Hidden Network: Detecting Hidden Networks created with USB Devices

Pablo González Pérez: pablo@11paths.com
Francisco José Ramírez Vicente: franciscojose.ramirezvicente@telefonica.com

Executive Summary

Many companies and government agencies today have communications isolated networks or with data flow restricted through different networks. These computers networks are created for particular situations, as these can be very special or have critical information such as, factory control system, highly-secured environments for processing of certain data, or networks complying with a safety standard. In recent Cyber-security history, it has been proven how this malicious software called *Stuxnet* infiltrated into an isolated network at a Nuclear Power Plant. In light of this fact, it could be seen that having a computers network not connected through Ethernet cable or *WiFi* to other networks is not enough. Any type of external connection for computers may constitute a threat. This paper reflects the possibilities provided by the so-called *Hidden Network* and how these can be identified and focused on protection of these issues inside a corporate network.

1. The risks of the connections

Regarding the safety of data networks, there is a trend to sketch networks through the connections at links level, such as *Ethernet, WiFi* connections, etc. Corporate networks are much more complex than this and they require analysis from different perspectives.

The traffic analysis on a corporate data network is the major tool provided to understand what is going on with it. In addition, it is the more consistent option for searching network nodes seizing the most used protocols, which are nodes conformed by the most essential services or those who act as bottlenecks. Given these statements, it can be extrapolated that a risks-based network analysis provides a great deal of data and information based on different guidelines.

Mapping a network in an aesthetically manner is right for understanding the nodes and network settings, therefore, a telemetry analysis is critical to contain and be prepared for such hazards. Due to these deal of data, we can achieve a greater level of understanding regarding the network and the threats inside it.

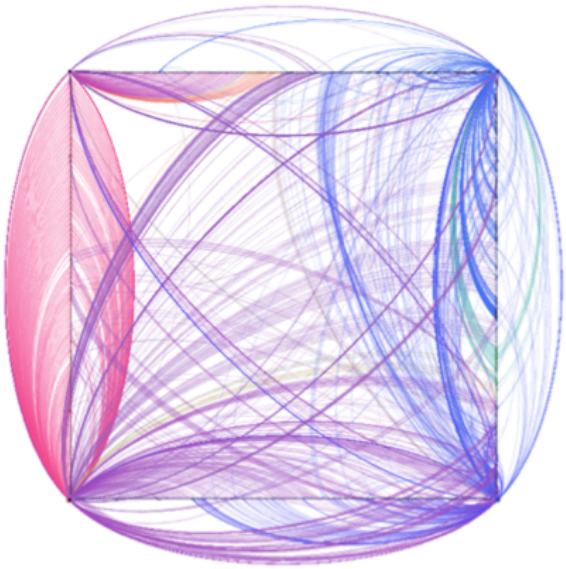

Figure 1: A network map simulation with different nodes and connections between them

Nodes representation and connection structures are vital to comprehend all different borders inside the network in order to mitigate intrusions, detecting attacks, or preventively carry out the implementation of safety measures.

The problem is more a case of understanding that this is a network. In many cases, a network is defined as a group of computers connected with the possibility to communicate with each other across different technologies and protocols. On most occasions, users or system and network administrators consider a good thing having a network connected through *Ethernet* or *WiFi* connection on different organization computers. This is not the case here, as one organization not implementing prevention measures regarding the use of USB devices, may enforce what is known as *Hidden*

Networks. These networks are created through the use of USB devices and allow communication between physically or logically isolated computers.

1.1. Network Isolation and USB Connection

To understand the hazards about *Hidden Networks*, which are created from USB devices, here is a simple example. Assuming an organization have a network formed by 3 types of VLANs. The first VLAN contains:

- A computer called A. This computer has connectivity with others computers of the same VLAN.
- A computer called B. This computer has connectivity with others computers of the same VLAN.

The second VLAN contains:

- A computer called C. This computer has connectivity with others computers of the same VLAN.
- A computer called D. This computer has connectivity with others computers of the same VLAN.
- A computer called E. This computer has connectivity with others computers of the same VLAN.

The third VLAN contains:

- A computer called F. This computer has connectivity with others computers of the same VLAN.

If we observe the networks outline on the image below, we can see how computers are isolated through different VLANs. Assuming employees of this organization exchange data by means of USB devices, there is a high probability for this information to pass from one VLAN's computer to another. Adding the USB device is itself a source of threats, a hidden network is being created inside the organization.

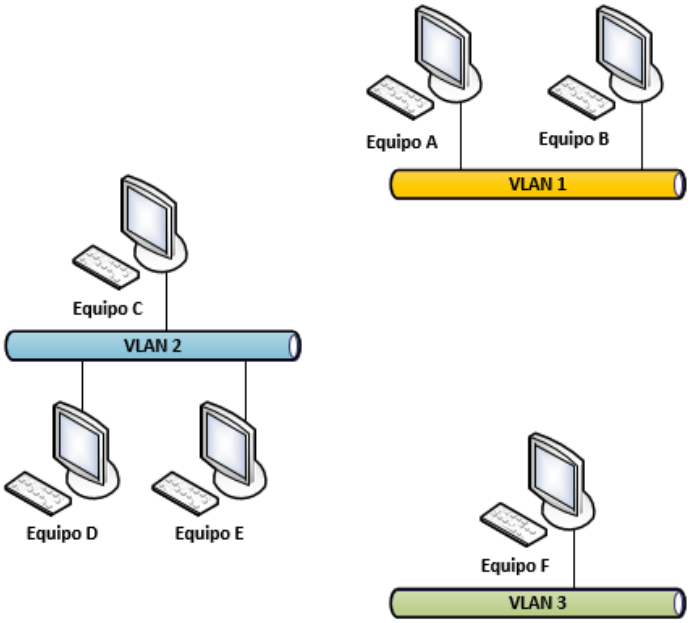

Figure 2: Network outline with computers connected to different VLANs

Assuming that users of computers F and E are exchanging information through a USB device, a hidden network is being created between both computers. This information can be represented with two nodes, E and F , and an arc projected between the computer first introduced the USB device and the second computer.

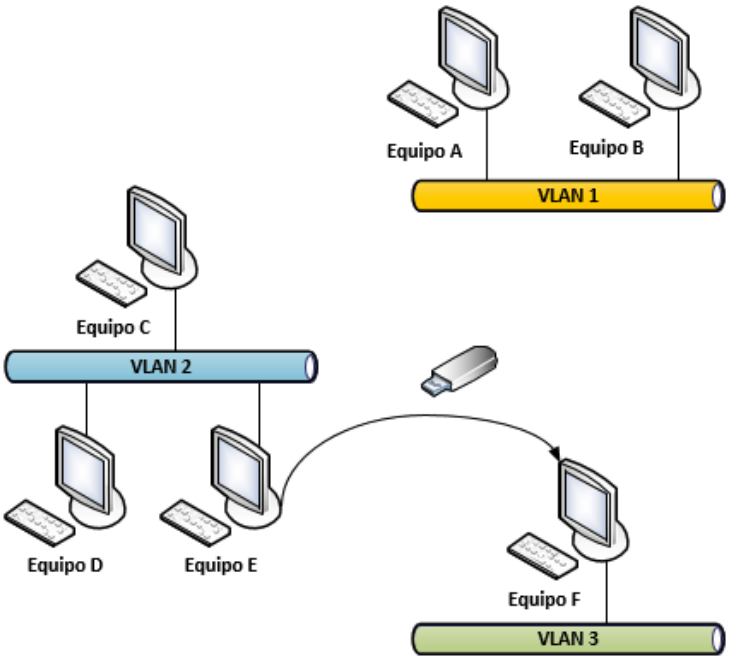

Figure 3: Hidden network overlapping on the previous network outline

2. USB Device Connections in the Computer System

When a USB device is connected from one computer to another, the term "*Pollination*" emerge. This concept is similar to the one used on other areas and is related with carrying the threat or risk from one USB device among different computers, even when these are connected to different networks.

When a user connects a USB device, a series of entries are created in the system of the Windows registry. This type of information is valuable, for example, in a forensic analysis, in order to know where the information leakage comes from or where did the threat enter in a *Post-Mortem.*

The *USBStor* key created in the Windows system registry save information regarding all different devices inserted in the computer. If USB N devices were inserted in a N computer, such N devices will be found at the *USBStor* key, which contains all the information needed to identify the device.

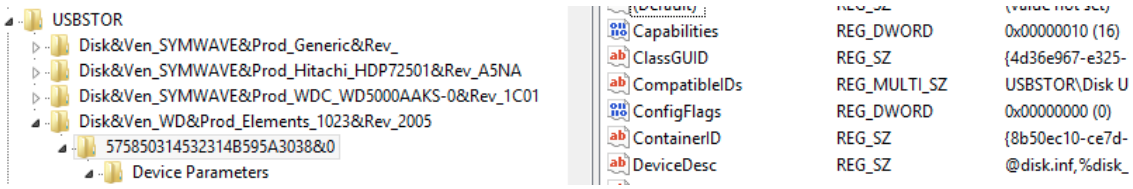

Figure 4: Registry visualization of the USB devices inserted

The following information can be collected from USB devices connected to one computer:

- Device name.
- *Class*.
- *ClassGUID*.
- *HardwareID*.
- Service provided by the device, e.g. a hard disk.
- *Driver*.
- *Etc.*

2.1. Hidden Links: Detection of These Kind of Networks

By knowing where and how the information of one USB device is stored within a *Microsoft* operative system, you could know who is sharing the USB device, and with whom. In this way, we can generate two nodes representing two computers and one arc which identifies the connection between both computers. A hidden network is being uncovered thanks to the *Hidden Link.* Besides, you can identify in which computer was connected a priori, as a result of the many events that can be obtained from an operative system. This is how the arc between nodes is directed.

To perform an automation of the *Hidden Links* detection, the following plan has been proposed:

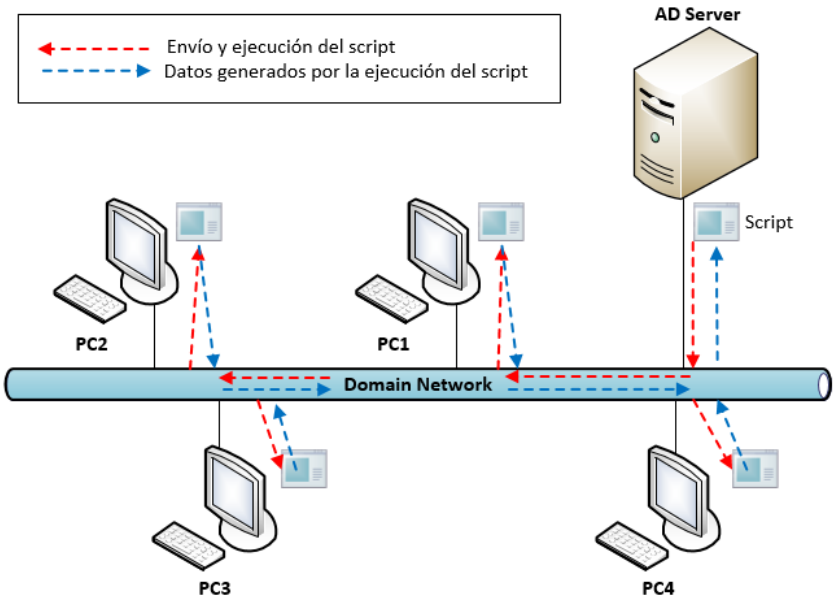

Figure 5: Script release diagram in an AD (Active Directory)

In the above image, it can be seen how the application is executed in a central node and how this is able to use several *Microsoft* technologies to execute commands in each of the computers within the domain. The examined technologies, which fit the solution design are as follows:

- *WinRM*.
- *SMB (Server Message Block)*.
- *WMI*.

Powershell is an object-oriented command line from *Microsoft,* which has a simple and powerful interaction with any structure inside a *Microsoft* operative system.

```
PS C:\> Get-ItemProperty -Path HKLM:\SYSTEM\CurrentControlSet\Enum\USBSTOR\*\*
        | Select FriendlyName
FriendlyName
------------
SanDisk U3 Cruzer Micro USB Device
WD Virtual CD 1110 USB Device
USB DISK 2.0 USB Device
USB DISK 2.0 USB Device
ADATA USB Flash Drive USB Device
Corsair Voyager USB Device
FLASH Drive AU_USB20 USB Device
hp USB Flash Drive USB Device
Kingston DT 101 G2 USB Device
Kingston DT 101 G2 USB Device
SanDisk U3 Cruzer Micro USB Device
USB Flash Disk USB Device
WD 3200BEV External USB Device
WD My Book 1110 USB Device
WDC WD25 00JB-00GVA0 USB Device
```

Figure 6: Collection of USB devices connected to Powershell

2.2. USB Hidden Networks for WinRM

The script *WinRM* version in *PowerShell* requires the activation of the *Windows Remote Management (WinRM)* service in each of the network computers to be audited:

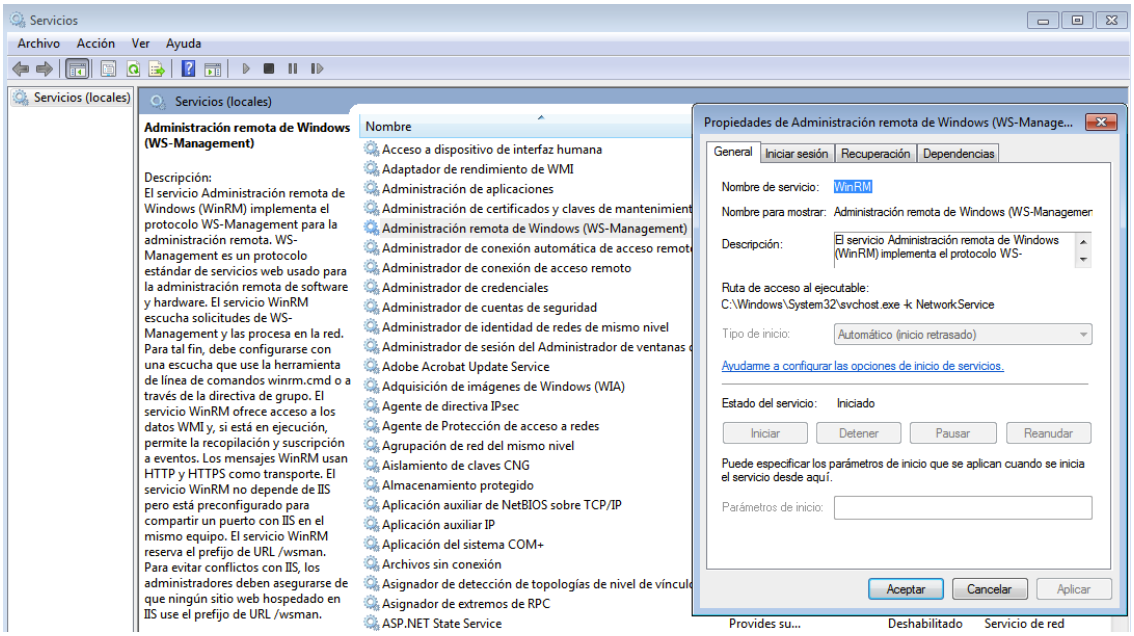

Figure 7: WinRM Service

Furthermore, the script has been tested on a single-*Domain* network with an *Active Directory (AD)* to automate the information collection as much as possible. Domain administrator credentials are used to approve execution on remote computers in the local network. Credentials will be required when executing the script.

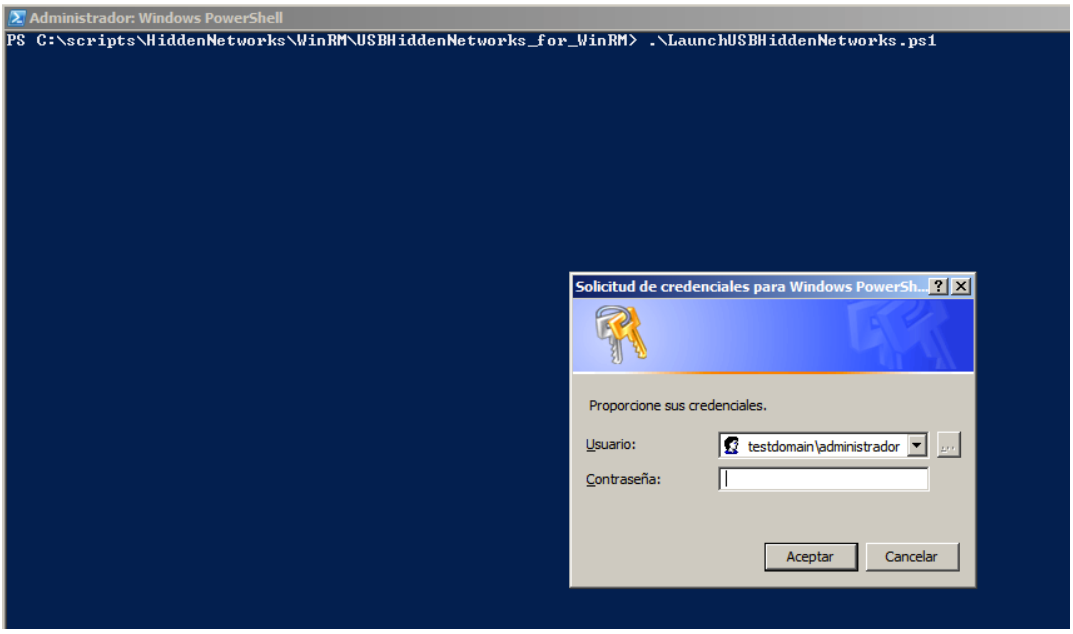

Figure 8: Credentials requirement regarding the script usage

The *script* primary implementation is performed through the "*LaunchUSBHiddenNetworks.ps1"* program, which connects remote computers using the script known as "*RecollectUSB.ps1",* which is passed as parameter to collect information from USB devices. Thus, the script shall be executed individually in each of the computers assigned.

2.2.1. Script: LaunchUSBHiddenNetworks

The execution of this command is based on the *PowerShell*'s command "*Invoke-Command".* This command allows to connect with a computer in the network passing the *FQDN*, computer name or IP address as parameters, and on the other hand, the script of *PoweShell* to be executed:

```
$salida=invoke-command -ComputerName (Get-Content servers.txt) -FilePath
'PathToScript\RecollectUSBData.ps1'-Credential testdomain\administrador
```

With the *–ComputerName* parameter, the name of computer(s) to be audited is assigned inside our AD. It is possible to directly introduce the name of computers followed by commas, but in this case, a TXT (*servers.txt)* file with a list of the computers has been used and passed as parameter.

The *-FilePath* parameter assign the path for the script on *PowerShell* that will be performing the data collection. Finally, the *–Credential* parameter allows the use of domain administrator credentials to approve the execution of remote computer, in which case, domain is *"testdomain"* and user *"administrator".*

The outcome of the run is stored in the object *$salida.* The information retrieved will be stored likewise in a *CSV* file called "*USBDATA.csv"* as follows:

```
$salida | Out-File USBDATA.csv
```

The formatting of *CSV* file has the following structure after running the *script*:

Name of the computer, IP (on IPv4 format), USB name, ID (unique identifier)

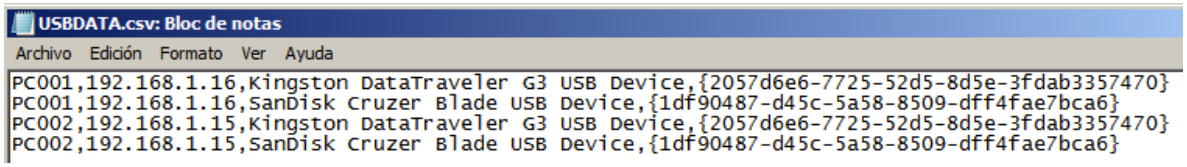

Figure 9: Results obtained in CSV format

With this information, we could create a graph like the one shown below with the *Gephi* application:

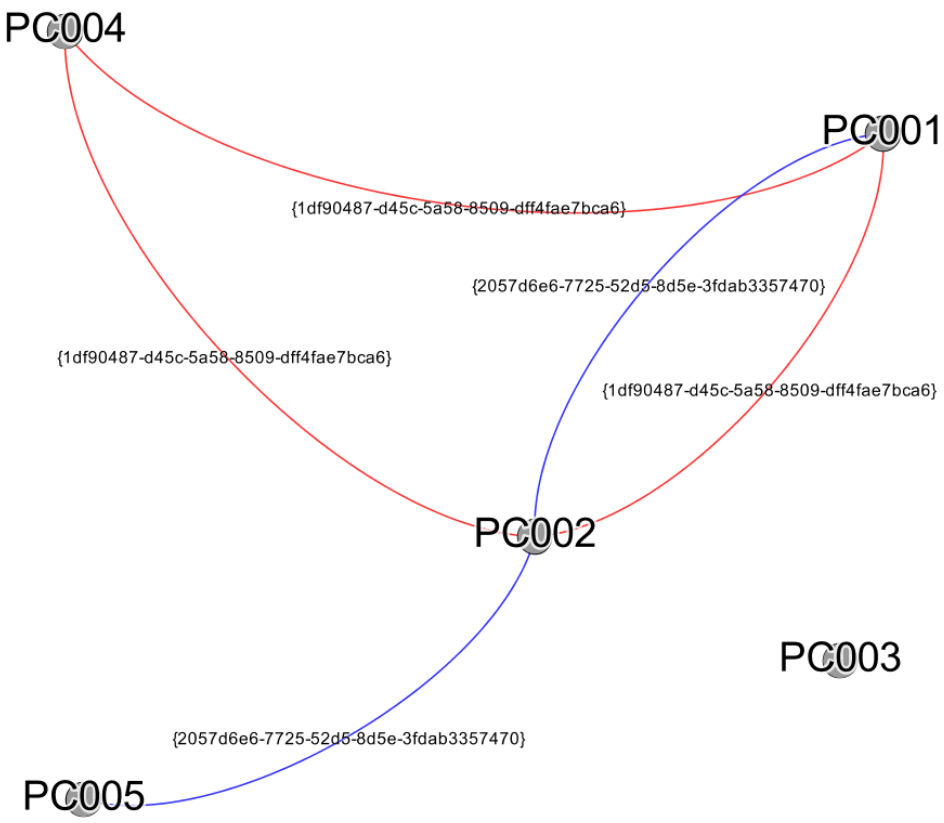

Figure 9: Graph representing hidden connections of USB devices in a network

2.2.2. Script: RecollectUSBData

This script is responsible for gathering all information referring the USB devices connected to the computer and it runs locally in the computers to be audited. The data is retrieved from a particular branch of the Windows registry.

```
$USBDevices = @()
$USBContainerID = @()
$USBComputerName = @()
$USBComputerIP = @()
$SubKeys2        = @()
$USBSTORSubKeys1 = @()
```

The matrixes, where information related to the audited computer is going to be stored, and for the data referred to USB devices stored in the registry or that were connected at some point in time to the computer, are launched.

```
$Hive    = "LocalMachine"
$Key     = "SYSTEM\CurrentControlSet\Enum\USBSTOR"
```

$Hive and *$Key* store the complete path for the registry branch where the data search related to USB devices is taking place. The variable *$Hive with "LocalMachine"* value equals to *HKLM* or *HKEY_LOCAL_MACHINE*.

```
$ComputerName = $Env:COMPUTERNAME
$ComputerIP = $localIpAddress=((ipconfig | findstr [0-9].\.)[0]).Split()[-1]
```

The name of local computer is stored, as well as the IP address and variables *$ComputerName* and *$ComputerIP*.

```
$Reg = [Microsoft.Win32.RegistryKey]::OpenRemoteBaseKey($Hive,$Computer)
$USBSTORKey = $Reg.OpenSubKey($Key)
$nop=$false
```

On *$Reg* object, the registry query is run using the command *OpenRemoteBaseKey*, using variables *$Hive* y *$Computer* as parameters, which establish the branch to be consulted. The variable *$nop* will be used later to control the execution flow.

```
Try {
    $USBSTORSubKeys1  = $USBSTORKey.GetSubKeyNames()
}
Catch
{
    Write-Host "Computer: ",$ComputerName -foregroundcolor "white" -
backgroundcolor "red"
    Write-Host "No USB data found"
    $nop=$true
}
```

The *Try – Cath* block is responsible for managing errors in case no information regarding USB devices can be found. If no information is found, the value *$true* is assigned to the *$nop* variable in order to avoid execution of the whole process of identification and retrieving of USB device data.

```
if(-Not $nop)
```

In case there is any entry associated with the USB device connection, *$nop* a *$true* variable, the following blocks will be run:

Block 1:

```
ForEach($SubKey1 in $USBSTORSubKeys1)
    {
        $Key2 = "SYSTEM\CurrentControlSet\Enum\USBSTOR\$SubKey1"
        $RegSubKey2 = $Reg.OpenSubKey($Key2)
        $SubkeyName2 = $RegSubKey2.GetSubKeyNames()
        $Subkeys2  += "$Key2\$SubKeyName2"
        $RegSubKey2.Close()

    }
```

Each of the existing items in the registry branch where the search is taking place is a different USB device. Each item is stored in the matrix *@Subkeys2*.

Block 2:

```
ForEach($Subkey2 in $Subkeys2)
    {
        $USBKey = $Reg.OpenSubKey($Subkey2)
        $USBDevice = $USBKey.GetValue('FriendlyName')
```

```
        $USBContainerID = $USBKey.GetValue('ContainerID')

        If($USBDevice)
        {
            $USBDevices += New-Object -TypeName PSObject -Property @{
            USBDevice = $USBDevice
            USBContainerID = $USBContainerID
            USBComputerName= $ComputerName
            ComputerIP = $ComputerIP
            }
        }

        $USBKey.Close()
    }
```

This block explore every USB device previously identified in the Block 1 and stored at the *@Subkeys2* matrix. For every item having a value at the *$USBDevice* field, the USB device ID is being retrieved; *USBContainerID*. The name and IP address of the computer it is also assigned in order to add this later to the output *CSV* file.

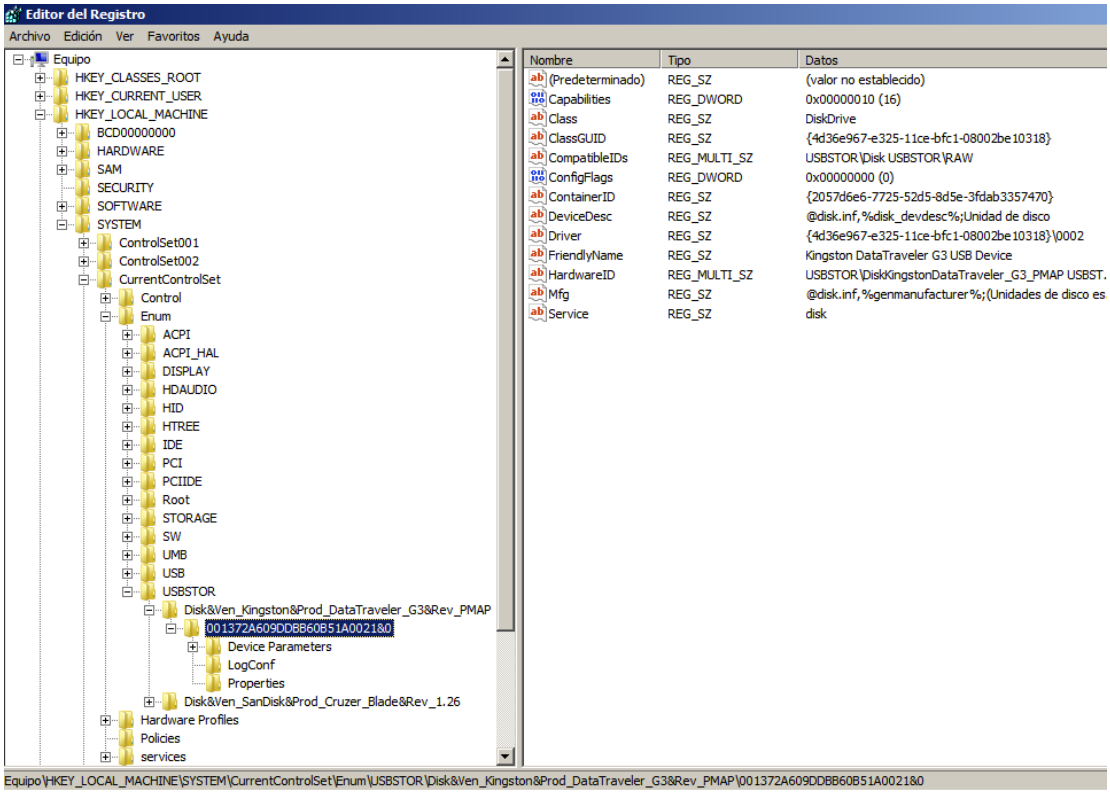

Figure 10: Registry branch where the USB devices are

Block 3:

```
for ($i=0; $i -lt $USBDevices.length; $i++) {
        $IDUnico=$USBDevices[$i] | Select -ExpandProperty "USBContainerID"
        $USBNombre=$USBDevices[$i] | Select -ExpandProperty "USBDevice"
        Write-Host "Computer: ",$ComputerName -foregroundcolor "black" -
backgroundcolor "green"
        Write-Host "IP: ",$ComputerIP
```

```
        Write-Host "USB found: ",$USBNombre
        Write-Host "USB ID: ",$IDUnico
        Echo "$ComputerName,$ComputerIP,$USBNombre,$IDUnico"
    }
```

Finally, this block displays pertinent information obtained from the remote computer. The *Write-Host* print command is used on the server screen, where the script was run. The *Echo* command is used as data output to subsequently write the data in the *CSV* file.

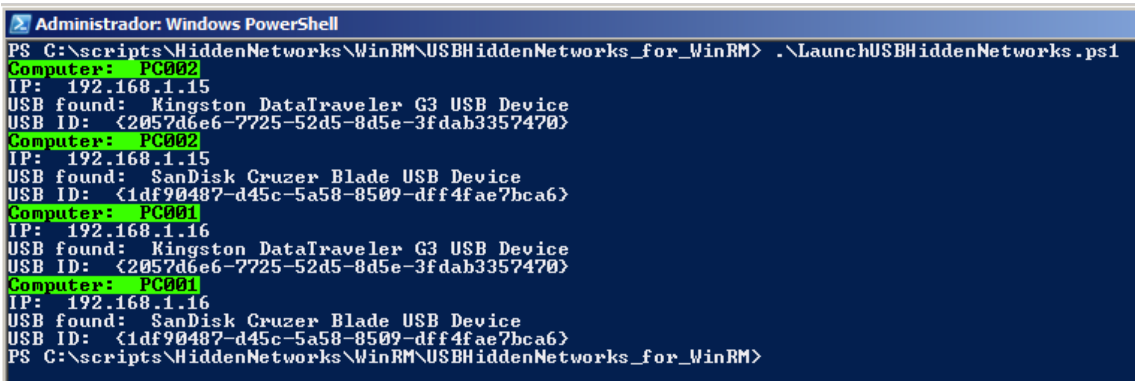

Figure 11: Output after the script running

2.3. USB Hidden Networks for SMB con PSExec

In order to run the script through SMB, it will be necessary to have *PSTools* previously installed, specifically to execute the PSExec command in the computers to be checked. The operating philosophy will be practically the same of the WinRM version. It will be connected from the server to the remote computer and the script should be run from the server with domain administrator account, then the USB data collection script will be executed.

The main script *LaunchUSBHiddenNetworks.ps1* will have a few modifications to fit with this new type of connection. The primary modification is that this time, the command *Invoke-Command* is not used to remotely run the script. A shell from Powershell will now be opened, and the script should be run from it. The script will be downloaded from the network location, preferably from a web server that would execute download through some HTTP protocol. In this way, subsequent problems with execution policy and permits, which you may find by accessing the local shared resource, are avoided.

Similar to the previous version of WinRM, results should be stored in a CSV file. To avoid synchronization problems and allow time for the program to run on the remote computer, some delays have been included as described in the code analysis below:

2.3.1. Script: LaunchUSBHiddenNetworks

```
$computers = gc
"C:\scripts\HiddenNetworks\PSExec\USBHiddenNetworks_for_SMB\servers.txt"
$url = "http://192.168.1.14/test/RecollectUSBData.ps1"
$sincro = 40
```

Several variables are assigned. The matrix where the server names or IP addresses will be stored, *$computers*, which are at the *servers.txt* file, the *$url* variable showing where the script RecollectUSBData.ps1 is and, ultimately, the waiting time to sync the operation. It should also be taken into consideration that this number may vary depending on the environment where the script is run. The following is an example of execution:

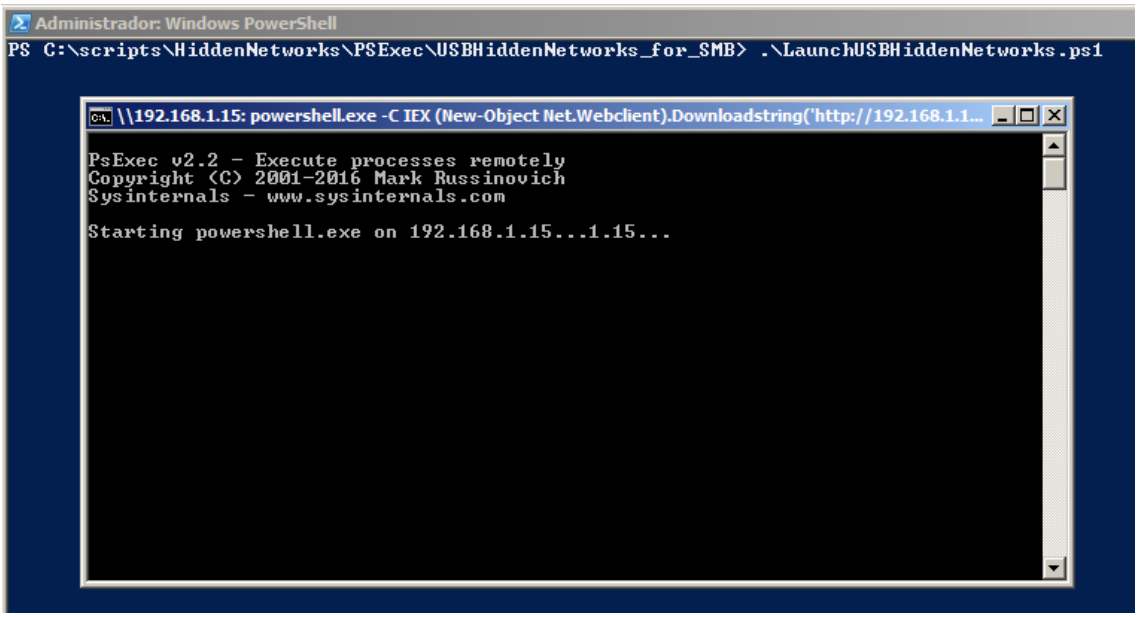

Figure 12: Powershell and script running through the PSEXEC tool.

The *servers.txt* file will have the computers names or directly the IP addresses stored as we can see below:

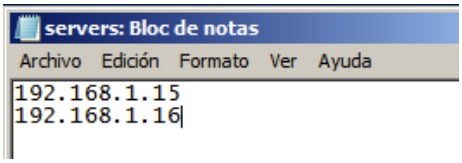

Figure 13: List of computers to be analyzed

```
foreach ($computer in $computers) {
      $Process = [Diagnostics.Process]::Start("cmd.exe","/c psexec.exe
                 \\$computer powershell.exe -C IEX (New-Object
                 Net.Webclient).Downloadstring('$url') >>
                 C:\scripts\HiddenNetworks\PSExec\USBHiddenNetworks_for_SMB\
                 usbdata.csv")
       $id = $Process.Id
       sleep $sincro
```

```
        Write-Host "Process created. Process id is $id"
        taskkill.exe /PID $id
}
```

In this loop, each of the computers to be analyzed is checked, which also have been loaded with *$computers* variables from the *servers.txt* file. The execution main body is focused in the object *$Process.* In it, a remote computer console is opened, which in turn will launch other *Powershell* console, passing the file RecollectUSBData.ps1 as parameter, which is at the location designated by the *$url* variable. Is critical to have properly configured the location paths for each of the files before running the script.

Before moving on to the following computer in the list, it will be necessary to be sure about termination of the information collection process. There are several manners to optimize this operation, but in this illustration the choice is simply adding an X seconds delay between each running by means of the *sleep* command. Once the data collection of the computer to be audited is terminated, we erase the execution process before moving on to the next with *taskkill* command. For information purposes, the ID and result of this operation is screen-printed, as pictured in the following snapshot:

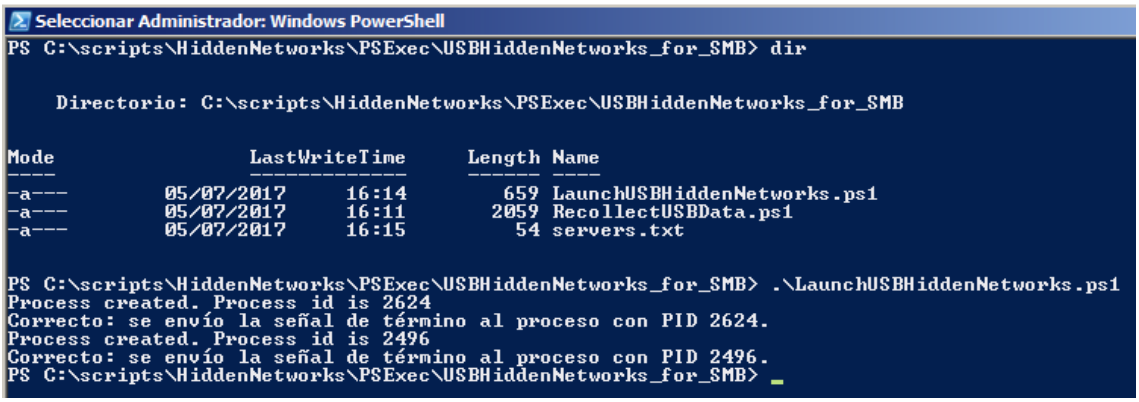

Figure 14: Script running in Powershell

2.3.2. Script: RecollectUSBData

This script has only been modified at the last block (Block 3) in order to adapt the output to the new execution type. As can be seen in the code shown below, command *Echo* has been replaced for a *Write-Host* with variables, eliminating the screen output:

```
    for ($i=0; $i -lt $USBDevices.length; $i++) {
        $IDUnico=$USBDevices[$i] | Select -ExpandProperty "USBContainerID"
        $USBNombre=$USBDevices[$i] | Select -ExpandProperty "USBDevice"
        Write-Host "$ComputerName,$ComputerIP,$USBNombre,$IDUnico"
    }
```

The generated USBData.CSV file will be exactly the same as the one previously shown.

2.4. Historical Information

It is also possible to get the registration of the dates of first connection in the computer, in case we need more information regarding the route of the USB device

inside the *HiddenNetwork.* At the event log, the branch capable of offering more information is disabled by default in all Windows versions. Such branch is as follows:

Windows Logs -> Applications and Services Logs -> Windows -> DriverFrameworks-UserMode -> Operational

Thus, the way we obtain the first connection date of the USB device in the computer, without accessing the computer assessment, is by analyzing the following file on the system:

C:\Windows\inf\setuoapi.dev.log

Within this file, the time of first connection, among other data, has been recorded. To properly locate the USB device inserted, it will be necessary to store a new value while running the "*RecollectUSBData.ps1*" script, and this field value would be *DiskID*:

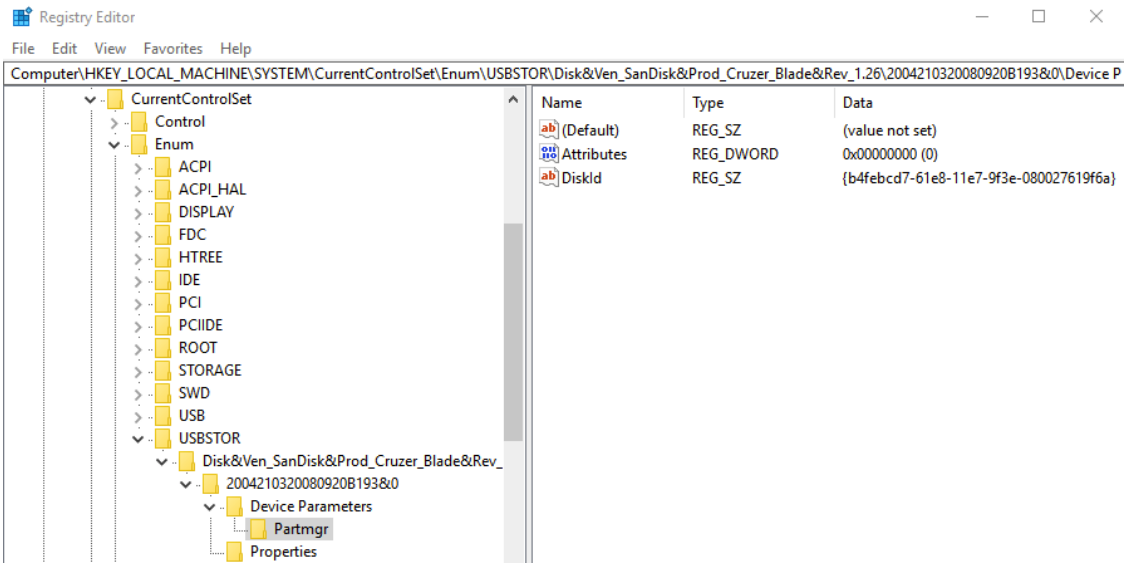

Figure 15: DiskID key search

This value is unique within the current Windows system, but it changes when connected to another computer, unlike *ContainerId* field, which is the same on each of the Windows computers. The USB can be identified inside the *setupoapi.dev.log* file with this value. In the illustration below, its location can be shown by using the *DiskID* and the date of first insertion within the audited system:

```
264       dvi: Device post-install completed. 18:16:50.685
265  <<<  Section end 2017/07/05 18:16:50.794
266  <<<  [Exit status: SUCCESS]
267
268
269  >>>  [Device Install (Hardware initiated) - SWD\WPDBUSENUM\{b4febcd7-61e8-11e7-9f3e-080027619f6a}#0000000000004000]
270  >>>  Section start 2017/07/05 18:41:28.029
271       dvi: {Build Driver List} 18:41:28.076
272       dvi:      Searching for compatible ID(s):
273       dvi:           wpdbusenum\fs
274       dvi:           swd\generic
275       dvi:      Created Driver Node:
276       dvi:           HardwareID    - wpdbusenum\fs
277       dvi:           InfName       - C:\Windows\System32\DriverStore\FileRepository\wpdfs.inf_amd64_e898714e5623f0fe\
278       dvi:           DevDesc       - WPD FileSystem Volume Driver
279       dvi:           Section       - Basic_Install
280       dvi:           Rank          - 0x00ff2000
281       dvi:           Signer Score  - INBOX
282       dvi:           DrvDate       - 06/21/2006
283       dvi:           Version       - 10.0.15063.0
284       dvi: {Build Driver List - exit(0x00000000)} 18:41:28.122
285       dvi: {DIF_SELECTBESTCOMPATDRV} 18:41:28.122
286       dvi:      Using exported function 'WpdClassInstaller' in module 'C:\Windows\system32\wpd_ci.dll'.
287       dvi:      Class installer == wpd_ci.dll,WpdClassInstaller
288       dvi:      Class installer: Enter 18:41:28.170
289       dvi:      Class installer: Exit
290       dvi:      Default installer: Enter 18:41:28.170
291       dvi:           {Select Best Driver}
292       dvi:                Class GUID of device changed to: {eec5ad98-8080-425f-922a-dabf3de3f69a}.
293       dvi:                {DIF_DESTROYPRIVATEDATA} 18:41:28.170
294       dvi:                     Class installer: Enter 18:41:28.170
295       dvi:                     Class installer: Exit
296       dvi:                     Default installer: Enter 18:41:28.170
```

Figure 16: Obtaining the connection date of the USB device

2.5. Hidden Links in OS X

On computer systems running *Mac OS X* or *macOS,* theses have a file with a *PLIST* extension, which store this information over the USB devices connected to the computer. The file is named *com.apple.finder.plist*. On image below, an example of information capture in *OS X* or *macOS* environments can be seen.

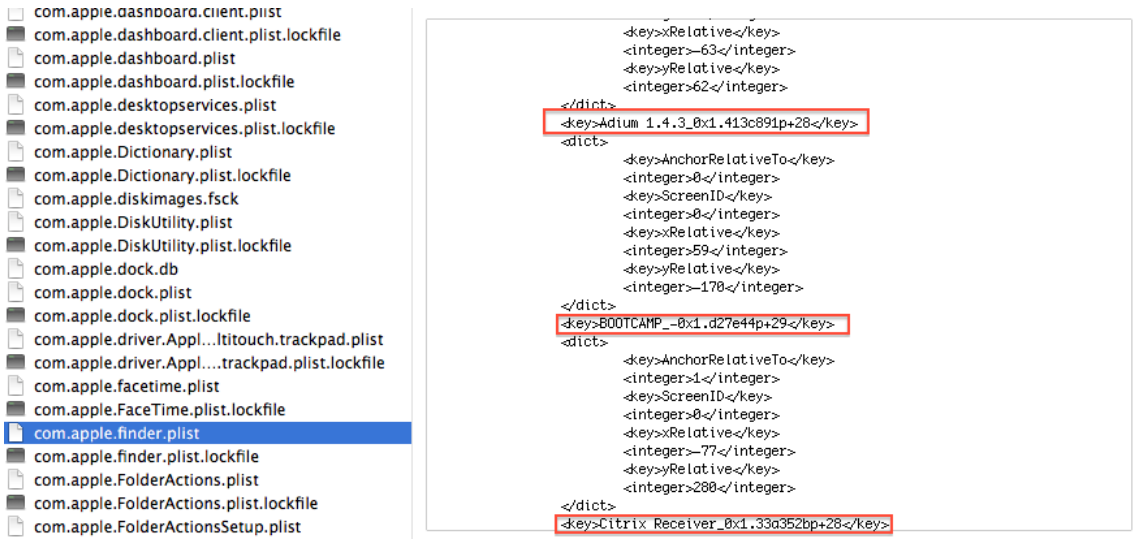

Figure 17: Data collection in USB devices connected to OS X systems

2.6. Mitigation

One way to prevent this "*Pollination"* between computers in a corporate network is to restrict the use of USB devices in computers. Mitigation or prevention through the forced use due to *Active Directory* policies, which restrict connection in a user computer to devices only approved by one user. The implementation of the safety policy along with a white list of approved devices for each user allows to avoid this kind of *Hidden Links*, but it is complex and expensive to maintain.

Figure 18: The use of the USB device is forbidden on this computer due to company policy

3. Conclusions

Beyond the possible leakage of corporate information, a *Hidden Network* is also a problem for our computer's integrity. Such USB devices may spread a malware towards different sections within the infrastructure, where, in theory, the security is higher. Having networks disconnected from the Internet provides this false security feeling regarding a higher level of protection before any incident, which increases the system vulnerability.

The malware infection through USB devices is a real and underlying problem, not only regarding the historic *Stuxnet* case, but with others of greatest relevance such as *Brutal Kangaroo* used by the CIA.

Due to the major impact such infections and information leakage may have in our infrastructure, we have created this paper to help in identifying these hidden networks and to offer a tool for their control. Thus, it will be easier to prevent incidents and also to provide a utility containing useful information for forensic analysis cases.

References

https://blogs.technet.microsoft.com/heyscriptingguy/2012/05/18/use-powershell-to-find-the-history-of-usb-flash-drive-usage/

http://www.elladodelmal.com/2017/06/brutal-kangaroo-y-la-infeccion-por-usb.html

https://github.com/ElevenPaths/USBHiddenNetworks