On the way to Rapid Recovery, part 4: Completing the foundation of the VMware omnibus script

In my last blog post, we made steady, albeit somewhat unexciting, progress by delving into encryption. One of my coworkers with whom I toiled together along the same lines for an unrelated project told me — I quote — "Encryption is fun." I don't feel the need to extract (or decrypt) the meaning of his words.

However, I believe that anyone would agree that the cool part of my previous post was using the MD5 hash to convert any password to a 16-byte (128-bit) encryption key, since the Advanced Encryption Standard (AES) requires keys to be of a fixed length byte array of 128-bit (16 bytes), 192-bit (24 bytes) or 256-bit (32 bytes). Since passwords are strings of variable length, calculating the MD5 hash of the password, which is always 16 bytes (128 bits), and using it as the encryption key is the good choice. More about AES can be found at http://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard.

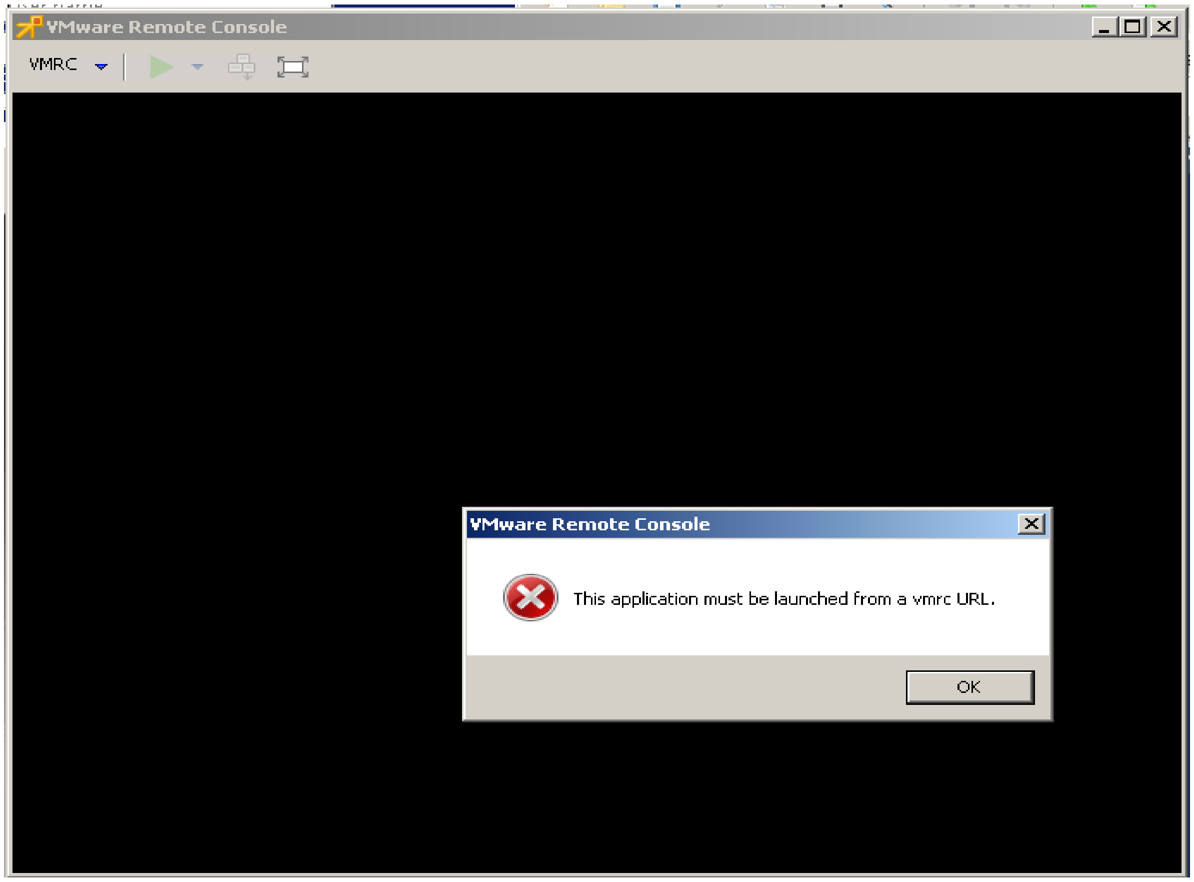
At this point, we are ready to leave encryption behind and attack the last two blocks of the foundation of the VMware omnibus script promised in the first blog post of the series.

The first block is how to operate the VMware Remote Console (VMRC). Just for the record, VMRC came initially as a browser plug-in that allowed connecting to vCenter-operated virtual machines (VMs) and interacting with the desktops of those VMs. This functionality is available natively in PowerCLI (the Open-VMConsoleWindow commandlet) but is mostly useless as both the Java version (called NPAPI) and the ActiveX version are now deprecated. The only browser that (kind of) still supports NPAPI is Firefox. VMware has addressed the issue by offering a stand-alone remote console. If you remember the previous posts, downloading and installing it was one of the prerequisites for our script. The default installation path of the executable is C:\Program Files (x86)\VMware\VMware Remote Console\vmrc.exe

Let's give it a try!

```
PS C:\ > Start-Process -FilePath "C:\Program Files (x86)\VMware\VMware Remote Console\vmrc.exe"
```

The console comes up, but in an unusable way.



VMware Remote Console

This application must be launched from a vmrc URL.

VMRC URL? Obviously, there's more to it. As much as I don't like it, it looks like reading some documentation is needed. Unfortunately, the VMware documentation, which usually is very clear, shows very little on how to launch VMRC from PowerCLI/PowerShell. However, after some poking around, it turns out that there are two ways of launching VMRC from a command line:

1. C:\Program Files (x86)\VMware\VMware Remote Console\vmrc.exe vmrc://[USERNAME]@[VC]/?moid=[VM-MOREF-ID]
2. C:\Program Files (x86)\VMware\VMware Remote Console\vmrc.exe vmrc://clone:[VC-TICKET]@[VC]/?moid=[VM-MOREF-ID]

It looks self-explanatory, but it is not. Let's attempt to translate:

- vmrc is obviously some kind of protocol. Nothing to do about it.
- [USERNAME] looks to be what it implies — a user name. Nothing to do here, either.
- [VC] is the FQDN or IP address of the virtual center. (It took me some time to figure it out.)
- [VM-MOREF-ID] is the Managed Object Reference ID of the VM — more about that later.
- [VC-TICKET] is a clone ticket that embeds information necessary to connect the VMRC to an already opened VMware session — I will talk about that later, as well.

After going through the elements of the two commands, it turns out that two pieces of information need to be assessed: [VM-MOREF-ID] and [VC-TICKET].

Every object within a vSphere environment is tracked by a unique identifier. This is composed by a prefix (that is, "vm", "datastore", "host") and a number, the two being separated by a hyphen. For instance, in my environment, the VM-MOREF-ID of the VM named "Skaro" is vm-289.

The interesting thing is that the ID of the VM returned by PowerCLI is a descriptive prefix plus the MOREF-ID of that machine. In my case, the ID of Skaro returned by PowerCLI is VirtualMachine-vm-289.

The VM-MOREF-ID can be obtained either by parsing the VM ID or by digging deeper into the VM information as shown below. First, the obvious way.

```
PS C:\temp>
PS C:\temp> (get-vm -Name skaro).Id
VirtualMachine-vm-289
```

Just for the record, the VM ID from the demo above can be parsed, as shown below.

```
PS C:\temp> $fullid = ((get-vm -Name skaro).Id).split("-")

PS C:\temp> $moref = "$($fullid[1])-$($fullid[2])"

PS C:\temp> $moref
vm-289

PS C:\temp>
```

The same result may be obtained in a single line, as shown below.

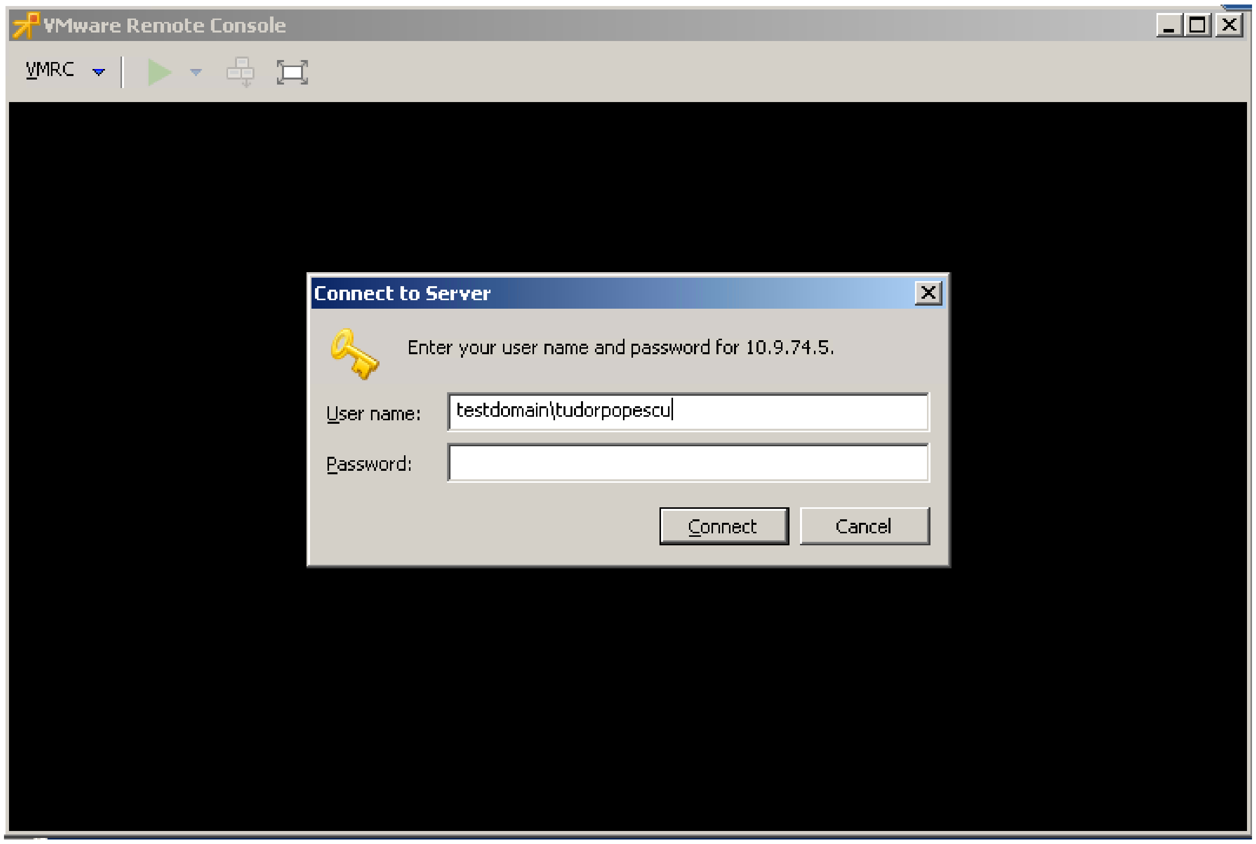
```
PS C:\temp> (get-vm -Name skaro).ExtensionData.moref.value
vm-289

PS C:\temp>
```

It goes without saying that the dig-down method is the preferred one to use as it requires less administrative effort to perform.

You can try the first method to connect to a VM using VMRC.

```
Start-Process -FilePath "C:\Program Files (x86)\VMware\VMware Remote Console\vmrc.exe"
vmrc://testdomain\tudorpopescu@10.9.74.5/?moid=vm-289
```

That seems to be successful (see below). As a nice touch, the user name was taken in the domain\user format, and the vCenter IP address was accepted, as well.

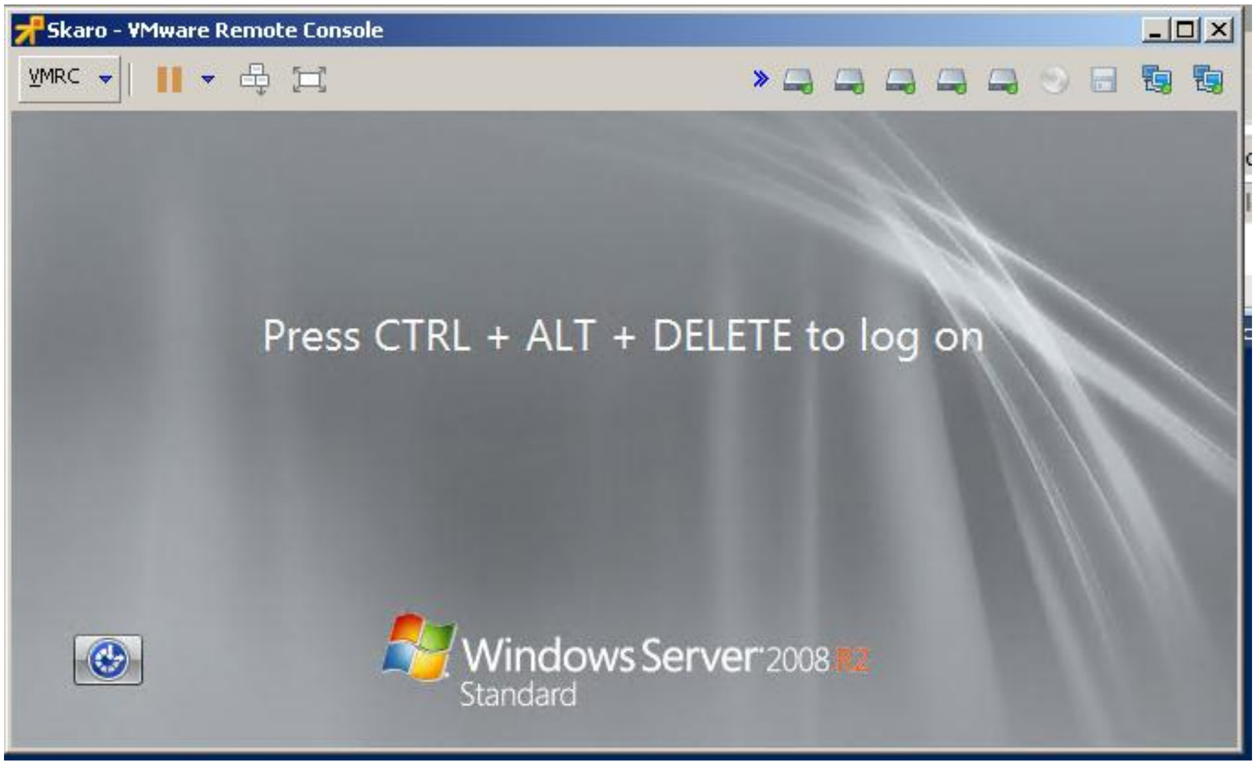


Enter your user name and password for 10.9.74.5.

User name: testdomain\tudorpopescu

Password:

After entering the password for the vCenter, the desktop of the desired machine shows up.


Skaro - VMware Remote Console
VMRC
Press CTRL + ALT + DELETE to log on
Windows Server 2008 R2
Standard

However, this isn't quite what we wanted, as the vCenter password needs to be entered manually. There may be a way of doing it automatically, but I didn't find how to do it.

As such, the second way to launch VMRC needs to be tried. Since it uses a clone ticket, no extra authentication steps are required.

According to the VMware VIX-API documentation: “[A] connection can be authenticated using an existing vSphere API session instead of with the username-password combination. To use an existing vSphere API session, a ‘clone ticket’ is required. To get a clone ticket, call the API method AcquireCloneTicket with the SessionManager object reference. Then using the ticket string returned by this method, call VixHost_Connect with NULL as the userName and the clone ticket as password.”

Please note that, in my experience, a clone ticket can be used just once. As such, for each new VMRC console we open, a new clone ticket is needed (sometimes it is necessary to open quite a few consoles at the same time).

Seems complicated, but it is much easier to understand by doing while following the VMware VIX information.

```
#get the Sessionmanager object
$thissession = Get-View Sessionmanager

#get the clone ticket
$sessionticket =  $thissession.AcquireCloneTicket()
```

```
PS C:\temp> $thissession = Get-View Sessionmanager


PS C:\temp> $sessionticket =  $thissession.AcquireCloneTicket()



PS C:\temp> $sessionticket
cst-VCT-52829a06-ada6-3315-16ee-772b5b167ba5--tp-3D-74-F1-B5-66-B6-BF-29-DE-E4-3E-52-CD-D6-4E-0F-
35-29-ED-76


PS C:\temp> $sessionticket =  $thissession.AcquireCloneTicket()


PS C:\temp> $sessionticket
cst-VCT-52f69d0b-b0ba-3044-fa8c-f8230dbe877a--tp-3D-74-F1-B5-66-B6-BF-29-DE-E4-3E-52-CD-D6-4E-0F-
35-29-ED-76


PS C:\temp>
```

Just one more thing before assembling everything together and attempting to run VMRC again: You will need to have a straightforward way for getting the vCenter name than just keeping this information in a variable and passing it around since you first connected to it. This is easily achieved using PowerCLI, which has a system variable called $DefaultVIServer. Running it and selecting the Name property gives the name of the vCenter in most cases.

```
PS C:\temp> $DefaultVIServer


Name                           Port  User
----                           ----  ----
10.9.74.5                      443   PROD\tpopescu
```

```
PS C:\temp> $DefaultVIServer.name
10.9.74.5
```

However, things get more complicated. This name is not always the correct one. (This would be when the display name differs from the host name.) Like in many other cases, PowerCLI has a very personal way of doing things. To make allowance for possible issues, more in depth digging is needed.

Going through the exposed properties shows that there is a property named “ServiceUri”.

```
PS C:\temp> $DefaultVIServer | fl


IsConnected   : True
Id            : /VIServer=prod\tpopescu@10.9.74.5:443/
ServiceUri    : https://10.9.74.5/sdk
SessionSecret : 52dae7f3-6802-6944-ace6-607556da4d17
Name          : 10.9.74.5
Port          : 443
SessionId     : 52dae7f3-6802-6944-ace6-607556da4d17
User          : PROD\tpopescu
Uid           : /VIServer=prod\tpopescu@10.9.74.5:443/
Version       : 5.5
Build         : 3142196
ProductLine   : vpx
InstanceUuid  : 71b48822-8bca-4298-87ef-1a3479c5bed9
RefCount      : 1
ExtensionData : VMware.Vim.ServiceInstance
Client        : VMware.VimAutomation.ViCore.Impl.V1.VimClient
```

Accessing the ServiceUri property exposes a number of other properties. The one of interest is “Host”.

```
$DefaultVIServer.serviceuri


AbsolutePath   : /sdk
AbsoluteUri    : https://10.9.74.5/sdk
LocalPath      : /sdk
Authority      : 10.9.74.5
HostNameType   : IPv4
IsDefaultPort  : True
IsFile         : False
IsLoopback     : False
PathAndQuery   : /sdk
Segments       : {/, sdk}
IsUnc          : False
Host           : 10.9.74.5
Port           : 443
Query          :
```

```
Fragment       :
Scheme         : https
OriginalString : https://10.9.74.5/sdk
DnsSafeHost    : 10.9.74.5
IdnHost        : 10.9.74.5
IsAbsoluteUri  : True
UserEscaped    : False
UserInfo       :
```

Selecting the Host property yields a name that can be trusted.

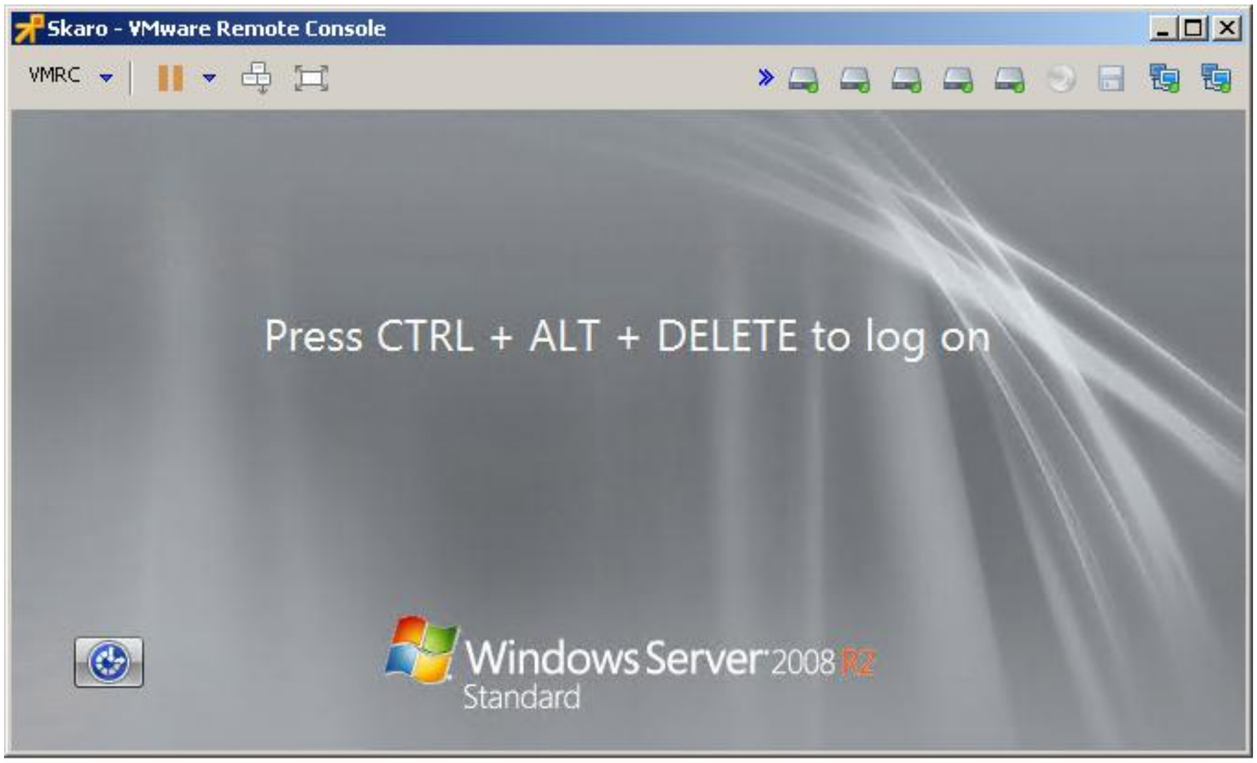
```
PS C:\temp> $DefaultVIServer.ServiceUri.Host
10.9.74.5

PS C:\temp>
```

Now you have everything that is needed to connect to the Skaro VM's desktop.

The code looks like this:

```
#get the moref id of the virtual machine
$vm = get-vm -Name "Skaro"
$vmmoref =  ($vm).ExtensionData.moref.value

#get the vCenter name or IP address
$vcentername = $DefaultVIServer.serviceuri.Host

#get the Sessionmanager object
$thissession = Get-View -Id Sessionmanager

#get the clone ticket
$sessionticket =  $thissession.AcquireCloneTicket()

#start the vmrc console
Start-Process -FilePath "C:\Program Files (x86)\VMware\VMware Remote Console\vmrc.exe" -
ArgumentList "vmrc://clone:$($sessionticket)@$($vcentername)/?moid=$($vmmoref)"}
```

Running it takes us straight to the VM desktop. No vCenter login is needed.


Skaro - VMware Remote Console
VMRC
Press CTRL + ALT + DELETE to log on
Windows Server 2008 R2
Standard

To wrap up, everything can be saved in a function called Open-VMconsole, which takes the ID of a VM and opens the desktop console for that machine.

```
function Open-VMConsole {
param($vmid)
if(!($vmid)){Write-Host " -vmid parameter is null. Exiting"; return}

try {
#get the moref id of the virtual machine
$vm = get-vm -Id $vmid
$vmmoref =  ($vm).ExtensionData.moref.value

#get the vCenter name or IP address
$vcentername = $DefaultVIServer.serviceuri.Host

#get the Sessionmanager object
$thissession = Get-View -Id Sessionmanager

#get the clone ticket
$sessionticket =  $thissession.AcquireCloneTicket()

#start the vmrc console
Start-Process -FilePath "C:\Program Files (x86)\VMware\VMware Remote Console\vmrc.exe" -
ArgumentList "vmrc://clone:$($sessionticket)@$($vcentername)/?moid=$($vmmoref)"}
catch {write-host $ErrorMessage}
}
```

Now, the last block is needed to lay the foundation of the omnibus script that has been discussed. This last one differs from the others in that it can't be embedded in a function. It needs to reside in the main body of the script, although it can call various functions, if needed.

This block performs the following functions:

1. Determines, at run time, the name and location of the script source code. In other words, no matter how the script is renamed and from what folder it is run, this information is available at run time.
2. Loads the body of the script source code as a single string. Since the script source code is already loaded into memory at run time, there is no file lock in place.
3. Identifies the location of the connection information and replaces it with the new/modified one.
4. Saves the modified script source code, replaces the old one, then exits execution. If the script is run again, the new source code is used.

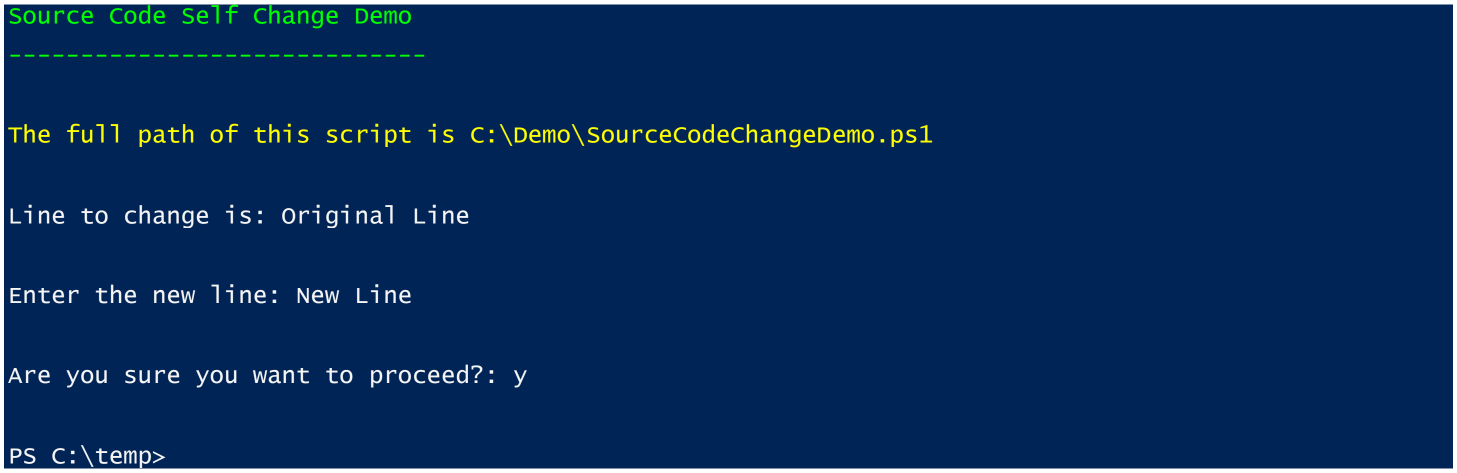
Since these operations in the omnibus script are very complex, I shall use a simple example:

Let's assume that the script to change path is C:\Demo\SourceCodeChangeDemo.ps1.

The code is shown below:

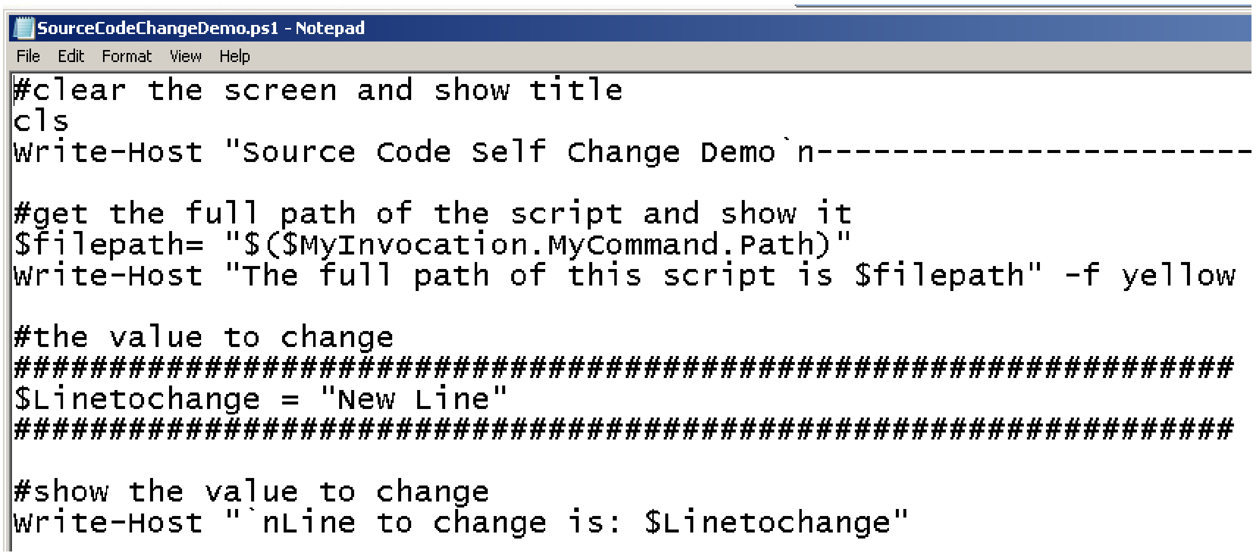
```
#clear the screen and show title
cls
Write-Host "Source Code Self Change Demo`n-----------------------------`n" -f green

#get the full path of the script and show it
$filepath= "$($MyInvocation.MyCommand.Path)"
Write-Host "The full path of this script is $filepath" -f yellow

#the value to change
################################################################
$Linetochange = "Original Line"
################################################################

#show the value to change
Write-Host "`nLine to change is: $Linetochange"

#enter the new value
$line = Read-Host "`nEnter the new line"
#OK to proceed
do {
$confirm = Read-Host "`nAre you sure you want to proceed?"
}until("y","n" -contains $confirm)
if($confirm -eq "y"){

#read the source code
$sourcecode = get-content $filepath

#modify the source code by replacing the old value with the new one and replacing everything in the
source code variable
$sourcecode=$sourcecode.replace($linetochange,$line)
#saving the modifed sourcecode
$sourcecode | out-file -FilePath $filepath

#open the file to check if the change was successful and exit
notepad $filepath

}
```

Running the script shows the following:



```
Source Code Self Change Demo
-----------------------------

The full path of this script is C:\Demo\SourceCodeChangeDemo.ps1

Line to change is: Original Line

Enter the new line: New Line

Are you sure you want to proceed?: y

PS C:\temp>
```

And the result is shown below.



SourceCodeChangeDemo.ps1 - Notepad
File Edit Format View Help

```
#clear the screen and show title
cls
Write-Host "Source Code Self Change Demo`n----------------------

#get the full path of the script and show it
$filepath= "$($MyInvocation.MyCommand.Path)"
Write-Host "The full path of this script is $filepath" -f yellow

#the value to change
################################################################
$Linetochange = "New Line"
################################################################

#show the value to change
Write-Host "`nLine to change is: $Linetochange"
```

Now that all the foundation blocks have been laid out, and you can begin writing the various VMware Rapid Recovery related snippets that simplify the overall administrative effort.

In the next installment of the series, I will discuss how to retrieve the relevant Rapid Recovery-related VM information for easy inspection and how to create sets of VMs that will be processed at the same time (for instance enabling CBT or changing UUID for virtual disks), as well as show you the underlying code.