[TFEInfrastructure]: 11/20/2023

Introduction

In this document, I'm going to cover a video that I recorded almost (2) years ago, which was a demonstration of the function [New-FEInfrastructure], which is part of the module [FightingEntropy(π)].

12/04/2021	[FightingEntropy(π)][FEInfrastructure]	https://youtu.be/6yQr06_rA4I

Seeing as how I was [recently injured] in a [felony hit and run] on [October 9th, 2023], and how my elbow was fractured in (4) places (*which required surgery and I've been slowly recovering my ability to use my left hand to [type stuff]*)…

I haven't updated the module for some time, partly because of the injury, but also because of the stresses of working on [Get-ViperBomb], and [Invoke-cimdb].

[Get-ViperBomb] is meant to help automate the configuration of a machines services, event logs, applications, AppX applications/packages, system settings, and other stuff related to system management and even (desired state configuration/features).

Both of these utilities are core components of the module, though [Invoke-cimdb] is more of a conceptualization of merging a (CRM/Client Relationship Module) with a front end to an SQL database and even [Active Directory Domain Services].

Regardless, there are plenty of things I know I need to work on, and update in the module.

Some of the things I'd like to finish working on in the function [New-FEInfrastructure]…
…which is meant to automate:

imaging	virtualization	installation	configuration

…of [servers] and [desktop clients] for a [computer], [network], [network of computers], or [network of networks of computers], and although this function does not currently install [Linux], [Unix], or [FreeBSD], it will eventually cover those operating systems.

Currently, [New-VmController] DOES make accommodations to install [Linux], [Unix], and [FreeBSD] on [virtual machines] straight from an [installation image], and it also contains functionality to automatically generate or map out an entire [virtual network], DHCP scopes and options, DNS pointers, and [manage credentials] that can interact with an existing [Active Directory Domain].

I have made many videos that feature [New-VmController], but it is just a much smaller version of [New-FEInfrastructure] that has newer features, controls, and logging capabilities.

As for the function [New-FEInfrastructure], it effectively shoestrings a number of [components] and [services] on a [Windows Server 2016+] instance, such as:

DNS	DHCP	ADDS	IIS	WDS	MDT	WinADK	WinPE	DISM	Veridian

…to [perform the above tasks].

However, being able to fully update THAT, would require a [decent workstation] with hardware that would allow me to [virtualize multiple systems], and to test these things in a fully functional [Active Directory Domain].

The above video link leads to the previous iteration of [New-FEInfrastructure] that was running on a [Dell PowerEdge R410] from (2009).

Here is an [extended description] what is happening in that video…

Description

[Megadeth - Peace Sells But Who's Buying (1986)] begins to play, with the header stating:

Preview/Demo featuring [FightingEntropy(π)] New-FEInfrastructure, PowerShell Deployment, FEWizard
Estimated Time: 120m

Switching the scene to the [local laptop], minimizing [OBS], switching to [RDP/Remote Desktop Protocol] over a customized [VPN/Virtual Private Network] tunnel (from/to):

10 Akin St., Johnstown, NY 12095 → 201D Halfmoon Circle, Clifton Park, NY 12065

(Highlighting + overloading) the function to [flush the module] from my [GitHub repository] at:
[github.com/mcc85s/FightingEntropy]

[Flushing the module] means to [remove all traces of my work], and then to [reinstall] all of the [prerequisites] to [perform the process seen in the video]. It is not all that dissimilar from taking a bomb, and [exploding my

car] ... but then, having all of the parts necessary to rebuild it all arrive at the same time, so that I can [immediately rebuild it] and drive where I gotta go.

This is important, because the module is meant to be able to distribute to all of the child item computers that the function is actually preparing.

Executing the function that was just loaded into memory, which flushes the module using a previous version of [FightingEntropy(π)][2021.10.0]… [the newest method is exponentially faster than this].

Once the module is reinstalled, the [PowerShell] host displays information about the module, author, company, registry path, resource path, classes, functions, et cetera.

The newer versions (2022.10.1+, some no longer available) are a bit more [compact] and [extensible], as I created a class called [ModuleController] which has [so much additional functionality] in relation to [regenerating the module + manifest], [logging], [writing to console], [dumping the console to file], [writing stuff to file], [writing stuff to registry], [accessing files] like [controls], [graphics], and [functions]…

…and being portable enough to use within [other components of the module].

In other words, the [ModuleController] provides much more [granular control] than the iteration seen in this video, and I've yet to implement these changes into the function [New-FEInfrastructure], and then [demonstrate] how much work it does to manage [system administration] and [network resources], as well as [provide evidence] that I am a [solutions architect] and a [security engineer], in addition to a slew of other titles and roles.

Then, I call ($Module = Get-FEModule), which automatically imports the module into memory, because the module is now a part of [installed modules]. Doing so actually uses the function [Write-Theme] to display in the console:

```
-------------------------------------------------
| Loaded Module [+] FightingEntropy [2021.10.0] |
-------------------------------------------------
```

…in a cool green, blue, and white themed message box in the console.

This is a [signature] of how [detail oriented] I am about [writing things to the console].

After declaring $Module, I then enter it into the console to get the loaded information back, which shows similar information to when the module was installed with the function [FlushModule]. By the way, the [Veridian/Hyper-V] window is seen to the right of the [Visual Studio Code] window, which shows a number of [virtual machines] that I already had on the [Dell PowerEdge R410] server.

I then call [New-FEInfrastructure], which is the custom function which loads the function, and says:

```
-------------------------------------------------------
| Initializing [~] Infrastructure Deployment System |
-------------------------------------------------------
```

...and then a XAML based graphical user interface that I designed myself in [Visual Studio], called [Get-FEADLogin], pops up on the screen.

What can be seen in this interface, is a login panel.

[Login]: Access Active Directory Resources
[Username], [Password], [Confirm]…

…as well as a dialog box that shows information for a domain controller that was found, its' [IP Address], [DNS Name], the [domain], and [NetBIOS] info. It also contains a textbox for [manual entry], and even provides an option to change the [login port] from (389), or you could enter (636) for [LDAP SSL].

This GUI is seen in this demonstration multiple times, and what it does is it accesses a component of [Active Directory] called the [DirectorySearcher] object, in order to validate whether or not the credentials entered into the login panel are (accurate/legitimate).

This is important, because if the login information fails to successfully log into [Active Directory], then [the rest of the process will fail to work].

I've yet to add [additional functionality] to show the [graphics] that [validate the entries], and prevent a user from failing to [completely] enter the [required information].

I then enter in [root@securedigitsplus.com] which acted as a hybrid servicing and admin account for the domain controller [dsc0.securedigitsplus.com], and a [16+ character password] with [complexity requirements] in the [Password] and [Confirm] password boxes.

Selecting one of the items in the DataGrid would've populated the server name box with that information, but I was having a conflict with that information since the server had [dual network interface cards] on the same network (originally meant for link aggregation), so I entered the [alternate address] manually.

At which point, the login was successful, and so the function began to collect information about the server in a similar manner to how the newer version of the module does, accessing [WMI/Windows Management Instrumentation] and [Get-CimInstance] information, like operating system, disk, network, processor, computer, UUID, motherboard, firmware, chassis, architecture... all now a part of [Get-FESystem] and [Get-ViperBomb].

Then, the function collects the current IP configuration, DHCP information, DNS (zones + records), Active Directory Domain nodes/OU/Sites/Computers/Users, Veridian info, WDS info, Mdt/WinPE/WinADK info, as well as IIS.

This information falls under a property of the [main controller class] to further manage all of those services.

Then, it loads the US postal code database which allows zip codes to be entered into potential Sites and Services topology. So, if a network existed in (12065), then the domain name for that zip code, state and town would be automatically generated.

Then, it loads the controllers for Sitelist, NetworkList, Sitemap, AddsController, VmController, loads all of the existing nodes for the VmController, meaning that all of those virtual machines that have a (*.vmx/*.vmdk) file will be targeted and templatized into the interface for management, including the file sizes for each item found…

Then, it loads the ImageController which is meant to locate .iso files to be used for (*.wim) file extraction, the image controller has been updated since then to accommodate non-Windows images.

It also loads the UpdateController which is meant to interface with files found through WSUS or in a target path so that each update file found can be processed and show its' content in the interface.

Lastly, before the [FEInfrastructure] GUI is loaded (which I also made), the MdtController loads all of the (Mdt shares/persistent drives/*.wim files) so that each Mdt deployment share can be managed via the GUI.

Well, it also loads the [WdsController] which interacts with the Wds service to load PXE boot images into the service seamlessly.

At which point, this message appears:

```
| Initialized [+] Infrastructure Deployment System |
```

…and then the graphical user interface for [FEInfrastructure] is loaded:

```
| Infrastructure Deployment System |
```

Description

Infrastructure Deployment System

Contains multiple tabs that control all aspects of the components I've just specified…

[Module] - shows the same information that was seen in the console about the module, as well as the files within the module.

[Config] - shows information aout what was collected from the server, and has subtabs like [Role], [System], [Network], [Dhcp], [Dns], [Adds], [Hyper-V], [Wds], [Mdt/WinADK/WinPE], and [IIS].

I'm not going to cover each subtab, because it is a lot of information and it's rather self-explanatory.

[Domain] - shows information about the domain, allows entry of the [Organization], [CommonName], as well as zip code entries to manage desired zip codes and domain labels.

This also validates the existence of potential Distinguished Names of sites and service links.

[Network] - Establishes the IP subnet ranges to authorize potential items for ADDS authorization.

[Sitemap] - Allows the domain sites and the subnets into a single object in order to manage sitelinks.

[Adds] - Manages all of the nodes within a particular unique sitelink, to automate the creation of gateways, servers, workstations, users, and service accounts.

This part is critical because it combines all of the information from the sitelinks in order to propagate the Adds information into the VmController nodes.

[Virtual] - Basically a much more complex version of [New-VmController], which allows management of virtual gateways, switches, servers, and workstations.

[Imaging] - Manages installation images, and .wim file extraction via DISM for Mdt share generation.

[Updates] - Manages [Windows Updates] to implement packages into (*.wim) files, effectively slipstreaming [Windows Updates] into the installation image, rather than to download the updates after the installation takes place.

This would effectively reduce bandwidth post-installation.

Not complete or implemented, just there as an idea to expand upon at a later date.

[Share] - Manages the entire process of the [Michaelsoft Deployment Toolkit] share creation, which allows the implementation of [PowerShell Deployment] by [Michael T. Neihaus], [Johan Arwidmark], [Mykael Nystrom], [Jordan Benzing], [Soupman], and the rest of the guys at [FriendsOfMdt].

There is a lot of stuff happening from here on out, so I won't be so specific about every little detail seen in the video. Though there are parts where the action is stagnant even though the system is busy doing stuff, it's all been orchestrated in this process.

I click on the module tab which shows information about the module, such as the GitHub repository, the author that went insane making this entire thing, and the company he started to focus on all of the things seen in this video… …a ridiculous description about [fighting ID theft and cybercrime] or whatever, and the installation date which shows exactly when the [FlushModule] completed installing the components of the module seen in the lower DataGrid.

Then, I click on the config tab and roll through all of the information that was collected from the server, all necessary to automate the process after this dialog has built a new PXE image for Wds to deploy an installation via the PSD shares and the [FEWizard] dialog in the PXE environment.

Then, I click on the domain tab to enter in the company name, and the common name which would be part of a [security certificate] once I complete the distribution method that would allow something like [LetsEncrypt] on an [OpnSense gateway] properly configured with the [top level domain provider] API keys, such as [GoDaddy] or [CloudFlare], allowing [SSL/secure socket layer] connections to other nodes in the (topology/domain) either internally, or externally for visiting websites in IIS and things of that nature.

Well, eventually, the [PSD/MDT] shares will be able to leverage [HTTP/HTTPS], rather than [SMB], theoretically reducing the [operating system installation time] with something like [BranchCache] or whatever it's called.

It also has a field for zip code entry which automatically generates unique domain names so that the gateways will be able to interface with [Active Directory] regardless of whether they are [FreeBSD]-based, or otherwise.

As it does this, it will allow [validation] of an [existing sitename], or to (create/delete/refresh) them…

Then, I click on the network tab to determine what the DHCP scopes should be for that particular sitelink.

I used the [string notation format] that is similar to that of [OpnSense], like [172.16.0.0/19], which provides a range from [172.16.0.0] to [172.16.31.255], with the [first address] being the [network address], and the [last address] being the [broadcast address].

Providing the range in [string notation] is pretty important, because it doesn't have to create (8192) objects in memory, it just creates (1). This information is meant to be propagated to potential [DHCP servers].

Also, this area allows the management and validation of existing entries, or to (create/delete/refresh) them.

Then, I click on the [sitemap tab] which allows an admin to create a [sitelink], as well as select whether or not they want to create (a/multiple) gateway(s), server(s), computer(s), user(s), or (a) service account(s).

It also allows for [validation] of those objects, being [organizational units].

The information here, allows for the [previous tabs] to [compile the template objects], and the objects just get bigger and bigger as they're processed from one tab to the next, until the [Virtual] tab.

Then, I click on the [Adds tab] which populates some of the information depending on the selected sitelink name. This information is used to create potential objects on either a single basis, or multiple.

So, if you wanted to create an array of these [object[]], then this is where the automation tasks would start to pay off, rather than having to individually hand-craft them for import into [Active Directory].

Since I wasn't creating [gateway], [servers], [users], or [service accounts], I only selected [workstation], and then imported an [existing text file] named [ws-12065.txt] with the [Windows Forms] open file dialog box, which automatically loaded that file into memory, and with that file, it was able to create an array of [workstation object templates], all of which already existed as a [virtual machine] in the [Hyper-V] window…

The aggregate box was populated in this process, and injected the template objects.

I then clicked the "get" button, which populated the output, and selecting a particular object in that box shows the individual properties of those templates below.

By clicking "remove", I was able to remove those existing entries from [Active Directory], though the actual [workstations] themselves still exist in the [hypervisor].

By clicking "new", it was able to [recreate those objects], albeit with a [brand new GUID] for each item.

Then, I click on the [virtual tab], and begin to play around with the objects that were already created, as the interface allows for me to [delete existing virtual machines] that match the newly generated [Adds nodes].

Clicking the "[delete] existing nodes" button brings up a confirmation box…

```
_________________________________
| Warning [!] Are you sure ... ? |
---------------------------------
```

This is [important], because in the [wrong hands], some idiot COULD actually remove a lot of existing virtual machines, and you don't want that. But- in the [right hands], idle hands are the devils plaything, and thus, I went ahead and committed to deletion, which then interfaces with Hyper-V and removes each individual workstation, its' (*.vmdk) and (*.vmx) files, and [relinquishes their existence].

Relinquishing their existence… is a rather dramatic way to say "*deleting all traces of each virtual machine*". But- it's accurate.

Probably sounds mean, but the thing is, after all of the verbose output shows in the console, the tool allows me to [recreate them], as well as to [configure them] with the [proper settings] that I want each [virtual machine] to have, like [RAM], [HDD size], [generation 1 or 2], [CPU cores], and [type].

The [type] selected is [Network], which will leverage the (Wds/Mdt) installation method.
The [path] will be the [filepath] where the [virtual desktop infrastructure] will place all of the machines.
The [image] (when available), looks for the (*.iso) file to use to [create the virtual machines].

The [script] is something I'd been working on with [New-VmController], where a [script] can be loaded to configure the machine from the server via [PowerShell Direct], or from the [workstation] itself, once it has fully installed the [operating system] and is ready to process things similar to a post-install task sequence.

Then, clicking "Get" will validate whether the function can procede,and then clicking "New" will actually go right ahead, and create the [virtual machines] via the combined [template objects] that the last (5) main tabs combined.

All according to the specifications that were selected with the GUI.

Then, I click on the [imaging tab], but I'm not quite ready to use it.

I passed right over the updates tab for now, and went to the [share tab].

This is where the [magic] starts to happen.

In the top DataGrid, is a single share named [FE001], with the corresponding properties for the [type], [root filesystem path], [share name], and [description]… though some of this stuff will automatically populate.

I scroll through the existing Mdt share properties, which shows information similar to that of the [Michaelsoft Deployment Toolkit], and then I click on the minus button, which loads a prompt that says:

```
_____________________________________________________________
| Warning [!] Proceed? / This will remove the share [FE001] |
-------------------------------------------------------------
```

Of course, [we want to remove it], so I clicked "Yes", which then provides all of the verbose output eliminating all of the files in that share, and once it's done, it says:

```
_____________________________________
| Removed [!] Persistent Drive (FE001) |
-------------------------------------
```

…removes the object from the DataGrid, and clears all of the information in the GUI pertaining to that (drive + share).

At some point, the [newer version] of the [module] will have all of this [verbose output] saved to the [module controller] console object, but for right now, it's just using the native verbose switch to produce the information in the console.

Since there's no drive selected in the DataGrid, portions of the GUI are disabled and greyed out, which prevents a user from entering information before it's working with a known existing (drive + share).

We'll come back to this tab in a little bit…

Then, I click on the [imaging tab], which allows for the entry of an image path.

Clicking “*select*” opens a [Windows Forms FolderBrowser] dialog box, to which I select the folder that has all of the (*.iso) files I want to use to extract (*.wim) files for a new [Mdt share/persistent drive].

Once the folder is slected, the function imports all of the (*.iso) files in it, so that I can select which ones I want to extract the (*.wim) files from.

The first I select is the [Win10_21H2_English_x64.iso], which is the standard installation image for [Windows 10] from the second half of (2021), then I click on “*Mount*”, which accesses [DISM] to show the content of that image.

This process takes a few seconds because of the age of the machine, but when it comes up, it processes each found (*.wim) file into an object that has the [edition], [expanded size], as well as the [architecture].

We're shooting to [eliminate as many assumptions as possible] in creating the desired output, and thus, selecting the editions I'd like to extract allows me to create a (*.wim) file extraction queue object, as the process isn't going to extract the (*.wim) files until ALL of the [desired images] are in the queue.

This process is important, because we'll see the [selected images] in the [FEWizard tool] within the [PXE environment] after [Mdt] updates the [boot image].

Which- can't be done quite yet, because no [Mdt shares] currently exist.

After creating the [queue object] of desired [selected indexes], I clicked the “dismount” button which removed the (*.iso) from view, and this cleared the [image viewer DataGrid].

I then repeat this process for [Windows 10 Enterprise], as well as [Windows Server 2019/2022 Datacenter] editions, respectively.

Then, clicking the “*select*” button opened another [Windows Forms FolderBrowser] dialog box for [DISM] to extract the (*.wim) files to. Once that was selected, I clicked on “*extract*”, which procedurally loads each of those (*.iso) files and properly extracts the image to a folder with the same name as the tag that will be used to import into the [Mdt share].

This process took [8:50 → 13:22]

```
$Start = [TimeSpan]"00:08:50"
$End   = [TimeSpan]"00:13:22"
$Time  = $End - $Start
$Time.ToString()
```

```
00:04:32
```

(4) minutes and (32) seconds, on an old [Dell PowerEdge R410].
Not bad. A newer system could probably do that in (1/2) → (1/4) of the time, maybe less.

Anyway, then I click on the [updates tab], and just to show off what I conceptualized, I clicked on "*Path*" which opens another [FolderBrowser] dialog box, and I selected a path to where I had a few downloaded [Microsoft Update] files, and the function used a native Windows executable to extract pertinent information about each update, not unlike how the [Microsoft Update Catalog] does...

Once all of those files were processed, the aggregate DataGrid lists each component, and selecting] any of these components allows each object to populate the viewer DataGrid. At some point, this panel will allow these updates to be [slipstreamed] into the [installation images]…

However, [that process is rather lengthy], especially if there are multiple (*.wim) files and editions.

To convey the sentiment, I then type in "*not yet implemented*" into the textbox for the (*.wim) file path, because more thought needs to go into how to properly allocate the [correct updates] for the correct (*.wim) files.

Then, I select the [share tab] again.

I clicked on the type, and dragged the ComboBox down to [PSD], which will install the necessary files to deploy a [PowerShell Deployment] share.

Then, I clicked on the "*root*" button, which opens another [FolderBrowser] dialog.
This is a little tricky, because you want to select a path that has no existing files in it.

I created a new folder named [FlightTest], basically the same as before.

At which point, everything is needed to spin up a new Mdt persistent drive.

By clicking on the [+] button, the [necessary prerequisites] for creating a [new share] are generated, such as a [Samba share], and a [PSDrive].

[Mdt] has a [module file] that will [deploy all of the necessary files to that location], but since [PSD] was selected, the function [New-FEInfrastructure] has to ALSO copy the files necessary to install the [PSD] deployment method.

In this process, the [correct templates] are copied from the [FEModule control path].
It also [unblocks] all of the [PSD files], and shows the [verbose output] as it does this.

Once the [share exists], the console says:

```
-----------------------------------------------
| Success [+] Added Persistent Drive: (FE001) |
-----------------------------------------------
```

And, the DataGrid has a new item in it, when selected, populates the corresponding dialog boxes for the [Mdt drive] properties.

Clicking on [branding], the boxes are all empty.

But since the server has all of the necessary registry paths filled out, clicking on "*collect*" will retrieve those values to propagate to [child items] created via the [Mdt share].

Clicking the "*apply*" button ensures that those values are committed to being processed.

Clicking the [local tab], the [local administrator] information is blank.

Filling out the [username], [password], and [confirm] boxes allows the [post-installation process] to utilize that information.

Clicking "*apply*" ensures that those values are committed to being processed.

Clicking the [domain tab], we can fill out the information manually, OR…

…clicking on "*login*" allows the [Get-FEADLogin] function to load again, allowing for another account to be used to [deploy potential operating systems].

I entered in "fightingentropy@securedigitsplus.com", as well as another 16+ character password with complexity requirements in place.

Clicking on "*Machine OU*" brings up an [organizational unit browser], selecting the [correct one] ensures that the [bootstrap.ini/customsettings.ini] file(s) receivee the correct place to put the new computer.

Clicking "*apply*" ensures that those values are committed to being processed.

Clicking the [OS/TS tab], and going to [import subtab] allows entering in the path where the (*.wim) files were extracted, and then those items are [imported] into the [import browser DataGrid].

The files are not yet imported to the share, but by clicking on the "*import*" button, it allows them to be copied if the ComboBox says "copy" (which takes longer), or to move them if it says "*move*".

Clicking on "*import*" actually imports the (*.wim) file into an [operating system object], but it ALSO creates a [standard task sequence object] as well, which means that right out of the box, the [boot images] just need to be processed... this might not be the most suitable thing if a particular administrator wants to add additional processes or task sequences...

But- the [post-installation process] will make accommodations to further sculpt out the [desired state configuration] of each system, and that's ultimately what [Get-ViperBomb] will do.

Anyway, now that all of the [operating systems] and [task sequences] are [imported] into this brand new share, the [current subtab] will show them with a [label], [name], [description], and [version].

All pretty important, as they each need to be [unique].

Clicking on the [config tab], the [Bootstrap subtab] has very little information in it, and yes, the content of the dialog box points to the file that the process created when it generated the share.

There's not much in it, but clicking on "*generate*" will pull all of the information that was committed into the [bootstrap.ini] file textbox…

Clicking "*apply*" ensures that those values are committed to being processed into the [bootstrap.ini] file itself.

Clicking on the [custom tab] repeats a similar process with [more information] pertaining to fields that the [bootstrap.ini] file does not cover.

Clicking "*generate*" inserts the proper information into the [customsettings.ini] file textbox.

Clicking on "*apply*" ensures that those values are committed to being processed into the [customsettings.ini] file itself…

Clicking on the [post subtab], handles all of the things that someone would want to do as [post-installation] criteria, in [PowerShell]. Currently, the information to install [FightingEntropy(π)] is there, but this will [require an internet connection] and the [current code will not work], because the [module] has been updated to a large degree, as the version featured in the video is [no longer existent].

However, the [module] could indeed be [localized], and that was a major concern when I developed the newer version of the module (2022.10.1+).

It also contains the information needed to import the (*.csv) file that has additional information to inject the graphics and company information into the system. Though, this part may be replaced in entirety, since the module will have a feature that does this via [Get-ViperBomb].

Clicking on "*apply*" ensures that those values are committed to being processed into the [Install-FightingEntropy.ps1] file itself.

Lastly, clicking the [key subtab], allows the [DSKey.csv] that I just specified, to be generated and put into the path that the [post script targets].

Clicking on "*generate*" will create the customized (*.csv) file, and clicking "*apply*" will write those values to the [DSKey.csv] file itself.

At this point, the ComboBox at the bottom has the "*Full*" option selected, meaning it's going to [create the entire boot image] that will be submitted to [WDS] for processing.

This part takes quite a while, [17:11 → 32:07]

```
$Start = [TimeSpan]"00:17:11"
$End   = [TimeSpan]"00:32:07"
$Time  = $End - $Start
$Time.ToString()

00:14:56
```

…14 minutes and 56 seconds, which would [probably take a lot less time on a faster system].
At [32:07], the boot image is automatically handed off to the [WDS service].

```
________________________________________________
| Updated [+] Mdt Deployment Share (FE001) |
------------------------------------------------
```

This takes about a minute… and then the dialog box for the [Infrastructure Deployment System] can be closed.

```
                                                                                          ___________________________________/
\________________________________________________________________________________________/ Infrastructure Deployment System
  Virtualization /-----------------------------------------------------------------------------------------------------------\
/---------------
```

Virtualization

At this point, [the utility has primed all of the prerequisites for the installation to proceed], and now I spend a couple minutes [staging the virtualization lab], which effectively does many of the same things as [New-VmController].

A dialog box shows up to allow login credentials to be used for the post-installation process, which is the same information as in the beginning, I just hadn't made it part of the same process since the function [New-FEInfrastructure] was all self-contained.

This part of the process is focused on orchestrating (1) of the [virtual machines] itself, to utilize the [PXE environment], and allow me to navigate the [FEWizard GUI], which kickstarts the [PSD installation] method.

This was all very experimental when I recorded this, and to be perfectly honest, this part takes a very

long time on this system, seeing as it is a rather old [Dell PowerEdge R410].

I had to take a step back from continuing to test with this setup, because of how many hours I was dumping into starting everything over from scratch, and the system just wasn't able to allow me to test the environment in a much faster manner.

Prior to recording this particular video, I had made some changes that prevented the written process from working, mainly because of the underscore in the image tag.

$Log.TX("Initializing Workstation") was just a message indication, but the newer version of the module will allow the [embedded console tracker] to do that instead.

[Reset-VMLab] just initiates the object from file, as the function [New-FEInfrastructure] deployed files to disk that allow the [virtual machine] to be [instantiated] and [controlled] from a [filesystem object], since this process was [crashing quite a lot], and eliminating my ability to use things in memory to debug or diagnose what was happening in each iteration.

Once the [virtual machine] is on, it immediately goes to [negotiate the DHCP/PXE process], and previous versions of the [boot image] were seen in [WDS]. The process actually looked for the [proper index] to tell the [virtual keyboard] how many times to hit the down button to [select the brand new image].

Now, I don't know why [Megadeth - My Last Words] wasn't on that playlist I had running, because that's a really awesome song… but after "*Wake Up Dead*" played for a second time, I decided to switch the music to [Iron Maiden Greatest Hits Volume 1], which is no longer available.

But- that's ok.
[Megadeth], and [Iron Maiden] are (2) really awesome bands.
Side point.

Once the [boot image] is loaded, the [PXE environment] is then loaded…

And this is where all of the [modifications] I've made to the [PSD scripts] begins to be seen by the executable that controls everything displayed on the background. Also, notice the [Secure Digits Plus LLC] background…

It takes a moment to load up the GUI, and all of the messages seen under "*Progress*" were things that I either edited or kept the same in order to load the [FEWizard GUI].

This GUI was something that I thought of back in (2019) when I ran across [Damien Van Robaeys PowerShell Wizard].

I had to learn how to write with [XAML] and do it with [PowerShell] in order to get around needing to do stuff with the [wizard.hta] hypertext application that is part of the [native MDT process].

I also had to make sure that if [MDT] was selected, that it would only modify certain aspects of that method.

However, in this [graphical user interface] at [39:01], there are tabs:

[+] [local]   - keyboard, language, timezone, secondary keyboard, product key, etc.

[+] [root]   - task sequence selection, applications, drivers, packages, profile, OS, extended shares, etc.

[+] [system] - system information, system name, password, and hard drives

[+] [domain] - organization, OU, homepage, domain username, and network information

[+] [control] - domain + netbios, installation mode (new, refresh, virtualize/devirtualize),
                and (backup/restore) information.

In the [local tab], certain things are required, like the [keyboard], [time zone], and even the [product key], but because this is a [test system], we're not using a [product key].

In the [root tab], you're able to select what [task sequence] you want to use, in addition to selecting [applications], [drivers], and all that other stuff that I wasn't primarily focused on demonstrating since this part was relatively [uncharted territory] for me.

In the [system tab], you're able to change the [system name], which isn't fully implemented because the process somehow fails to localize some of the variables upon installation. Still, this provides a way to do it upon installation time, or to name the system something other than the [fan speed], like [Mykael Nystrom] talked about back at [Ignite 2016].

In the [domain tab], this shows all of the stuff pertinent for the [domain], [OU], and [networking type stuff].

There's an area for logging into [Active Directory], but that's just pulling information from the [bootstrap.ini/customsettings.ini] files.

In the [control tab], I've made it rather apparent that the utility is meant to [transfer systems] and [profiles] around, whether it is [virtualizing a system], or even [devirtualizing a system].

[Devirtualization] is the process of taking a [virtualized system], and turning it into a [real system].

This is the exact opposite of [P2V], and apparently it is [difficult to do] because of the [potential issues] that may arise. Still, [I've done it plenty of times], albeit manually.

Once all of these things are acceptable, clicking the start button kickstarts the installation process.

The installation process takes from [40:14 → 1:38:20]

```
$Start = [TimeSpan]"00:40:14"
$End   = [TimeSpan]"01:38:20"
$Time  = $End - $Start
$Time.ToString()
```

```
00:58:06
```

(58) minutes and (6) seconds is a pretty long time, and the only reason it takes that long is because of the [age of the system], the [Dell PowerEdge R410]. And also, the fact that the process hits a few speed bumps in relation to how the scripts I modified are handling those modifications.

This is the main reason why I had to take a step back from further implementing changes to the process, because that is a really long time for a system to [realize].

But also, I lost a few minutes because I forgot to have the process change the [virtual machine's boot priority] after the [first reboot].

Still, once the desktop appears, I took a few minutes to get back from whatever else I was doing…

And then I started looking through the [SMSOSD logs] in order to figure out:

[+] what happened
[+] when it happened
[+] why it happened
[+] whether the (7) dwarves were actually with [Snow White] at the time that they claimed to have been with her
[+] why they were hesitant to tell me the truth
[+] whether I could trust the (7) dwarves, and…
[+] if [Snow White] was actually just sleeping, or if she was GONE…

Once you suspect that (7) dwarves may have killed [Snow White], they start to look like they're not very innocent after all… nah.

Look, I'm kidding about the (7) dwarves… they were not involved in this process.

Anyway, the script "*Install-FightingEntropy*" was in fact, transferred over to the system, however it failed to launch because of something I didn't configure correctly in the [PSD scripts].

So then, I decided to [install the module] by utilizing the [same installation string] as in the beginning, when I used the custom function [FlushModule].

After installing the module, I didn't really accomplish much except to load the module into memory, and play around with features that have since been [antiquated].

The connection to the share was effectively [lost] (*unless it wasn't and I just didn't invoke [Get-SmbMapping]*), which prevented the module from being able to load the information in the [DSKey.csv] file, and as such…

…I decided to end the video.

Conclusion / Virtualization

This video is almost a couple years old, and for what it's worth, I managed to orchestrate a lot of work that required some planning ahead in order to comparmentalize the things I've continued to work on over the last (2) years.

Here's a video of what I had [last October]…

10/28/2022	[FightingEntropy(π)][2022.10.1]	https://youtu.be/S7k4lZdPE-I

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