-V.

```
#    ____    ____________________________________________________________________________________________________
#   //¯¯\\__//¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\\___
#   \\__//¯¯¯ Creation Area [~]                                                                     ___//¯¯\\
#    ¯¯¯\\__________________________________________________________________________________________//¯¯\\__//
#        ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯    ¯¯¯¯

    # // =================================================================================
    # // | Spawn up a hive controller to create file system objects for each Hyper-V host |
    # // | to automate all of the stuff each VM node will be reproducing                  |
    # // =================================================================================

    # // Generates the factory class
    $Hive = [VmController]::New("C:\FileVm","securedigitsplus.com","SECURED")
    $Hive.GetNodeController()

    $Hive.Node.SetTemplate("C:\VDI",2048MB,64GB,2,2,"External",
                           "C:\Images\Windows_Server_2016_Datacenter_EVAL_en-us_14393_refresh.ISO")

    # // Populates the factory class with (3) nodes (0, 1, 2)
    0..2 | % { $Hive.Node.AddNode("server0$_") }

    # // Exports the file system objects
    $Hive.Node.Export()
    $Hive.Node.WriteAdmin()
```

```
PS Prompt:\> $Hive = [VmController]::New("C:\FileVm","securedigitsplus.com","SECURED")
PS Prompt:\> $Hive

Path     : C:\FileVm
Domain   : securedigitsplus.com
```

```
NetBios  : SECURED
Node     :
Admin    :
Selected : 0
File     : {}

PS Prompt:\>
```

So, in the above output we can see that $Hive is capturing the class [VmController], whereby allowing the above information to be readily available.

The properties “Node” and “Admin” are <empty>/$Null.

We can populate that information by doing the following...

```
$Hive.GetNodeController()
```

```
PS Prompt:\> $Hive.GetNodeController()
PS Prompt:\> $Hive

Path     : C:\FileVm
Domain   : securedigitsplus.com
NetBios  : SECURED
Node     : <FEVirtual.VmNode[Controller]>
Admin    :
Selected : 0
File     : {}

PS Prompt:\> $Hive.Node

Path     : C:\FileVm
Domain   : securedigitsplus.com
NetBios  : SECURED
Admin    : <FEVirtual.VmAdminCredential>
Config   : NetIPConfiguration
Network  : <FEVirtual.VmNetwork[Controller]>
Template :

PS Prompt:\>
```

So, right away, we can see that using that command populates the property for Node, with the class from up above, named “VmNodeController”. The actual string information says “<FEVirtual.VmNode[Controller]>”, but...
...that is just a .ToString() representation of the actual object behind the console output.

By accessing the property $Hive.Node, we are able to see that there's a LOT more information there, now.

```
PS Prompt:\> $Hive.Node.Admin

UserName      Credential
--------      ----------
Administrator System.Management.Automation.PSCredential

PS Prompt:\> $Hive.Node.Config

ComputerName                            : L420-X64
InterfaceAlias                          : vEthernet (External)
InterfaceIndex                          : 26
InterfaceDescription                    : Hyper-V Virtual Ethernet Adapter #2
NetCompartment.CompartmentId            : 1
NetCompartment.CompartmentDescription   : Default Compartment
NetAdapter.LinkLayerAddress             : DE-56-7D-78-94-3A
NetAdapter.Status                       : Up
NetProfile.Name                         : Network 177
NetProfile.NetworkCategory              : Private
NetProfile.IPv6Connectivity             : NoTraffic
NetProfile.IPv4Connectivity             : Internet
IPv6LinkLocalAddress                    : fe80::bf96:b672:7015:7147%26
IPv4Address                             : 192.168.42.2
IPv6DefaultGateway                      :
IPv4DefaultGateway                      : 192.168.42.129
NetIPv6Interface.NlMTU                  : 1500
```

```
NetIPv4Interface.NlMTU               : 1500
NetIPv6Interface.DHCP                : Enabled
NetIPv4Interface.DHCP                : Enabled
DNSServer                            : 192.168.42.129

PS Prompt:\> $Hive.Node.Network

Domain   : securedigitsplus.com
NetBios  : SECURED
Trusted  : 192.168.42.2
Prefix   : 24
Netmask  : 255.255.255.0
Wildcard : 1.1.1.256
Gateway  : 192.168.42.129
Dns      : {192.168.42.129}
Dhcp     : <FEVirtual.VmNetwork[Dhcp]>
Networks : {<FEVirtual.VmNetwork[List]>}
Hosts    : {192.168.42.0, 192.168.42.1, 192.168.42.2, 192.168.42.3...}
Nodes    : {}

PS Prompt:\>
```

Looks like the “Template” field is unpopulated, however. Let's populate it.

```
$Hive.Node.SetTemplate("C:\VDI",2048MB,64GB,2,2,"External",
"C:\Images\Windows_Server_2016_Datacenter_EVAL_en-us_14393_refresh.ISO")

# // Populates the factory class with (3) nodes (0, 1, 2)
0..2 | % { $Hive.Node.AddNode("server0$_") }
```

```
PS Prompt:\> $Hive.Node.SetTemplate("C:\VDI",2048MB,64GB,2,2,"External",
>>  "C:\Images\Windows_Server_2016_Datacenter_EVAL_en-us_14393_refresh.ISO")
>>  # // Populates the factory class with (3) nodes (0, 1, 2)
>>  0..2 | % { $Hive.Node.AddNode("server0$_") }

[+] Node [server00] added
[+] Node [server01] added
[+] Node [server02] added
PS Prompt:\>
```

Then, we can see the nodes in the property `$Hive.Node.Network.Nodes`

```
PS Prompt:\> $Hive.Node.Network

Domain   : securedigitsplus.com
NetBios  : SECURED
Trusted  : 192.168.42.2
Prefix   : 24
Netmask  : 255.255.255.0
Wildcard : 1.1.1.256
Gateway  : 192.168.42.129
Dns      : {192.168.42.129}
Dhcp     : <FEVirtual.VmNetwork[Dhcp]>
Networks : {<FEVirtual.VmNetwork[List]>}
Hosts    : {192.168.42.0, 192.168.42.1, 192.168.42.2, 192.168.42.3...}
Nodes    : {<FEVirtual.VmNetwork[Node]>, <FEVirtual.VmNetwork[Node]>, <FEVirtual.VmNetwork[Node]>}

PS Prompt:\> $Hive.Node.Network.Nodes | Format-Table

Index Name     IpAddress    Domain               NetBios Trusted      Prefix Netmask       Gateway        Dns
----- ----     ---------    ------               ------- -------      ------ -------       -------        ---
    0 server00 192.168.42.1 securedigitsplus.com SECURED 192.168.42.2     24 255.255.255.0 192.168.42.129 ...
    1 server01 192.168.42.3 securedigitsplus.com SECURED 192.168.42.2     24 255.255.255.0 192.168.42.129 ...
    2 server02 192.168.42.4 securedigitsplus.com SECURED 192.168.42.2     24 255.255.255.0 192.168.42.129 ...

PS Prompt:\>
```

And NOW, we can use the following methods to export the nodes to files.

```
    # // Exports the file system objects
    $Hive.Node.Export()
    $Hive.Node.WriteAdmin()
```

```
PS Prompt:\> # // Exports the file system objects
>> $Hive.Node.Export()
>> $Hive.Node.WriteAdmin()
Exported  [+] File: [C:\FileVm\server00.txt]
Exported  [+] File: [C:\FileVm\server01.txt]
Exported  [+] File: [C:\FileVm\server02.txt]
Exported  [+] File: [C:\FileVm\admin.txt]
PS Prompt:\>
```

The second area here, is the action area.

```
#    ____    ______________________________________________________________________________________________
#   //¯¯\\__//¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\\___
#   \\__//¯¯¯ Action Area [~]                                                                       ___//¯¯\\
#    ¯¯¯\\__________________________________________________________________________________________//¯¯\\__//
#        ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯    ¯¯¯¯

    # // Reinstantiates the file system information
    $Token       = Switch ([Environment]::MachineName)
    {
        l420-x64    { "server00.txt" }
        lobby-comp1 { "server01.txt" }
        lobby-comp2 { "server02.txt" }
    }

    $Hive.GetNodeAdminCredential()
    $Hive.GetNodeInputObject($Token)

    # // Checks for existence of virtual machine by that name
    $Hive.Prime()

    # // Object instantiation
    $Vm          = $Hive.VmObject()
    $Vm.New()
    $Vm.AddVmDvdDrive()
    $Vm.LoadIso($Vm.Iso)
    $Vm.SetIsoBoot()
    $Vm.Connect()

    # // Start Machine
    $Vm.Start()
    $Vm.Control  = $Vm.Wmi("Msvm_ComputerSystem") | ? ElementName -eq $Vm.Name
    $Vm.Keyboard = $Vm.Wmi("Msvm_Keyboard") | ? Path -match $Vm.Control.Name

    # // Wait for "Press enter to boot from CD/DVD", then press enter
    $Vm.Timer(2)
    $Vm.TypeKey(13)

    # // Wait for "Install Windows" menu
    $Vm.Idle(5,2)

    # // Enter menu
    $Vm.TypeKey(13)
    $Vm.Timer(5)
    $Vm.TypeKey(13)

    # // Wait to select installation
    $Vm.Idle(5,5)

    # // Select installation
    $Vm.TypeChain(@(40,40,40,13))

    # // Wait to accept license terms
    $Vm.Idle(5,2)

    # // Accept license terms
```

```
    $Vm.TypeChain(@(32,9,9,9,9,13))

    # // Wait Windows Setup
    $Vm.Idle(5,2)

    # // Windows Setup
    $Vm.SpecialKey(67)

    # // Wait partition
    $Vm.Idle(5,2)

    # // Set partition
    $Vm.SpecialKey(78)

    # // Wait until Windows installation completes
    $Vm.Idle(5,5)

    # // Catch and release ISO upon reboot
    $Vm.Uptime(0,5)
    $Vm.UnloadIso()

    # // Wait for the login screen
    $Vm.Idle(5,5)
```

Suppose that I just run the first few lines there, so that I could scope out the I/O of the object being created there...?

```
    # // Reinstantiates the file system information
    $Token       = Switch ([Environment]::MachineName)
    {
        l420-x64    { "server00.txt" }
        lobby-comp1 { "server01.txt" }
        lobby-comp2 { "server02.txt" }
    }

    $Hive.GetNodeAdminCredential()
    $Hive.GetNodeInputObject($Token)
```

```
PS Prompt:\> # // Reinstantiates the file system information
>>    $Token       = Switch ([Environment]::MachineName)
>>    {
>>        l420-x64    { "server00.txt" }
>>        lobby-comp1 { "server01.txt" }
>>        lobby-comp2 { "server02.txt" }
>>    }
>>
>>    $Hive.GetNodeAdminCredential()
>>    $Hive.GetNodeInputObject($Token)
PS Prompt:\> $Hive

Path     : C:\FileVm
Domain   : securedigitsplus.com
NetBios  : SECURED
Node     : <FEVirtual.VmNode[Controller]>
Admin    : <FEVirtual.VmAdminCredential>
Selected : 0
File     : {<FEVirtual.VmNode[InputObject]>}

PS Prompt:\> $Hive.File

Path      Object
----      ------
C:\FileVm @{Index=0; Name=server00; IpAddress=192.168.42.1; Domain=securedigitsplus.com; NetBios=SECURED...

PS Prompt:\> $Hive.File.Object

Index     : 0
Name      : server00
IpAddress : 192.168.42.1
```

```
Domain    : securedigitsplus.com
NetBios   : SECURED
Trusted   : 192.168.42.2
Prefix    : 24
Netmask   : 255.255.255.0
Gateway   : 192.168.42.129
Dns       : {192.168.42.129}
Dhcp      : @{Name=192.168.42.0/24; SubnetMask=255.255.255.0; Network=192.168.42.0; StartRange=192.168.42.1;
            EndRange=192.168.42.254; Broadcast=192.168.42.255; Exclusion=System.Object[]}
Base      : C:\VDI
Memory    : 2147483648
Hdd       : 68719476736
Gen       : 2
Core      : 2
SwitchId  : External
Image     : C:\Images\Windows_Server_2016_Datacenter_EVAL_en-us_14393_refresh.ISO

PS Prompt:\> $Hive.File.Admin

UserName                        Password
--------                        --------
Administrator System.Security.SecureString

PS Prompt:\>
```

Alright, so now we're cooking with gas.
(Don't cook with gas anymore, it's bad for your health. https://youtu.be/qzq0RbkHV78)

You'll likely want to have Hyper-V manager open for this portion from here forward, because the remaining variables and properties are going to directly access the Hyper-V portion of the operating system.

I'm probably at some point, going to make a number of scripts that (enable/install) Hyper-V on particular systems, however- Hyper-V actually requires some hardware features like Intel x-VT or whatever, in order for the embedded virtualization hypervisor to work.

I'm not going to get into how to set up Hyper-V in this document, though to be clear, it is probably a good idea to capitalize on how to install it. There are some scripts out there that people can use or follow, in order to deploy or install it.

```
    # // Checks for existence of virtual machine by that name
    $Hive.Prime()

    # // Object instantiation
    $Vm          = $Hive.VmObject()
```

```
PS Prompt:\>  # // Checks for existence of virtual machine by that name
>> $Hive.Prime()
[00:00:00] (State: 0/Status: Running [~] (1/25/2023 3:52:59 PM))
[00:00:01.0965469] (State: 0/Status: [~] Getting VmFirmware : server00)
[00:00:01.1722192] (State: 1/Status: [_] Removing server00)
[00:00:01.3152153] (State: 1/Status: [_] Console    : 00:00:01.3122148)
[00:00:01.3162137] (State: 1/Status: [_] Name       : server00)
[00:00:01.3202146] (State: 1/Status: [_] Memory     : 2.00 GB)
[00:00:01.3222141] (State: 1/Status: [_] Path       : C:\VDI\server00)
[00:00:01.3242182] (State: 1/Status: [_] Vhd        : C:\VDI\server00\server00_1CBE162B-5516-42F6-A1A1-
FC9676ECDE4D.avhdx)
[00:00:01.3272165] (State: 1/Status: [_] VhdSize    : 64.00 GB)
[00:00:01.3292154] (State: 1/Status: [_] Generation : 2)
[00:00:01.3312149] (State: 1/Status: [_] Core       : 2)
[00:00:01.3332147] (State: 1/Status: [_] Switch     : System.Object[])
[00:00:01.3352154] (State: 1/Status: [_] Firmware   : VMFirmware (VMName = 'server00') [VMId = '6cd62a1c-8254-
49dc-927b-42ed1291c822'])
[00:00:01.3372145] (State: 1/Status: [_] Exists     : 1)
[00:00:01.3392159] (State: 1/Status: [_] Guid       : 6cd62a1c-8254-49dc-927b-42ed1291c822)
[00:00:01.3412147] (State: 1/Status: [_] Network    : )
[00:00:01.3442158] (State: 1/Status: [_] Iso        : )
[00:00:01.3482168] (State: 1/Status: [_] Script     : )
[00:00:01.3812135] (State: 0/Status: [~] Removing : server00)
[00:00:01.3992215] (State: 0/Status: [~] State : server00 [attempting shutdown])
[00:00:01.4282159] (State: 1/Status: [~] Starting : server00)
```

```
[00:00:09.3798506] (State: 0/Status: [~] Stopping : server00)
[00:01:30.5321587] (State: 0/Status: [~] Vhd  : [C:\VDI\server00\server00_1CBE162B-5516-42F6-A1A1-
FC9676ECDE4D.avhdx])
[00:01:30.5491568] (State: 0/Status: [~] Path : [C:\VDI\server00])
[00:01:30.5791570] (State: 0/Status: [~] C:\VDI\server00\server00\Virtual Machines)
[00:01:30.5851549] (State: 0/Status: [~] C:\VDI\server00\server00\Snapshots)
[00:01:30.5921558] (State: 0/Status: [~] C:\VDI\server00\server00.vhdx)
[00:01:30.5961550] (State: 0/Status: [~] C:\VDI\server00\server00)
[00:01:30.5991551] (State: 1/Status: [ ] Removed : C:\VDI\server00\server00)
[00:01:30.6531570] (State: 100/Status: [+] Dumping console: [C:\ProgramData\Secure Digits Plus LLC\Logs\2023-
0125_155430-server00.log])
[00:00:00.0009987] (State: 0/Status: Running [~] (1/25/2023 3:54:30 PM))
PS Prompt:\> # // Object instantiation
>>    $Vm          = $Hive.VmObject()
[00:00:00] (State: 0/Status: Running [~] (1/25/2023 3:55:05 PM))
PS Prompt:\> $Vm

Console    : 00:00:54.8491136
Name       : server00
Memory     : 2.00 GB
Path       : C:\VDI\server00
Vhd        : C:\VDI\server00\server00.vhdx
VhdSize    : 64.00 GB
Generation : 2
Core       : 2
Switch     : {External}
Firmware   :
Exists     : 0
Guid       :
Network    : <FEVirtual.VmNetwork[Node]>
Iso        : C:\Images\Windows_Server_2016_Datacenter_EVAL_en-us_14393_refresh.ISO
Script     :

PS Prompt:\>
```

At this point, the console is running (if you don't have the [FightingEntropy()] module installed, it won't actually proceed past any of the steps where the console is required. I could make this script non-dependent, but at some point it WILL be integrated into the rest of the module.

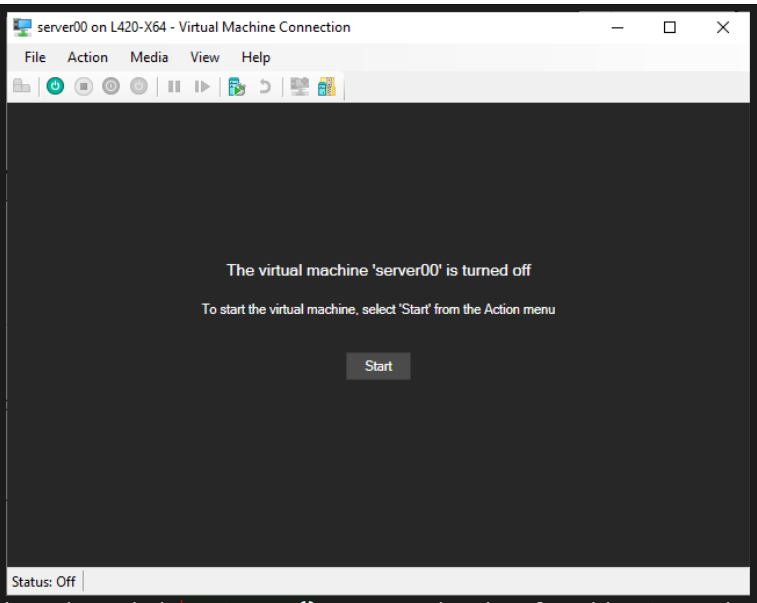
The idea here, is that if I want to do some really cool/amazing/jaw-dropping awesome fun-time stuff...?
Well, [FightingEntropy()] stops looking like an AFTERTHOUGHT, and it slowly becomes a factory of coolness.

You download [FightingEntropy()], and then all of the dependencies for each of the functions are included.
That's the idea behind the complexity of this module.

Anyway, now I could go ahead and begin creating the virtual machine with all of that template information up above with the following block.

```
    $Vm.New()
    $Vm.AddVmDvdDrive()
    $Vm.LoadIso($Vm.Iso)
    $Vm.SetIsoBoot()
    $Vm.Connect()
```

```
PS Prompt:\> $Vm.New()
>> $Vm.AddVmDvdDrive()
>> $Vm.LoadIso($Vm.Iso)
>> $Vm.SetIsoBoot()
>> $Vm.Connect()
[00:05:34.8045699] (State: 0/Status: [~] Creating : server00)
[00:05:38.6542121] (State: 0/Status: [~] Getting VmFirmware : server00)
[00:05:38.6756858] (State: 0/Status: [~] Setting VmProcessor (Count): [2])
[00:05:39.4126918] (State: 0/Status: [+] Adding VmDvdDrive)
[00:05:39.6672785] (State: 0/Status: [~] Setting VmDvdDrive (Path):
[C:\Images\Windows_Server_2016_Datacenter_EVAL_en-us_14393_refresh.ISO])
[00:05:40.1512843] (State: 0/Status: [~] Setting VmFirmware (Boot order) : [2,0,1])
[00:05:40.1522823] (State: 0/Status: [~] Getting VmFirmware : server00)
[00:05:40.3492796] (State: 0/Status: [~] Connecting : server00)
PS Prompt:\>
```



At which point, the method $Vm.Connect() opens up the Virtual Machine Connection manager...

Now, in order to AUTOMATE the process of installing Windows Server 2016 (easily adaptable for 2019 + 2022), we run this following code.

```
# // Start Machine
$Vm.Start()
$Vm.Control  = $Vm.Wmi("Msvm_ComputerSystem") | ? ElementName -eq $Vm.Name
$Vm.Keyboard = $Vm.Wmi("Msvm_Keyboard") | ? Path -match $Vm.Control.Name

# // Wait for "Press enter to boot from CD/DVD", then press enter
$Vm.Timer(2)
$Vm.TypeKey(13)

# // Wait for "Install Windows" menu
$Vm.Idle(5,2)

# // Enter menu
$Vm.TypeKey(13)
$Vm.Timer(5)
$Vm.TypeKey(13)

# // Wait to select installation
$Vm.Idle(5,5)

# // Select installation
$Vm.TypeChain(@(40,40,40,13))

# // Wait to accept license terms
$Vm.Idle(5,2)

# // Accept license terms
$Vm.TypeChain(@(32,9,9,9,9,13))

# // Wait Windows Setup
$Vm.Idle(5,2)

# // Windows Setup
$Vm.SpecialKey(67)

# // Wait partition
$Vm.Idle(5,2)

# // Set partition
$Vm.SpecialKey(78)
```

```
    # // Wait until Windows installation completes
    $Vm.Idle(5,5)

    # // Catch and release ISO upon reboot
    $Vm.Uptime(0,5)
    $Vm.UnloadIso()

    # // Wait for the login screen
    $Vm.Idle(5,5)
```

That code above will automate the entire installation process, and then when the installation process gets to the point where it asks for an administrator password...? That's when we run the following code and then kickstart the remainder of the installation so that it produces a server that runs Active Directory Domain Services, with FightingEntropy, Chocolatey, VSCode, PowerShell extension, BossMode (MY custom VSCode theme), and a bunch of services related to Internet Information Services, Domain Name Services, Dynamic Host Control Protocol Services, Windows Deployment Services, and even utilizing the function “Initialize-FeAdInstance” to distribute an OU, group, and user with administrative privileges to populate the Active Directory instance even more.

Oh, by the way, it also deploys WinRM management, renames the computer, sets the administrator password, configures DHCP with certain information, and it all does this from the following code that I'm about to paste below (which is also pasted earlier in this section...).

```
    # // Establish administrator account
    $Vm.SetAdmin($Hive.Admin)

    # Wait for actual login
    $Vm.Uptime(1,60)
    $Vm.Idle(20,5)

    # Enter (CTRL + ALT + DEL) to sign into Windows
    $Vm.Login($Hive.Admin)

    # Wait for operating system to do [FirstRun/FirstLogin] stuff
    $Vm.Timer(30)
    $Vm.Idle(5,5)

    # Press enter for Network to allow pc to be discoverable
    $Vm.TypeKey(13)

    # Launch PowerShell
    $Vm.LaunchPs()

    # Loads all scripts
    $Vm.Load()

    # Set persistent info
    $Vm.RunScript()
    $Vm.Timer(5)

    # Set time zone
    $Vm.RunScript()
    $Vm.Timer(1)

    # Set computer info
    $Vm.RunScript()
    $Vm.Timer(3)

    # Set Icmp Firewall
    $Vm.RunScript()
    $Vm.Timer(5)

    # Set network interface to null
    $Vm.RunScript()
    $Vm.Timer(5)

    # Set static IP
    $Vm.RunScript()
    $Vm.Connection()

    # Set WinRm
```

```
    $Vm.RunScript()
    $Vm.Timer(5)

    # Set WinRmFirewall
    $Vm.RunScript()
    $Vm.Timer(5)

    # Set Remote Desktop
    $Vm.RunScript()
    $Vm.Timer(5)

    # Install FightingEntropy
    $Vm.RunScript()
    $Vm.Idle(0,5)

    # Install Chocolatey
    $Vm.RunScript()
    $Vm.Idle(0,5)

    # Install VsCode | (Timer + Idle) Network Metering needed here...
    $Vm.RunScript()
    $Vm.Idle(0,5)

    # Install BossMode
    $Vm.RunScript()
    $Vm.Idle(0,5)

    # Install PsExtension
    $Vm.RunScript()
    $Vm.Idle(0,5)

    # Restart computer
    $Vm.RunScript()
    $Vm.Uptime(0,5)
    $Vm.Uptime(1,40)
    $Vm.Idle(5,5)

#     ____    ____________________________________________________________________________________________________
#    //--\\__//                                                                                            \\___
#    \\__//--- Launch PowerShell and Get-FEDCPromo -Mode [~]                                                ___//--\\
#     ---\\_____________________________________________________________________________________________//--\\__//
#         ------------------------------------------------------------------------------------------    ----

    # Login
    $Vm.Login($Hive.Admin)

    # Wait idle
    $Vm.Idle(5,5)

    # Launch PowerShell
    $Vm.LaunchPs()

    # Kill Server Manager
    $Vm.TypeText("Get-Process -Name ServerManager -EA 0 | Stop-Process -Force -Confirm:`$False")
    $Vm.TypeKey(13)

    # Launch FEDCPromo
    $Vm.TypeText("Get-FEDCPromo -Mode 1")
    $Vm.TypeKey(13)

    # Wait idle
    $Vm.Idle(5,5)

    # Switches to FEDCPromo GUI
    $Vm.PressKey(18)
    $Vm.TypeKey(9)
    $Vm.ReleaseKey(18)
    $Vm.Timer(1)

    # Command Tab | [Forest] Already selected
    $Vm.TypeKey(9)
    $Vm.Timer(1)
```

```
    # Cycle to tab control, tab over to Names, tab into Domain name
    $Vm.TypeChain(@(9,9,9,39,39,39,9))
    $Vm.Timer(1)

    # Type domain name
    $Vm.TypeText($Vm.Network.Domain)
    $Vm.Timer(1)

    # Tab into Netbios
    $Vm.TypeKey(9)

    # Type NetBIOS name
    $Vm.TypeText($Vm.Network.NetBios)
    $Vm.Timer(1)

    # Back up to tab control
    $Vm.PressKey(16)
    $Vm.TypeKey(9)
    $Vm.TypeKey(9)
    $Vm.ReleaseKey(16)
    $Vm.Timer(1)

    # Cycle over to (Connection/Dsrm) tab
    $Vm.TypeKey(39)
    $Vm.TypeKey(39)
    $Vm.Timer(1)

    # (Tab into/type) Dsrm password
    $Vm.TypeKey(9)
    $Vm.TypePassword($Hive.Admin.Password())
    $Vm.Timer(1)

    # (Tab into/type) Confirm password
    $Vm.TypeKey(9)
    $Vm.TypePassword($Hive.Admin.Password())
    $Vm.Timer(1)

    # (Tab into/press) Start
    $Vm.TypeKey(9)
    $Vm.TypeKey(13)

#    ____    _____________________________________________________________________________________________
#   //¯¯\\__//¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\\___
#   \\__//¯¯¯ Installing [~] [FightingEntropy()] Domain Controller                                        ___//¯¯\\
#    ¯¯¯\\___________________________________________________________________________________________//¯¯\\__//
#        ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯    ¯¯¯¯

    # Wait for (reboot/idle)
    $Vm.Uptime(0,5)
    $Vm.Idle(5,10)

    # Login
    $Vm.Login($Hive.Admin)

    # Wait for reboot
    $Vm.Uptime(0,5)

    # Wait for Active Directory to finish installing
    $Vm.Uptime(1,60)
    $Vm.Idle(1,10)

    # Login
    $Vm.Login($Hive.Admin)

    # Wait idle
    $Vm.Idle(5,5)

    # Launch PowerShell, then configure Dhcp
    $Vm.LaunchPs()

    # Configure Dhcp
    $Vm.RunScript()
    $Vm.Idle(5,5)
```

```powershell
    # Configure Dns (what's left)...
    # - Sign the zones

    # Populate Active Directory with (Organizational Unit, Group, User)
    $Vm.InitializeFeAd($Hive.Admin.Password())
    $Vm.RunScript()
    $Vm.Idle(0,5)
```

_____/ Output

Conclusion /

/------------

So, there ya have it.
The above information leads to a system that has all of that stuff configured and ready to go, so that I can then continue to develop the function “Get-FEDCPromo” even more.

The issue here is this...
If I continue to run into people that like to SABOTAGE my operations, and things of that nature, I have to come up with a system that is virtually indestructible even without the hardware, or a place to do the lab experiments, or even (morons/people) that have very reliably proven that they cannot be trusted, and don't actually care if they're doing everything possible, to sabotage my work on day-to-day basis.

Some people genuinely do not care whether or not the work I'm doing, will eventually lead to being a billion dollar program.

_____/ Conclusion

Michael C. Cook Sr.
Security Engineer
Secure Digits Plus LLC