Institut Universitaire de Technologie,
Aix-Marseille Université

RAPPORT DE STAGE
Diplôme Universitaire de Technologie
Spécialité Réseaux et Télécommunications

Améliorer le logiciel de supervision

Gianluca CORBO

Anyware Video

Responsable entreprise : Xavier PICAT

Responsable académique : Ivan MADJAROV

2022

Table des matières

1 Introduction

Ce stage se déroule dans une très petite entreprise, nommée Anyware Video. Je travaillais dans le service développement et recherche. Mon stage consistait à améliorer le logiciel de supervision des machines de la société.

Un logiciel de supervision permet le développement et/ou l'exploitation d'une application de supervision. Une application de supervision est avant tout un outil de conduite d'une installation automatisée. Il permet également l'acquisition de données, leur archivage et leur consultation.

Mon employeur, qui est aussi mon responsable de stage, est chef et créateur de l'entreprise Anyware Video. Monsieur PICAT était intéressé par mes compétences afin de développer en PowerShell* un script client permettant d'envoyer les données nécessaires à la supervision des machines. Cela n'empêchait pas d'avoir parfois d'autre tâches à réaliser.

Dans un premier temps, je vais vous présenter l'entreprise, ses activités, ses clients, sa culture et son organisation. Dans un second temps, je vais étudier le cadre technique général du sujet. Dans un troisième temps, je parlerai du travail que j'ai réalisé durant ce stage.

2 Anyware Video

Figure 1 : Logo de l'entreprise

2.1 Ses activités

Anyware Video est une société développant, depuis 1998, des solutions numériques pour ses clients. Ces dernières sont CastGenie, DubMaster et Pige Antenne.

DubMaster est un logiciel qui permet aux opérateurs d'effectuer de multiples traitements sur des vidéos qu'ils souhaiteraient diffuser, notamment sur la qualité de la vidéo et l'ajout d'effets.

Pige Antenne est un outil utile pour surveiller la diffusion, enregistrer celle-ci et permettre des vérifications pour s'assurer de l'absence de problèmes et/ou que toutes les obligations légales liées aux partenaires publicitaires sont bien respectées.

OneGenie regroupe l'ensemble du savoir-faire d'Anyware Video, dont les solutions DubMaster et Pige Antenne permettent d'offrir à ses clients une solution complète pour répondre à leurs besoins liés à la diffusion de médias (figure 2).

Figure 2 : Capture d'écran de OneGenie

2.2 Ses clients

Anyware video vend des logiciels permettant la diffusion. Ses clients sont donc des chaînes de télévisions. Il y a par exemple TF1, FranceTélévisions, Mediawan … (figure 3), mais l'éventail ne se limite pas à la France. Son offre s'étant au groupe canadien Canadian Broadcast Corporation (figure 3) et à d'autres chaînes australiennes, japonaises, américaines, camerounaises.

Lorsqu'un client est intéressé par ces technologies, il est possible de prendre rendez-vous sur le site, afin de faire une démonstration. Les clients vont parfois avoir des demandes très précises auxquelles l'entreprise va répondre.

Figure 3 : Logo de chaînes de télévision

2.3 Sa culture

Anyware Video a sa propre personnalité, c'est-à-dire qu'elle possède des habitudes, des valeurs, des pratiques… qui lui sont propres. La première valeur que j'ai pu remarquer, c'est l'ouverture d'esprit ; les employés ne portent aucun jugement et il est très facile de discuter avec eux et de s'intégrer à l'équipe.

De plus, les locaux sont situés à Marseille mais la majorité des employés n'y habitent pas donc le télétravail est une pratique courante dans l'entreprise. J'ai moi-même télétravaillé plusieurs fois au cours du stage. De ce fait, on a l'habitude de parler dans un groupe Teams*, c'est là où l'on va se dire bonjour, au revoir, bon appétit etc.

Tous les matins, il y a une réunion sur Teams pour faire le point sur ce qui a été fait et ce qui doit être fait. En effet, tous les soirs avant de partir il faut envoyer un compte rendu de notre journée au patron.

Dans les locaux, il y a une cuisine dans laquelle nous nous retrouvons souvent pour manger ensemble. C'est le patron qui généralement attribue les tâches à effectuer. Même si la hiérarchie est respectée, il est possible de discuter avec lui pour lui exposer une nouvelle idée.

2.4 Son organisation

L'entreprise est généralement composée d'environ 8 ou 9 membres. Mais ce chiffre peut varier car elle accueille souvent des stagiaires ou des alternants.

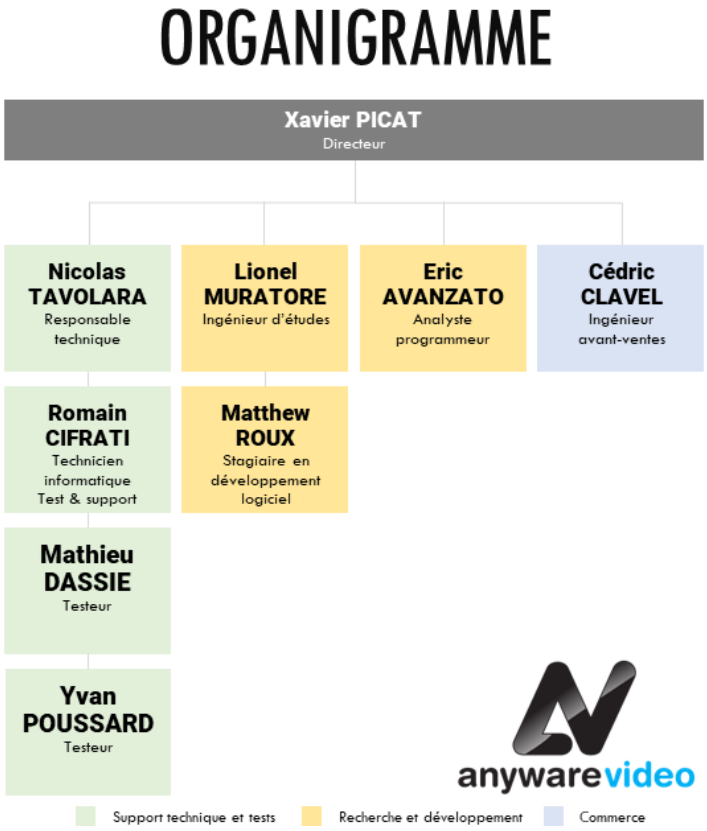

Figure 4 : Organigramme de l'entreprise

Lors de mon stage, je faisais partie du service Recherche et Développement (figure 4). J'ai communiqué principalement avec mon maître de stage et un employé qui m'a beaucoup aidé. Etant donné que mon maître de stage est le chef de l'entreprise il manquait parfois de temps pour répondre à mes questions. J'ai aussi discuté avec le directeur technique pour mettre en place mon travail.

3 Objectifs du stage

3.1 Objectifs professionnels

Comme vu précédemment, l'objectif principal de mon stage était d'améliorer le logiciel de supervision. Pour cela il fallait en premier lieu comprendre comment il fonctionnait. Cette première version du logiciel avait été développée par un étudiant stagiaire en école d'ingénieur il y a 8 mois.

Ma mission étant de réécrire le script côté client en PowerShell, je devais analyser le code écrit précédemment en Python*. Une fois cette étape passée, j'ai pu commencer à développer le script client en PowerShell. Ce script doit envoyer des données récoltées sur le PC et les envoyer au serveur. Lorsque ces données sont bien reçues, le serveur les affiche sur un site internet accessible qu'en interne. Je devais pour finir faire une documentation pour expliquer comment fonctionne mon script et où se situaient mes erreurs.

Les données à récolter étant :

- La date d'envoi des données
- Les adresses IPs
- Les adresses MACs
- Le nom du système d'exploitation
- Le nom du processeur
- Le nombre de cœur du processeur
- Le nom et la version des drivers du processeur graphique
- Le nom des disques, leur taille maximale et restante
- Le bon fonctionnement de l'antivirus et sa version
- La liste des processus appartenant à Anywware Video
- Le chemin d'accès des logs
- La mémoire vive utilisée
- La mémoire vive totale
- Le compte Windows actuellement utilisé
- Le nom de l'ordinateur

Ma deuxième mission était de répondre au besoin de l'entreprise. Je devais écrire un script PowerShell sur la demande d'un client avec des consignes bien précises. Le but de ce script était d'analyser des fichiers Extensible Markup Language ou XML* pour ensuite déplacer des fichiers vidéo dans différents dossiers selon le contenu du fichier XML. J'ai ensuite commencé l'écriture d'un script qui supprime tous les fichiers et dossiers d'un répertoire sauf la dernière version de l'installeur.

3.2 Objectifs personnels

Mes objectifs durant ce stage étaient de réussir à m'intégrer dans une entreprise. En effet, ce stage était pour moi le premier contact avec le monde professionnel, et je voulais que tout se passe bien. Je souhaitais aussi produire du travail qui servirait à l'entreprise. Je cherchais aussi à me créer un réseau professionnel. Ce stage était aussi important pour déterminer si le domaine de l'informatique et particulièrement la programmation était pour moi un domaine où je pourrais travailler tout au long de ma carrière.

4 Réalisation

Mon stage a d'abord commencé par la lecture de la documentation administrative, tels que la charte informatique, le règlement intérieur, les préventions incendies etc.
Dès lors cette lecture terminée, j'ai pu débuter ma mission principale.

4.1 Analyse du code Python

Pour commencer, j'ai lu la documentation du logiciel mais elle était incomplète, il manquait des informations essentielles à la compréhension du programme. J'ai donc commencé à effectuer mes premiers tests. Ces derniers étaient concluants mais je ne comprenais pas comment cela fonctionnait.

Après ces tests, j'ai analysé les différents scripts Python qui formaient le logiciel. À la suite de cette lecture, j'en ai conclu que certains scripts n'allaient pas me servir et que j'utiliserai seulement deux d'entre eux : AVVerifierServer.py et AVVerifierService.py. Le premier est le serveur qui reçoit les données puis les affiche sur une page internet et le deuxième est le client qui récupère et qui envoie les données au serveur.

L'envoi des données est basé sur une Application Programming Interface ou API. N'ayant jamais étudié ce procédé, j'ai dû effectuer des recherches pour pouvoir continuer ma compréhension du programme. L'API est une solution informatique qui permet à des applications de communiquer entre elles et de s'échanger mutuellement des services ou des données.

Dans mon cas, l'envoi de données se faisait avec le format JavaScript Objet Notation. Le JSON est un langage léger d'échange de données textuelles. Pour les ordinateurs, ce format se génère et s'analyse facilement. Pour les humains, il est pratique à écrire et à lire grâce à une syntaxe simple et à une structure en arborescence.

J'ai ensuite posé des questions à un collègue qui était le maître de stage de l'étudiant ayant développé ce logiciel. Il m'a permis de comprendre comment fonctionnaient les scripts et il m'a conseillé d'afficher sur la console du serveur les données que j'envoyais. Ce conseil m'a permis de voir précisément les données à envoyer.

4.2 Connexion au serveur

J'ai débuté en effectuant des recherches pour savoir comment envoyer une requête JSON via PowerShell, je n'y trouvais que des mauvais exemples. J'ai essayé d'écrire une première version du script en m'inspirant de ce que j'avais pu voir mais sans succès. J'ai donc décidé de demander de l'aide à un collègue et il m'a expliqué où se situaient mes erreurs. Cela m'a permis de produire un début de script permettant de se connecter au serveur mais sans envoyer encore aucune donnée (figure 5).

```
$Body = @{
    success= $true
    pcname= ""
    pcips= ""
    macs= ""
    win_user= ""
    date= ""
    memory= ""
    log_path= ""
    proc= @()
    state= 1
    eset= ""
    disk= ""
    name_cpu= ""
    name_os= ""
    memory_total= ""
    cpu_heart= ""
    gpu= ""
    user= ""
    reason= ""
}
$JsonBody = $Body | ConvertTo-Json
$Params = @{
    Method = "Post"
    Uri = "http://localhost:6123/setPCData"
    Body = $JsonBody
    ContentType = "application/json"
}
$Response = Invoke-RestMethod @Params
$Response
```

Figure 5 : Script se connectant au serveur

Le Body est la partie où l'on met les données à envoyer. Il s'agit d'un point très important car le nom doit correspondre à celui qui est écrit dans le serveur, sinon, le serveur ne va pas comprendre les données et ne s'aura pas comment les interpréter. Il faut convertir ces données en format JSON pour que le serveur puisse traiter ces informations.

Il faut ensuite envoyer la requête au serveur hébergé sur ma machine. Pour ce faire, on va entrer l'adresse du site et c'est la raison pour laquelle il y a écrit localhost. Pour envoyer ces données on utilise la méthode POST*.

4.3 Récupération des informations

Il était nécessaire d'effectuer toutes les fonctions de récupération, les fonctions de la figure 6 et 7 n'ont pas posé de problème. Cela m'a pris du temps d'effectuer des recherches sur les commandes adéquates et de les tester pour parvenir à renvoyer toutes les informations dans le bon format pour que le serveur puisse les traiter. En faisant ces fonctions cela m'a fait prendre conscience de la puissance du langage PowerShell. Pour construire ces fonctions, j'ai utilisé des boucles (for) et des conditions (if). Ces fonctions ne sont pas très complexes, le meilleur moyen pour les comprendre est de les tester.

```
function AVGet_Date{
Get-Date -Format "yyyy-MM-ddTHH:mm:ss:fff"
}

function AVGet_IP{
$IP = (Get-NetIPAddress -AddressFamily IPV4).IPAddress
for ($i = 0; $i -lt $IP.length; $i++){
    if ($IP[$i] -ne "127.0.0.1"){
        $TIP +=  ,$IP[$i]
    }
}
return $TIP
}

function AVGet_MAC{
(Get-NetAdapter).MacAddress
}

function AVGet_OS_Name{
$name = ((Get-CimInstance -ClassName Win32_OperatingSystem).Name).Split("|")
$name[0]
}

function AVGet_CPU_Name{
(Get-CimInstance -ClassName Win32_Processor).Name
}

function AVGet_Memory_Total{
[math]::Round((((Get-WmiObject -Class WIN32_OperatingSystem).TotalVisibleMemorySize)/1Gb),2)
}

function AVGet_Memory{
$var =  Get-WmiObject -Class WIN32_OperatingSystem
[math]::Round((($var.TotalVisibleMemorySize - $var.FreePhysicalMemory)/1Gb),2)
}

function AVGet_GPU_Name{
$msg = "Name: " + (Get-CimInstance -ClassName Win32_VideoController).Name + ", Driver: " + (Get-CimInstance -ClassName Win32_VideoController).DriverVersion
return $msg
}
```

Figure 6 : Fonctions de récupération 1

```
function AVGet_CPU_Cores{
(Get-CimInstance -ClassName Win32_Processor).NumberOfCores
}

function AVGet_Log_Path{
Get-ItemPropertyValue -Path "HKLM:\SOFTWARE\Anyware Video" -Name "LogDirectory"
}

function AVGet_Eset{
$process = (Get-Process).ProcessName
$version = (Get-Item 'C:\program files\ESET\ESET Security\eguiProxy.exe').VersionInfo.FileVersion
if ("eguiProxy" -and "ekrn" -and "ERAAgent" -in $process){
    $Bool = "True"
    }
else{
    $Bool = "False"
    }
$message = $Bool + " - Version : " + $version
return $message

}

function AVGet_Proc{
$proc = @((Get-Process).ProcessName)
$AVlist = @('AllInOne','AllInOneMediaManager','IngenieRecorder','AllInOneMediaReader',
            'AVMediaTranscoder','DMPTest','AMPTest','CastGenie','AVSupervisor',
            'AVController','CastEdit','DUBMaster','PigeEncoder','PigeManager')
$AVproc = @()
for ($i = 0; $i -lt ($proc.Length); $i++){
    if ($proc[$i] -in $AVlist){
        if ($proc[$i] -notin $AVproc){
            $AVproc += $proc[$i]
        }
    }
}
return $AVproc
}
```

Figure 7 : Fonctions de récupération 2

Après cette étape, les deux fonctions à développer (figure 8 et 9) ont nécessité beaucoup de temps et de tests avant d'obtenir un résultat satisfaisant.

La fonction AVGet_Ini (figure 8) sert à lire un fichier Ini* se situant dans le même répertoire que le script. Pour cela, il va récupérer chaque ligne du fichier puis les découper pour en sortir l'information qui nous intéresse. Dans notre cas, les informations étaient le port et l'adresse IP du serveur, le nom de l'utilisateur et la raison pour laquelle il utilise la machine. Mon maître de stage m'a demandé d'ajouter à cette fonction des valeurs par défaut dans le cas où le fichier ini serait absent ou qu'il y manquerait des informations, le but étant de pouvoir changer facilement ces valeurs par défaut.

```
#IMPORTANT IL FAUT QUE LES CLES/VALEUR SOIT COLLE DANS LE FICHIER INI EXEMPLE : url=192.168.20.199
function AVGet_Ini{
$Param = @{port="6123";url="192.168.20.199";user="Gianluca";reason="Test"}
try{
    $Location = Get-Location
    $Ini = Get-Content "AVVerifierService.ini" -Force -ErrorAction Stop
    $IniHash = @{}
    $IniTemp = @()
    ForEach ($Line in $Ini){
        If ($Line -ne "" -and $Line.StartsWith("[") -ne $True){
            $IniTemp += $Line
        }
    }
    ForEach($Line in $IniTemp){
        $SplitArray = $Line.Split("=")
        $IniHash += @{$SplitArray[0] = $SplitArray[1]}
    }
    if ($IniHash.port -or $IniHash.url -or $IniHash.user -or $IniHash.reason -eq ""){
        Write-Host "Une valeur est manquante dans le fichier AVVerifierService.ini, les valeurs par défauts vont être utilisés."
        $IniHash = $Param
        $Str_IniHash = $IniHash | Out-String
        Write-Host $Str_IniHash
    }
}
catch{
    $IniHash = $Param
}
return $IniHash
}
$Ini = AvGet_Ini
```

Figure 8 : Fonctions de récupération 3

La fonction AVGet_Disk (figure 9) a été la plus longue à développer. La première version de la fonction fonctionnait très bien sur mon PC mais lorsque j'ai demandé à l'un de mes collègues de la tester sur sa machine plus rien ne fonctionnait. Dans un premier temps, je ne traitais pas les partitions car sur ma machine il n'y en avait pas. Dans un second temps, j'ai traité les partitions mais l'un de mes disques avait la particularité de ne pas s'afficher dans le récapitulatif. J'ai mis plusieurs jours avant de parvenir à ce résultat qui permet de récupérer les informations de tous les disques partitionnés ou non et de les afficher dans le bon format. Le seul bémol de la fonction est que les disques partitionnés ne sont pas regroupés.

```
function AVGet_Disk{
$Disks_Letter = (Get-Volume).DriveLetter
$Disk_Letter_Partition = (Get-Partition).DriveLetter
$Disks_Name = (Get-PhysicalDisk).FriendlyName
$Message = ""
$i = 0
$Disks_Name
Foreach($Disk_Letter in $Disks_Letter){
    if($Disk_Letter -ne $null){
        $Disk_Type = (Get-Volume -DriveLetter "$Disk_Letter").DriveType
        if($Disk_Type -ne "CD-ROM"){
            if($Message -ne ""){$Message += " - "}
            if ($Disk_Letter -in $Disk_Letter_Partition){
                $Partitions = Get-Partition -DriveLetter "$Disk_Letter"
                $Disk_Number = $Partitions.DiskNumber
                $Disk_Space = [math]::round((Get-Volume -DriveLetter "$Disk_Letter").Size/1Gb,2)
                $Disk_Space_Remaning = [math]::round((Get-Volume -DriveLetter "$Disk_Letter").SizeRemaining/1Gb,2)
                $Disk_Name = (Get-PhysicalDisk -DeviceNumber "$Disk_Number").FriendlyName
                $Message += $Disk_Name + " (" + $Disk_Letter + ":\ Total: " + $Disk_Space + " Go; Free: " + $Disk_Space_Remaning + " Go)"
            }
            else{
                $Disk_Name = $Disks_Name[$i]
                $Disk_Space = [math]::round((Get-Volume -DriveLetter "$Disk_Letter").Size/1Gb,2)
                $Disk_Space_Remaning = [math]::round((Get-Volume -DriveLetter "$Disk_Letter").SizeRemaining/1Gb,2)
                $Message += $Disk_Name + " (" + $Disk_Letter + ":\ Total: " + $Disk_Space + " Go; Free: " + $Disk_Space_Remaning + " Go)"
            }
        }
    }
    $i += 1
}
return $Message
}
```

Figure 9 : Fonctions de récupération 4

Cette partie du script est la même que la figure 5 mais j'y ai ajouté les données récoltées grâce aux fonctions. De plus, j'ai fait en sorte que l'adresse d'envoi des données devienne adaptable en fonction des informations présentes dans le fichier ini ou dans les valeurs par défaut.

```
$Body = @{
    success= $true
    pcname= $env:computername
    pcips= AVGet_IP
    macs= AVGet_MAC
    win_user= $env:UserName
    date= AVGet_Date
    memory= AVGet_Memory
    log_path= AVGet_Log_Path
    proc= AVGet_Proc
    state= 1
    eset= AVGet_Eset
    disk= AVGet_Disk
    name_cpu= AVGet_CPU_Name
    name_os= AVGet_OS_Name
    memory_total= AVGet_Memory_Total
    cpu_heart= AVGet_CPU_Cores
    gpu= AVGet_GPU_Name
    user= $Ini.user
    reason= $Ini.reason
}
$JsonBody = $Body | ConvertTo-Json
$Params = @{
     Method = "Post"
     Uri = "http://" + $Ini.url + ":" + $Ini.port + "/setPCData"
     Body = $JsonBody
     ContentType = "application/json"
}
$Response = Invoke-RestMethod @Params
$Response
```

Figure 10 : Envoi des données

Dorénavant, toutes les données récoltées sont envoyées au serveur (figure 11) et on peut retrouver ces informations sur le site à l'adresse « (adresse IP du serveur):(le port)/AV_Monitoring »(figure 12).

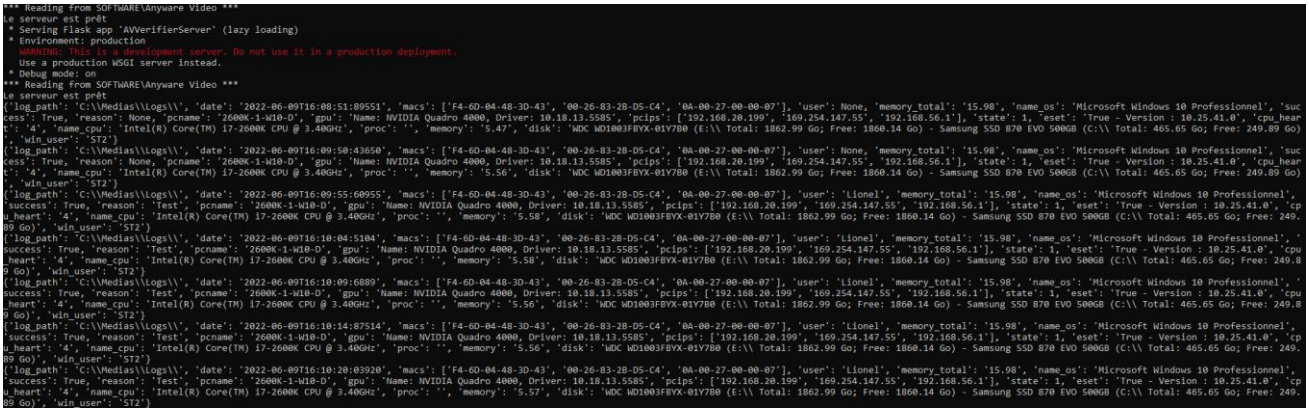

Figure 11 : Données reçues par le serveur

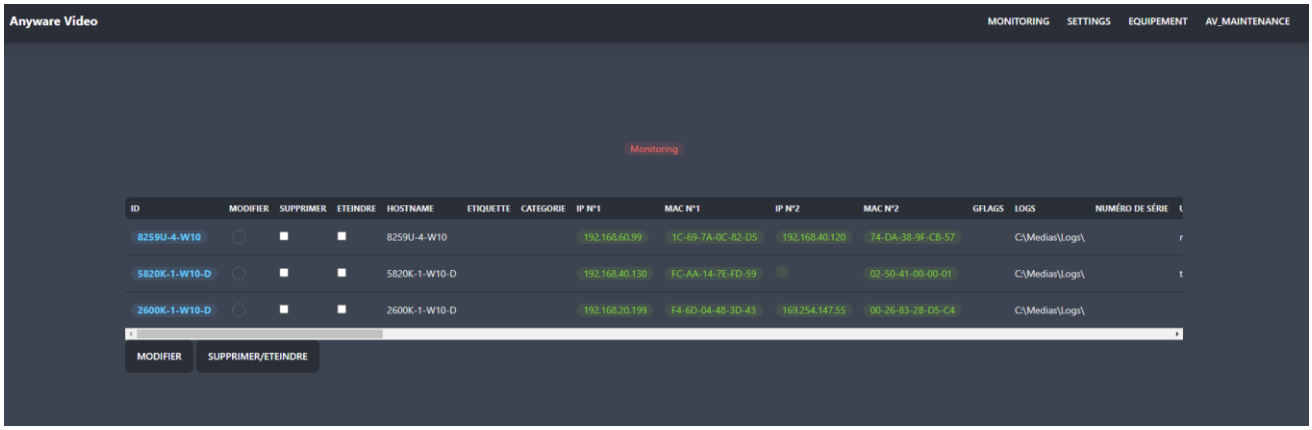

Figure 12 : Affichage des données sur un site

4.4 Passage aux classes et boucle infinie

Le passage aux classes (figure 13 et 14) de mon script a été compliqué car dans un premier temps je ne connaissais pas la syntaxe des classes PowerShell et cela ne ressemblait pas du tout à ce que j'avais pu étudier au cours de l'année. Dans un second temps, je suis resté bloqué sur une erreur de type, c'est-à-dire qu'au lieu de mettre « [Hashtable] » j'avais écrit « [String] ».

```
class CRecup{
    [String]$m_strMAC
    [String]$m_strDate
    [String]$m_strIP
    [String]$m_strCPU_Name
    [String]$m_strOs_Name
    [String]$m_strMemory
    [String]$m_strMemory_Total
    [String]$m_strGPU_Name
    [String]$m_strCPU_Cores
    [String]$m_strDisk
    [String]$m_strLog_Path
    [String]$m_strEset
    [String]$m_strProc
    [Hashtable]$m_Ini

    [void]Update()
    {
        $this.m_strIP = $this.AVGet_IP()
        $this.m_strMAC = $this.AVGet_MAC()
        $this.m_strDate = $this.AVGet_Date()
        $this.m_strCPU_Name = $this.AVGet_CPU_Name()
        $this.m_strOs_Name = $this.AVGet_OS_Name()
        $this.m_strMemory = $this.AVGet_Memory()
        $this.m_strMemory_Total = $this.AVGet_Memory_Total()
        $this.m_strGPU_Name = $this.AVGet_GPU_Name()
        $this.m_strCPU_Cores = $this.AVGet_CPU_Cores()
        $this.m_strDisk = $this.AVGet_Disk()
        $this.m_strLog_Path = $this.AVGet_Log_Path()
        $this.m_strEset = $this.AVGet_Eset()
        $this.m_strProc = $this.AVGet_Proc()
        $this.m_Ini = $this.AVGet_Ini()
    }
```

Figure 13 : Syntaxe des classes

Il convient de retenir que le but de ce script est d'être lancé lorsqu'une machine s'allume et d'envoyer continuellement des informations au serveur sur son état. Il faut donc faire une boucle infinie et pour cela j'ai utilisé la condition « Tant que Vrai ». La commande « start-sleep -seconds 5 » permet de relancer la boucle toutes les 5 secondes.

```
    [String]AVGet_IP(){...}
    [String]AVGet_MAC(){...}
    [String]AVGet_Date(){...}
    [String]AVGet_CPU_Name(){...}
    [String]AVGet_OS_Name(){...}
    [String]AVGet_Memory(){...}
    [String]AVGet_Memory_Total(){...}
    [String]AVGet_GPU_Name(){...}
    [String]AVGet_CPU_Cores(){...}
    [String]AVGet_Disk(){...}
    [String]AVGet_Log_Path(){...}
    [String]AVGet_Eset(){...}
    [String]AVGet_Proc(){...}
    [Hashtable]AVGet_Ini(){...}
}

$CR = [CRecup]::new()
$CR.Update()
while ($True ){
    $CR.Update()
    $Body = @{...}
    $JsonBody = $Body | ConvertTo-Json
    $Params = @{...}
    Write-host $Param.Uri
    $Response  = Invoke-RestMethod @Params
    $Response
    start-sleep -seconds 5
```

Figure 14 : Boucle infinie

4.5 Ajout de la dialogue box et des logs

Les logs sont indispensables pour suivre l'envoi de données, ils permettent de savoir si les données envoyées sont les bonnes et si le serveur les a bien reçues. En premier lieu, j'ai créé un fichier dans le répertoire des logs qui a pour nom la date de création et l'utilisateur de la machine à ce moment-là (figure 15). En second lieu, j'ai affiché sur la console toutes les données envoyées et la réponse du serveur, puis j'ai ajouté ces mêmes informations au fichier texte créé précédemment (figure 16 et 17).

```
$Global:Fichier_Log = New-Item -Path "C:\Medias\Logs\AVSet_PC_Data\Log-$(Get-Date -Format "
dd-MM HH-mm-ss")-$($Global:DialogueBox.User).txt" -ItemType File
```

Figure 15 : Création d'un fichier texte Log

```
    Write-Host "
----------------------------------------------------

Log $(Get-Date -Format "dd/MM HH:mm:ss")>$($JsonBody)"

    Write-host "
Log $(Get-Date -Format "dd/MM HH:mm:ss")<$($Response)

----------------------------------------------------"

    if($Global:Compteur -gt 0){
        Add-Content -Path $Global:Fichier_Log -Value "
        ----------------------------------------------------

        Log $(Get-Date -Format "dd/MM HH:mm:ss")>$($JsonBody)

        Log $(Get-Date -Format "dd/MM HH:mm:ss")<$($Response)

        ----------------------------------------------------"
    }
```

Figure 16 : Affichage et ajout des données

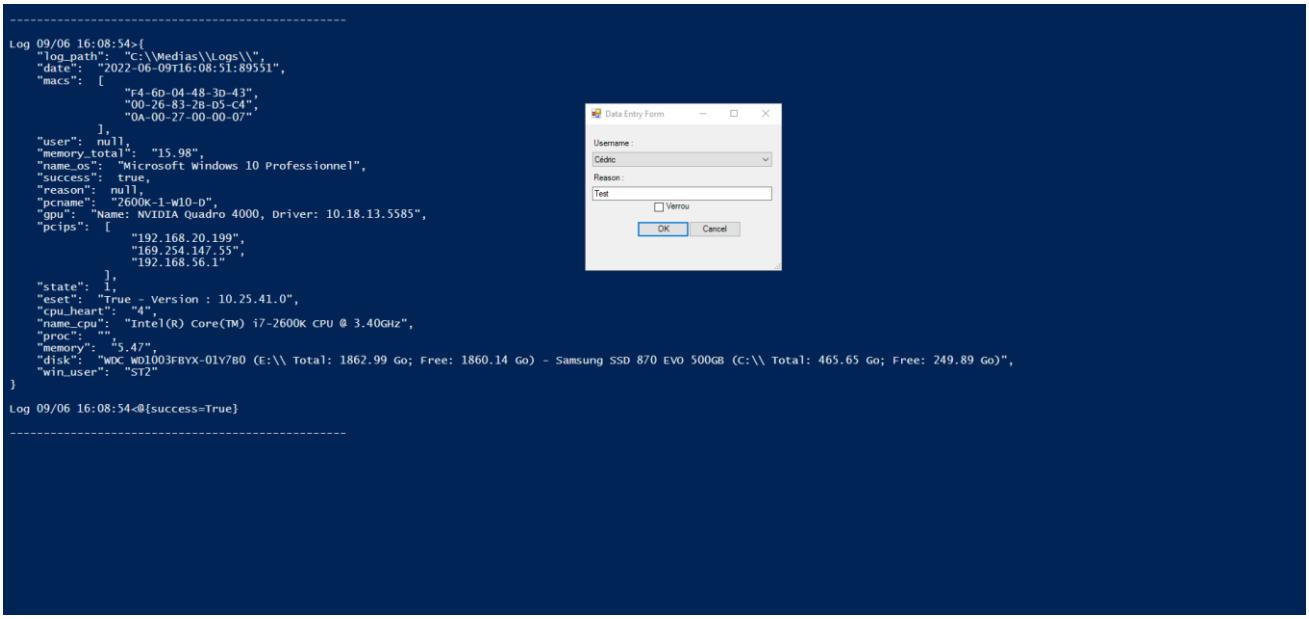

Figure 17 : Dialogue box et Log

La dialogue box* permet de récupérer le nom de l'utilisateur et la raison pour laquelle il utilise la machine. Il est censé permettre également d'informer si la machine est verrouillée pour des tests. A ce jour, cette fonctionnalité n'est pas intégrée car elle dépend entièrement du serveur (figure 17, 18 et 19).

```
function DialogueBox{
    Add-Type -AssemblyName System.Windows.Forms
    Add-Type -AssemblyName System.Drawing

    $form = New-Object System.Windows.Forms.Form
    $form.Text = 'Data Entry Form'
    $form.Size = New-Object System.Drawing.Size(300,250)
    $form.StartPosition = 'CenterScreen'

    $okButton = New-Object System.Windows.Forms.Button
    $okButton.Location = New-Object System.Drawing.Point(75,140)
    $okButton.Size = New-Object System.Drawing.Size(75,23)
    $okButton.Text = 'OK'
    $okButton.DialogResult = [System.Windows.Forms.DialogResult]::OK
    $form.AcceptButton = $okButton
    $form.Controls.Add($okButton)

    $cancelButton = New-Object System.Windows.Forms.Button
    $cancelButton.Location = New-Object System.Drawing.Point(150,140)
    $cancelButton.Size = New-Object System.Drawing.Size(75,23)
    $cancelButton.Text = 'Cancel'
    $cancelButton.DialogResult = [System.Windows.Forms.DialogResult]::Cancel
    $form.CancelButton = $cancelButton
    $form.Controls.Add($cancelButton)

    $label1 = New-Object System.Windows.Forms.Label
    $label1.Location = New-Object System.Drawing.Point(10,20)
    $label1.Size = New-Object System.Drawing.Size(280,20)
    $label1.Text = 'Username : '
    $form.Controls.Add($label1)

    $liste = New-Object System.Windows.Forms.ComboBox
    $liste.Location = New-Object System.Drawing.Point(10,40)
    $liste.Size = New-Object System.Drawing.Size(260,20)
    $liste.DropDownStyle = "DropDownList"
    $liste.Items.AddRange($Global:Liste_agent)
    $form.controls.add($liste)
```

Figure 18 : Dialogue box code 1

```
    $label2 = New-Object System.Windows.Forms.Label
    $label2.Location = New-Object System.Drawing.Point(10,70)
    $label2.Size = New-Object System.Drawing.Size(280,20)
    $label2.Text = 'Reason : '
    $form.Controls.Add($label2)

    $textBox = New-Object System.Windows.Forms.TextBox
    $textBox.Location = New-Object System.Drawing.Point(10,90)
    $textBox.Size = New-Object System.Drawing.Size(260,20)
    $form.Controls.Add($textBox)

    $checkbox = New-Object System.Windows.Forms.checkbox
    $checkbox.Location = New-Object System.Drawing.Point(100,110)
    $checkbox.Size = New-Object System.Drawing.Size(280,20)
    $checkbox.Text = 'Verrou '
    $form.Controls.Add($checkbox)

    $form.Topmost = $true

    $form.Add_Shown({$textBox.Select()})
    $result = $form.ShowDialog()

    if ($result -eq [System.Windows.Forms.DialogResult]::OK){
        $Message = @{User=$liste.Text;Reason=$textBox.Text;Verrou=$checkbox.Checked}
    }
    return $Message
}
```

Figure 19 : Dialogue box code 2

4.6 Correctifs et ajouts

Le premier ajout effectué était de mettre en première ligne les paramètres globaux, pour pouvoir les modifier plus simplement (figure 20).

```
#Paramètres globaux
$Global:Liste_agent = @("Xavier","Nicolas","Lionel","Cedric","Romain","Eric","Ines")
$Global:URL = "192.168.20.199"
$Global:Port = "6123"
$Global:Timer = 5
```

Figure 20 : Paramètres globaux

Le deuxième ajout et la présélection du prénom de l'utilisateur dans la dialogue box (figure 21).

```
if($env:UserName -in $Global:Liste_agent){
    for ($i = 0; $i -lt ($Global:Liste_agent).Length; $i++){
        if ($env:UserName -eq $Global:Liste_agent[$i]){
            $liste.SelectedIndex = $i
        }
    }
}
```

Figure 21 : Présélection dialogue box

Le troisième ajout a été la création automatique du dossier AVSet_PC_Data dans les fichier logs, et si le dossier existe déjà rien ne se passe. On peut aussi y voir les paramètres globaux utilisé (figure 22).

```
if($Global:Compteur -eq 1){
    $Global:DialogueBox = DialogueBox
    if($(Test-Path -LiteralPath "C:\Medias\Logs\AVSet_PC_Data" -PathType Container) -eq $False){
        New-Item "C:\Medias\Logs\AVSet_PC_Data" -itemType Directory
    }
    $Global:Fichier_Log = New-Item -Path "C:\Medias\Logs\AVSet_PC_Data\Log-$(Get-Date -Format "
    dd-MM HH-mm-ss")-$($Global:DialogueBox.User).txt" -ItemType File
}
$Param = @{port=$Global:Port;url=$Global:URL;user=$Global:DialogueBox.User;reason=$Global:DialogueBox.Reason}
```

Figure 22 : Création d'un dossier de log

J'ai fusionné les fonctions AVGet_MAC et AVGet_IP, pour que lors de l'envoi des données elles soient affichées dans le même ordre, j'ai créé des couples Mac-IP. Lorsqu'il y a plusieurs IP pour une seule adresse Mac c'est la première adresse IP qui est gardée. Dans le cas où la machine possède un virtual private network ou VPN* ou des machines virtuelles, et que leur interface est désactivée, ce qui est le cas de la majorité des machines chez Anyware Video, la valeur renvoyée sera nulle. Si cela se produit, cela peut entraîner des erreurs critiques.

```
[Hashtable]AVGet_Mac_Ip(){
    $Dico = @{}
    $IntIndexs = (Get-NetAdapter).InterfaceIndex
    foreach($IntIndex in $IntIndexs){
        $Mac = (Get-NetAdapter -InterfaceIndex "$IntIndex").MacAddress
        $IP = (Get-NetIPAddress -InterfaceIndex "$IntIndex" -AddressFamily IPV4).IPAddress
        if($IP.count -gt 1){
            if ($IP[0] -ne $null){
                $Dico += @{$Mac = $IP[0]}
            }
        }
        else{
            if ($IP -ne $null){
                $Dico += @{$Mac = $IP}
            }
        }
    }
    return $Dico
}
```

Figure 23 : Correctif adresse IP et Mac

Suite à des tests sur diverses machines pour déployer le logiciel, je me suis rendu compte que parfois, il pouvait y avoir plusieurs cartes graphiques. J'ai donc fait en sorte que la première soit affichée. De plus, j'ai ajouté une ligne de code permettant de retirer tout accent car c'était l'une des raisons principales d'erreurs.

```
[String]AVGet_GPU_Name(){ #Exemple Name: NVIDIA Quadro 4000, Driver: 10.18.13.5585
    $Name = (Get-CimInstance -ClassName Win32_VideoController).Name
    $Driver = (Get-CimInstance -ClassName Win32_VideoController).DriverVersion
    if($Name.count -gt 1){
        $msg = "Name: " + $Name[0] + ", Driver: " + $Driver[0]
    }
    else{
        $msg = "Name: " + $Name + ", Driver: " + $Driver
    }
    return [Text.Encoding]::ASCII.GetString([Text.Encoding]::GetEncoding("Cyrillic").GetBytes($msg))
}
```

Figure 24 : Correctif fonction GPU

Un jour, l'une des machines est tombée en panne à cause d'une surchauffe du processeur. À la suite de cette panne, mon patron m'a demandé de voir comment récupérer la température du processeur et de la carte graphique avec PowerShell. Après quelques recherches, j'ai trouvé une commande qui pourrait fonctionner mais elle a finalement produit une erreur. J'ai poursuivi mes recherches sur l'erreur mais tout ce que j'ai pu lire sur le sujet ne m'a pas aidé. Après avoir expliqué à mon patron que la commande ne fonctionnait pas il m'a demandé de voir s'il était possible de récupérer des informations sur les ventilateurs. J'ai alors trouvé une classe Windows faite pour cela mais après des heures de tests et de recherches cela ne fonctionnait toujours pas. Je me suis ensuite demandé si le problème pouvait être matériel, et en effet, il s'est avéré qu'il provenait de la carte mère. Elle n'était pas suffisamment récente pour pouvoir récupérer ce type d'informations.

J'ai ajouté en annexe le code final.

4.7 Gestion de fichier via XML

Durant mon stage, le responsable technique a eu une demande de client qui souhaitait un script PowerShell pour analyser des fichiers XML et déplacer en conséquence des fichiers. La première version du script fonctionnait très bien mais je n'avais pas bien exploité les fichiers XML et cela pouvait créer des bugs si les fichiers n'étaient pas conformes. J'avais opté pour un décomposition ligne par ligne comme pour le fichier ini, ce qui empêchait la gestion des erreurs.

Après une nouvelle explication, j'ai réécrit entièrement le script pour parvenir à celui-ci (figure 25). Pour effectuer mes tests, j'ai dû recréer l'arborescence de l'entreprise.

```
$IP = "\\192.168.10.34"
$DossierXML = (Get-ChildItem "$IP\interfaces_1\TRF-ANY\ArchiveList").Name #Récupère la liste de tous les fichiers xml
ForEach($Fichier in $DossierXML){
    Start-Sleep -Seconds 5 #Permet d'attendre 5 secondes entre chaque fichier traité
    If ($Fichier.EndsWith(".xml") -eq $True){ #Pour ne traiter que les fichier.xml dans le cas où il y aurait d'autre fichier
        try{
            $XML = [xml] (Get-Content "$IP\interfaces_1\TRF-ANY\ArchiveList\$Fichier") #Récupère le contenu du fihier xml
            $ID = $XML.Request.Purge.ID #Récupère l'ID du fichier xml
            if($ID -eq "" -or $ID -eq $null ){ # Si la ID ne vaut rien alors déplacer le fichier xml dans KO
                Write-host "$Fichier : Il n'y a pas d'ID"
                Move-Item -Path "$IP\interfaces_1\TRF-ANY\ArchiveList\$Fichier" -Destination "$IP\interfaces_1\TRF-ANY\ArchiveList\KO\"
            }
            else{ #Sinon on essaie de déplacer le fichier mxf correspondant puis on déplace le fichier xml
                Try{
                    Move-Item -Path "$IP\interfaces_1\TEMP\$ID.mxf" -Destination "$IP\interfaces_1\Archive\" -ErrorAction Stop
                    Write-Host "$ID : OK"
                }
                Catch{
                    Write-Host "Le fichier $ID.mxf n'a pas été trouvé"
                }
                Move-Item -Path "$IP\interfaces_1\TRF-ANY\ArchiveList\$Fichier" -Destination "$IP\interfaces_1\TRF-ANY\Purge_Lists\"
            }
        }
        Catch{#Si la structure du fichier est incorrecte mettre le fichier xml dan KO
            Move-Item -Path "$IP\interfaces_1\TRF-ANY\ArchiveList\$Fichier" -Destination "$IP\interfaces_1\TRF-ANY\ArchiveList\KO\"
            Write-host "$Fichier : La structure est incorrecte"
        }
    }
}
```

Figure 25 : Script Gestion de fichiers XML

5 Conclusion

En résumé, j'ai développé un script en PowerShell tout en développant mes connaissances dans ce langage. Ce développement s'est fait étape par étape, la première a été de comprendre comment fonctionnait ce logiciel. Puis la deuxième étape, a été de réussir à me connecter au serveur avec un requête JSON. La troisième étape a été de faire toutes les fonctions qui récupèrent les informations de l'ordinateur. Il a ensuite fallu modifier le code pour qu'il y ait des classes, afin de permettre une meilleure reprise en main du projet. La dernière étape a été d'ajouter des logs et une dialogue box. J'ai par la suite effectué une batterie de tests pour corriger et améliorer quelques détails du script.

Je pense que pour améliorer ce logiciel, il serait nécessaire de restructurer la partie serveur afin qu'elle devienne plus compréhensible et plus facilement modifiable. Il conviendrait également d'ajouter la réception puis l'envoi de l'information de verrouillage de la machine pour un nombre d'heures défini. Il serait peut-être intéressant de renouveler l'ergonomie du logiciel.

J'ai donné le meilleur de moi-même pour réaliser que ce script soit le plus fonctionnel possible et qu'il soit exploitable et opérationnel pour l'entreprise. Les erreurs faites lors de ce stage ont été très instructives et formatrices. Le fait de pouvoir recevoir de l'aide lorsque j'en avais besoin a été très bénéfique à la complétude de ma formation. Compte tenu de l'absence d'orientation de ma formation dans le domaine de la programmation, j'ai parfois rencontré quelques difficultés que je suis parvenu à dépasser.

Enfin, le script va être déployé sur les machines de l'entreprise c'est donc pour moi une réussite et une satisfaction totales. De plus, mon intégration au sein de l'entreprise était très simple et pour une première expérience dans le monde du travail je n'aurais pas pu rêver mieux que Anyware Video. Cette expérience m'a conforté dans mon choix d'orientation qui est d'aller poursuivre mes études dans une école d'ingénieur au Canada et d'y étudier la spécialité « les jeux vidéo et la réalité virtuelle ».

6 Remerciements

Je souhaiterais, en premier lieu, remercier, mon maître de stage Monsieur Xavier Picat, pour, sa disponibilité, son partage d'expérience et surtout pour m'avoir accordé sa confiance tout au long de mon stage.

En second lieu, je remercie toute l'équipe d'Anyware Video et plus particulièrement Monsieur Eric Avanzato qui, malgré ses contraintes professionnelles m'a consacré du temps de travail et du temps personnel pour m'accompagner durant tout le stage.

Je tenais à remercier également mon responsable académique Monsieur Ivan Madjarov.

Et enfin, merci à Monique ma « Maman » pour sa patience et le temps consacré à la relecture de ce rapport. Merci à François « mon Papa » de m'avoir préparé tous les déjeuners durant le stage.

7 Glossaire

PowerShell, est une solution multiplateforme d'automatisation des tâches, composée d'un interpréteur de commandes (shell), d'un langage de script et d'un framework de gestion de la configuration.

Teams, est une application qui fait partie de la suite Office 365 et permet aux équipes une collaboration dans le cloud. Elle offre, entre autres, des fonctionnalités de messagerie d'entreprise, d'appel, de visioconférence et de partage de fichiers. Elle peut être utilisée par des entreprises de toutes tailles.

Python, est un langage de programmation interprété, multi-paradigme et multiplateformes. Il favorise la programmation impérative structurée, fonctionnelle et orientée objet.

XML, Langage de structuration de données, utilisé notamment pour la gestion et l'échange d'informations sur Internet.

POST, La méthode POST écrit les paramètres URL dans la requête HTTP pour le serveur. Les paramètres ne sont donc pas visibles pour les utilisateurs et la portée des requêtes POST est illimitée.

Ini, un fichier INI est un fichier de configuration dans un format de données introduit par les systèmes d'exploitation Windows en 1985. Par convention les noms de ces fichiers portent l'extension «.ini ».

VPN, Un VPN est un réseau privé virtuel (en anglais virtual private network). C'est un outil numérique qui redirige votre trafic Internet via un tunnel sécurisé, masquant votre adresse IP et chiffrant vos données. C'est ainsi qu'un VPN protège vos données privées et vous protège des cyberattaques potentielles.

Dialogue box, est une fenêtre secondaire qui permet aux utilisateurs d'effectuer une commande, de poser une question ou de fournir aux utilisateurs des informations ou des commentaires de progression.

8 Annexes

```
#Paramètres globaux
$Global:Liste_agent = @("Xavier","Nicolas","Lionel","Cedric","Romain","Eric","Ines") # ATTENTION IL NE FAUT PAS METTRE LES ACCENTS
$Global:URL = "192.168.20.199"
$Global:Port = "6123"
$Global:Timer = 5

#Variables globales
$Global:Compteur = 0
$Global:DialogueBox = @{}

class CRecup{
    [String]$m_strCPU_Cores
    [String]$m_strCPU_Name
    [String]$m_strDate
    [String]$m_strDisk
    [String]$m_strEset
    [String]$m_strGPU_Name
        [Hashtable]$m_Ini
    [String]$m_strLog_Path
    [Hashtable]$m_Mac_Ip
    [String]$m_strMemory
        [String]$m_strMemory_Total
        [String]$m_strOs_Name
        [String]$m_strProc

    [void]Initialisation(){
        $this.m_strCPU_Cores = $this.AVGet_CPU_Cores()
        $this.m_strCPU_Name = $this.AVGet_CPU_Name()
        $this.m_strDisk = $this.AVGet_Disk()
        $this.m_strGPU_Name = $this.AVGet_GPU_Name()
        $this.m_strLog_Path = $this.AVGet_Log_Path()
        $this.m_strOs_Name = $this.AVGet_OS_Name()
        $this.m_strMemory_Total = $this.AVGet_Memory_Total()
    }

        [void]Update()
        {
        $this.m_strDate = $this.AVGet_Date()
        $this.m_strEset = $this.AVGet_Eset()
        $this.m_Ini = $this.AVGet_Ini()
                $this.m_Mac_Ip = $this.AVGet_Mac_Ip()
        $this.m_strMemory = $this.AVGet_Memory()
                $this.m_strProc = $this.AVGet_Proc()

        }

    [String]AVGet_CPU_Cores(){
                return (Get-CimInstance -ClassName Win32_Processor).NumberOfCores
        }

    [String]AVGet_CPU_Name(){
                return (Get-CimInstance -ClassName Win32_Processor).Name
        }

    [String]AVGet_Date(){
                return Get-Date -Format "yyyy-MM-ddTHH:mm:ss:fffs"
        }

    [String]AVGet_Disk(){
                $Disks_Letter = (Get-Volume).DriveLetter
                $Disk_Letter_Partition = (Get-Partition).
                $Disks_Name = (Get-PhysicalDisk).FriendlyName
                $Message = ""
                $i = 0
                Foreach($Disk_Letter in $Disks_Letter){
                        if($Disk_Letter -ne $null){
                                $Disk_Type = (Get-Volume -DriveLetter "$Disk_Letter").DriveType
                                if($Disk_Type -ne "CD-ROM"){
                                        if($Message -ne ""){$Message += " - "}
```

```
                                        if ($Disk_Letter -in $Disk_Letter_Partition){
                                                $Partitions = Get-Partition -DriveLetter "$Disk_Letter"
                                                $Disk_Number = $Partitions.DiskNumber
                                                $Disk_Space = [math]::round((Get-Volume -DriveLetter
"$Disk_Letter").Size/1Gb,2)
                                                $Disk_Space_Remaning = [math]::round((Get-Volume -DriveLetter
"$Disk_Letter").SizeRemaining/1Gb,2)
                                                $Disk_Name = (Get-PhysicalDisk -DeviceNumber
"$Disk_Number").FriendlyName
                                                $Message += "$($Disk_Name) ($($Disk_Letter):\ Total:
$($Disk_Space) Go; Free: $($Disk_Space_Remaning) Go)"
                                        }
                                        else{
                                                $Disk_Name = $Disks_Name[$i]
                                                $Disk_Space = [math]::round((Get-Volume -DriveLetter
"$Disk_Letter").Size/1Gb,2)
                                                $Disk_Space_Remaning = [math]::round((Get-Volume -DriveLetter
"$Disk_Letter").SizeRemaining/1Gb,2)
                                                $Message += "$($Disk_Name) ($($Disk_Letter):\ Total:
$($Disk_Space) Go; Free: $($Disk_Space_Remaning) Go)"
                                        }
                                }
                        }
                        $i += 1
                }
                return $Message
        }

    [String]AVGet_Eset(){
                $process = (Get-Process).ProcessName
                $version = (Get-Item 'C:\program files\ESET\ESET Security\eguiProxy.exe').VersionInfo.FileVersion
                if ("eguiProxy" -and "ekrn" -and "ERAAgent" -in $process){
                        $Bool = "True"
                        }
                else{
                        $Bool = "False"
                        }
                $message = $Bool + " - Version : " + $version
                return $message
        }

     [String]AVGet_GPU_Name(){ #Exemple Name: NVIDIA Quadro 4000, Driver: 10.18.13.5585
       $Name = (Get-CimInstance -ClassName Win32_VideoController).Name
       $Driver = (Get-CimInstance -ClassName Win32_VideoController).DriverVersion
       if($Name.count -gt 1){
          $msg = "Name: " + $Name[0] + ", Driver: " + $Driver[0]
       }
       else{
          $msg = "Name: " + $Name + ", Driver: " + $Driver
       }
                return [Text.Encoding]::ASCII.GetString([Text.Encoding]::GetEncoding("Cyrillic").GetBytes($msg))
          }

    [Hashtable]AVGet_Ini(){
       if($Global:Compteur -eq 1){
          $Global:DialogueBox = DialogueBox
          if($(Test-Path -LiteralPath "C:\Medias\Logs\AVSet_PC_Data" -PathType Container) -eq $False){New-Item
"C:\Medias\Logs\AVSet_PC_Data" -itemType Directory}
          $Global:Fichier_Log = New-Item -Path "C:\Medias\Logs\AVSet_PC_Data\Log-$(Get-Date -Format "dd-MM HH-
mm-ss")-$($Global:DialogueBox.User).txt" -ItemType File
       }
                $Param =
@{port=$Global:Port;url=$Global:URL;user=$Global:DialogueBox.User;reason=$Global:DialogueBox.Reason}
                try{
                        $Ini = Get-Content "$PSScriptRoot\AVVerifierService.ini" -Force -ErrorAction Stop
                        $IniHash = @{}
                        $IniTemp = @()
                        ForEach ($Line in $Ini){
                                If ($Line -ne "" -and $Line.StartsWith("[") -ne $True){
                                        $IniTemp += $Line
                                }
                        }
                        ForEach($Line in $IniTemp){
```

```
                                $SplitArray = $Line.Split("=")
                            $SplitArray[0] = $SplitArray[0].Trim()
                            $SplitArray[1] = $SplitArray[1].Trim()
                                $IniHash += @{$SplitArray[0] = $SplitArray[1]}
                        }
                        if ($IniHash.port -eq "" -or $IniHash.url -eq "" -or $IniHash.user -eq "" -or $IniHash.reason -eq
""){
                                Write-Host "Une valeur est manquante dans le fichier AVVerifierService.ini, les valeurs
par défauts vont être utilisés."
            $IniHash = $Param
            $Str_IniHash = $IniHash | Out-String
            Write-Host $Str_IniHash
                        }
                    }
                    catch{
        $IniHash = $Param
                    }
                    return $IniHash
        }

    [String]AVGet_Log_Path(){
                    return Get-ItemPropertyValue -Path "HKLM:\SOFTWARE\Anyware Video" -Name "LogDirectory"
        }

        [Hashtable]AVGet_Mac_Ip(){
    $Dico = @{}
    $IntIndexs = (Get-NetAdapter).InterfaceIndex
    foreach($IntIndex in $IntIndexs){
        $Mac = (Get-NetAdapter -InterfaceIndex "$IntIndex").MacAddress
        $IP = (Get-NetIPAddress -InterfaceIndex "$IntIndex" -AddressFamily IPV4).IPAddress
        if($IP.count -gt 1){
            if ($IP[0] -ne $null){
                $Dico += @{$Mac = $IP[0]}
            }
        }
        else{
            if ($IP -ne $null){
                $Dico += @{$Mac = $IP}
            }
        }
    }
    return $Dico
}

    [String]AVGet_Memory(){
                    $var =  Get-WmiObject -Class WIN32_OperatingSystem
                    return [math]::Round((($var.TotalVisibleMemorySize - $var.FreePhysicalMemory)/1024/1024),2)
        }

        [String]AVGet_Memory_Total(){
                    return [math]::Round((((Get-WmiObject -Class
WIN32_OperatingSystem).TotalVisibleMemorySize)/1024/1024),2)
        }

        [String]AVGet_OS_Name(){
                    $name = ((Get-CimInstance -ClassName Win32_OperatingSystem).Name).Split("|")
                    return $name[0]
        }

        [String]AVGet_Proc(){
                    $proc = @((Get-Process).ProcessName)
                    $AVlist = @('AllInOne',
                                        'AllInOneMediaManager',
                                        'IngenieRecorder',
                                        'AllInOneMediaReader',
                                        'AVMediaTranscoder',
                                        'DMPTest',
                                        'AMPTest',
                                        'CastGenie',
                                        'AVSupervisor',
                                        'AVController',
                                        'CastEdit',
                                        'DUBMaster',
```

```
                                        'PigeEncoder',
                                        'PigeManager')
                $AVproc = @()
                for ($i = 0; $i -lt ($proc.Length); $i++){
                        if ($proc[$i] -in $AVlist){
                                if ($proc[$i] -notin $AVproc){
                                        $AVproc += $proc[$i]
                                }
                        }
                }
                return $AVproc
        }

}

function DialogueBox{
    Add-Type -AssemblyName System.Windows.Forms
    Add-Type -AssemblyName System.Drawing

    $form = New-Object System.Windows.Forms.Form
    $form.Text = 'Data Entry Form'
    $form.Size = New-Object System.Drawing.Size(300,250)
    $form.StartPosition = 'CenterScreen'

    $okButton = New-Object System.Windows.Forms.Button
    $okButton.Location = New-Object System.Drawing.Point(75,140)
    $okButton.Size = New-Object System.Drawing.Size(75,23)
    $okButton.Text = 'OK'
    $okButton.DialogResult = [System.Windows.Forms.DialogResult]::OK
    $form.AcceptButton = $okButton
    $form.Controls.Add($okButton)

    $cancelButton = New-Object System.Windows.Forms.Button
    $cancelButton.Location = New-Object System.Drawing.Point(150,140)
    $cancelButton.Size = New-Object System.Drawing.Size(75,23)
    $cancelButton.Text = 'Cancel'
    $cancelButton.DialogResult = [System.Windows.Forms.DialogResult]::Cancel
    $form.CancelButton = $cancelButton
    $form.Controls.Add($cancelButton)

    $label1 = New-Object System.Windows.Forms.Label
    $label1.Location = New-Object System.Drawing.Point(10,20)
    $label1.Size = New-Object System.Drawing.Size(280,20)
    $label1.Text = 'Username : '
    $form.Controls.Add($label1)

    $liste = New-Object System.Windows.Forms.ComboBox
    $liste.Location = New-Object System.Drawing.Point(10,40)
    $liste.Size = New-Object System.Drawing.Size(260,20)
    $liste.DropDownStyle = "DropDownList"
    $liste.Items.AddRange($Global:Liste_agent)
    $form.controls.add($liste)

    $label2 = New-Object System.Windows.Forms.Label
    $label2.Location = New-Object System.Drawing.Point(10,70)
    $label2.Size = New-Object System.Drawing.Size(280,20)
    $label2.Text = 'Reason : '
    $form.Controls.Add($label2)

    $textBox = New-Object System.Windows.Forms.TextBox
    $textBox.Location = New-Object System.Drawing.Point(10,90)
    $textBox.Size = New-Object System.Drawing.Size(260,20)
    $form.Controls.Add($textBox)

    $checkbox = New-Object System.Windows.Forms.checkbox
    $checkbox.Location = New-Object System.Drawing.Point(100,110)
    $checkbox.Size = New-Object System.Drawing.Size(280,20)
    $checkbox.Text = 'Verrou '
    $form.Controls.Add($checkbox)

    $form.Topmost = $true

    if($env:UserName -in $Global:Liste_agent){
```

```
        for ($i = 0; $i -lt ($Global:Liste_agent).Length; $i++){
            if ($env:UserName -eq $Global:Liste_agent[$i]){
                $liste.SelectedIndex = $i
            }
        }
    }
    $form.Add_Shown({$textBox.Select()})
    $result = $form.ShowDialog()
    if ($result -eq [System.Windows.Forms.DialogResult]::OK){
        $Message = @{User=$liste.Text;Reason=$textBox.Text;Verrou=$checkbox.Checked}
    }
    return $Message
}

$CR = [CRecup]::new()
$CR.Initialisation()

while ($True){
    $CR.Update()
    $Body = @{
                success= $true
                pcname= $env:computername
                pcips= $CR.m_Mac_Ip.Values
                macs= $CR.m_Mac_Ip.Keys
                win_user= $env:UserName
                date= $CR.m_strDate
                memory= $CR.m_strMemory
                log_path= $CR.m_strLog_Path
                proc= $CR.m_strProc
                state= 1
                eset= $CR.m_strEset
                disk= $CR.m_strDisk
                name_cpu= $CR.m_strCPU_Name
                name_os= $CR.m_strOs_Name
                memory_total= $CR.m_strMemory_Total
                cpu_heart= $CR.m_strCPU_Cores
                gpu= $CR.m_strGPU_Name
                user= $CR.m_Ini.user
                reason= $CR.m_Ini.reason
    }
    $JsonBody = $Body | ConvertTo-Json
    $Params = @{
        Method = "Post"
        Uri = "http://" + $CR.m_Ini.url + ":" + $CR.m_Ini.port + "/setPCData"
        Body = $JsonBody
        ContentType = "application/json"
    }

    $Response  = Invoke-RestMethod @Params

    Write-Host "
----------------------------------------------

Log $(Get-Date -Format "dd/MM HH:mm:ss")>$($JsonBody)"

    Write-host "
Log $(Get-Date -Format "dd/MM HH:mm:ss")<$($Response)

----------------------------------------------"

    if($Global:Compteur -gt 0){
        Add-Content -Path $Global:Fichier_Log -Value "
        ----------------------------------------------

        Log $(Get-Date -Format "dd/MM HH:mm:ss")>$($JsonBody)

        Log $(Get-Date -Format "dd/MM HH:mm:ss")<$($Response)

        ----------------------------------------------"
    }
    $Global:Compteur += 1
    start-sleep -seconds $Global:Timer
}
```