



Analyseur de droits NTFS

Rapport TPI

Travail Pratique Individuel

Neal van Rooij – École des Arches

15 mars au 4 avril 2017

Chef de projet : M. Olivier CLERC

Experts : M. Patrick PRETRE et M. Laurent DUDING

Table des matières

Analyse préliminaire.....	3
Introduction.....	3
Objectifs.....	3
Type de client	3
Planification initiale.....	3
Analyse	5
Stratégie de test	6
Budget	6
Étude de faisabilité.....	7
Compléments d'analyse	7
Planification finale.....	7
Historique	9
Conception	9
Dossier de conception	9
Règles pour l'écriture des scripts	9
Historique	10
Réalisation	10
Domaine	10
Prérequis	10
Dossier de réalisation	10
Fonctionnement	12
Lecture des résultats	16
Descriptions des tests effectués.....	16
Erreurs restantes	18
Dossiers d'archivage.....	19
Mise en service.....	19
Installation du produit.....	19
Liste de documents fournis	19
Conclusion	20
Objectifs atteints / non-atteints.....	20
Problèmes rencontrés et résolus	20
Points positifs/négatifs.....	20
Difficultés particulières.....	21
Suites possibles pour le projet (évolutions & améliorations)	21
Bilan de planification.....	21
Bilan personnel.....	21
Remerciements	21
Annexes	22
Bibliographie.....	22
Journal de travail	23
Archive du projet.....	23

Analyse préliminaire

Introduction

Ce projet a pour objectif de répondre à une demande venant d'un client, qui était :

- Mettre en place des rapports réguliers des accès sur une arborescence choisie et permettre une consultation des rapports établis

« Il ne s'agit pas de mettre en place une veille avec alerte, sujet déjà abondamment couvert par des outils du commerce, mais d'établir des états successifs que l'on pourra comparer au besoin.

Le choix de l'outil d'export brut des données est Powershell. Ce choix est délibéré : le candidat souhaite approfondir ses connaissances sur cet outil, et son côté universel convient bien à la nature du projet ; passé la phase de TPI, ce projet pourra en effet faire l'objet d'un déploiement dans des environnements hétérogènes. » Extrait du descriptif du projet, cahier des charges

L'objectif principal est de posséder de bonnes bases dans le langage PowerShell une fois le projet terminé. Le design des rapports n'est pas quelque chose d'attendu dans ce TPI, mais pourra faire partie d'un développement post-TPI.

Objectifs

Les objectifs fixés dans ce travail sont les suivants :

- *« Mise en service proprement dite du laboratoire : création du domaine Windows, utilisateurs, groupes, arborescence de documents avec des droits divers : hérités et non hérités, accessibles et non accessibles aux administrateurs, tenant compte des attributs allow et deny. La distinction des droits suivants suffit : read/write/list/full control*
- *Élaboration du ou des scripts requis pour exporter les données nécessaires*
- *Discussion avec le client sur la manière de présenter les données*
- *Correction du ou des scripts, fusion des scripts en un seul outil*
- *Document de présentation basique sous excel, en guise de frontend*
- *Validation du résultat »* Extrait du descriptif du projet, cahier des charges

Les points qui seront évalués dans ce projet sont :

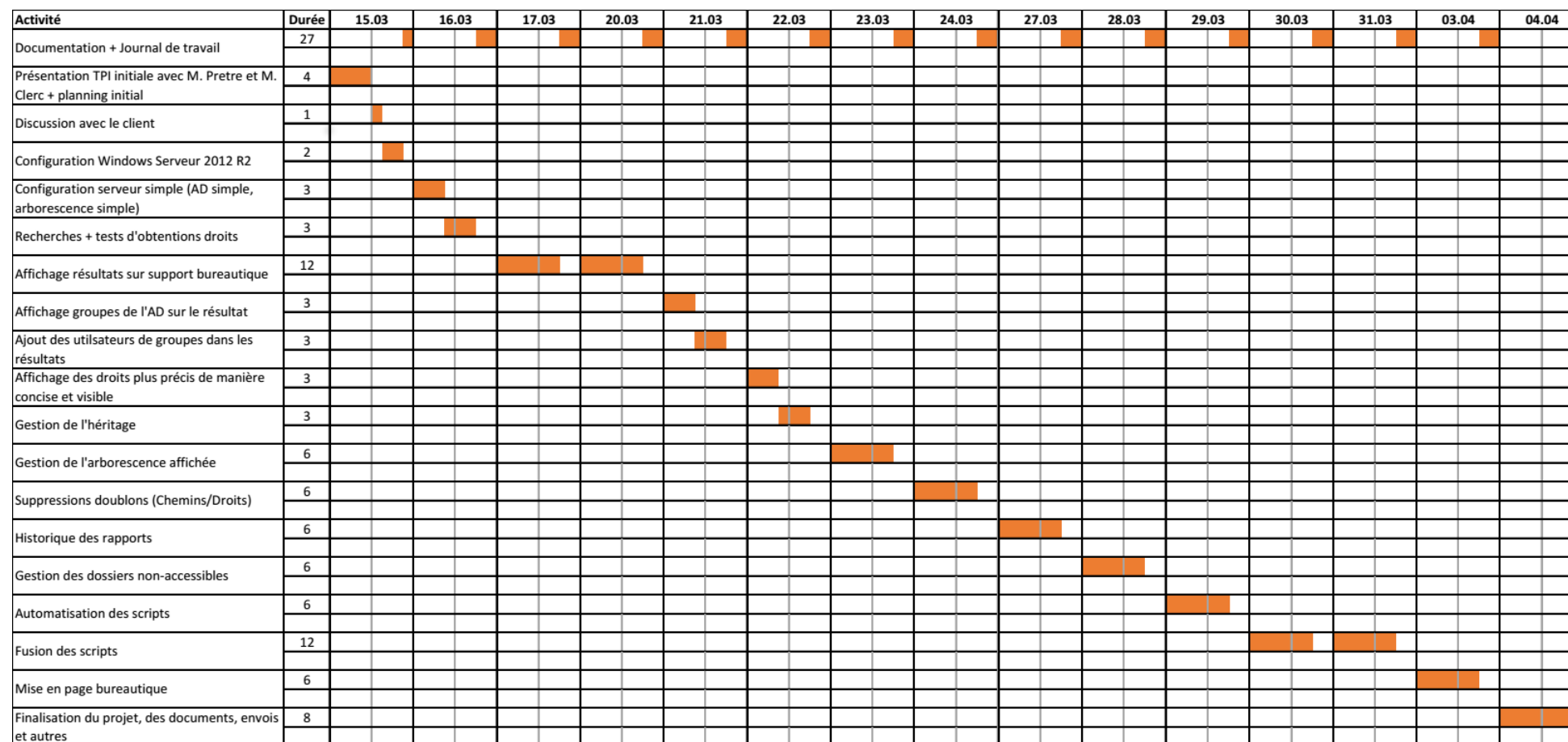
- Un bon fonctionnement du script, sans erreur ni argument inconnu
- Une bonne lisibilité du résultat obtenu
- Que le produit soit utilisable par le client

Type de client

Le client ayant effectué la demande du projet est un employé de l'entreprise d'informatique Practeo SA et il possède de bonnes connaissances en informatique. Le rapport avec différentes commandes ou instructions que l'on pourrait y trouver ainsi que le mode d'emploi sont adaptés à ses connaissances et ne sont pas vulgarisées pour un utilisateur lambda du projet.

Planification initiale

La planification initiale est disponible à la page suivante, sous la forme d'un diagramme Gantt.



1 - Organigramme Gantt initial

Analyse

Des maquettes internes, pour avoir une meilleure vision du résultat souhaité, ont été produites durant le premier jour du travail. Il s'agissait de maquettes sans véritablement savoir comment le projet allait se dérouler, donc qui appelaient à être changées. La première maquette donne une idée de ce à quoi le résultat pourrait ressembler.

Chemin					Type de Contrôle	Niveau de contrôle	Utilisateurs
C:							
	Users						
		Neal					
			Documents				
				Fichier.txt	Deny	FullControl	Domaine\Administrateur
					Allow	FullControl	Domaine\Neal
		Administrateur					
			Documents				
				AutreFichier.txt	Allow	FullControl	Domaine\Administrateur
					Deny	FullControl	Domaine\Neal








2 - Première version de la maquette

Suite au premier affichage de résultat, une seconde version de la maquette plus adaptée a été réalisée. C'est sous cette forme qu'un résultat intermédiaire a été montré au client, qui n'a pas fait de demande ou de réclamation quant à un affichage différent.

Chemin	Niveau de contrôle	Type de Contrôle	Utilisateurs
C:\Users\Neal\Documents\Fichier.txt	FullControl	Deny	Domaine\Administrateur
		Allow	Domaine\Neal
		Allow	Domaine\Romain
		Allow	Domaine\Stephane
C:\Users\Administrateur\AutreFichier.txt	FullControl	Allow	Domaine\Administrateur
		Deny	Domaine\Neal
		Deny	Domaine\Romain
	Read and Execute	Allow	Domaine\Stephane

3 - Seconde version de la maquette

L'enregistrement des rapports a également fait office d'une maquette, avec une arborescence de chaque année, puis de chaque mois, numérotés pour qu'ils apparaissent dans l'ordre du calendrier, contenant chacun un nombre de rapports équivalent au nombre de jours dans ce mois.

- ▼  Projet TPI
- ▼  Rapports
 - ▼  Année1
 -  01 - Janvier
 -  02 - Février
 -  Année2
 -  Scripts

4 - Maquette de la structure d'enregistrement des rapports

Stratégie de test

Des tests sont effectués pour vérifier le bon fonctionnement des différents scripts.

Les premiers tests auront pour objectifs de vérifier que la récupération des données soit exacte et que les données modifiées soient correctement prises en compte.

La première série de test sera tout d'abord effectuée sur la console intégrée au programme Windows PowerShell ISE, avec une arborescence très réduite, pour vérifier que les résultats obtenus coïncident avec les données effectives. Cette restriction sert à pouvoir avancer dans le projet dans le cas où l'affichage de données pose problème et prend plus de temps que prévu, tout comme pouvoir plus facilement repérer les données pas correctes pour modifier si besoin la récolte d'informations.

La seconde série de tests débutera une fois l'exportation des données réalisée. Il s'agira de vérifier que le tri des informations et le nettoyage de données inutiles soient corrects, pour éviter que des données qui doivent être affichées ne le soient pas. La base de travail de cette seconde partie de tests doit commencer avec la même arborescence que lors des premiers tests, pour les mêmes raisons de vérifications des données affichées.

Une fois le bon fonctionnement de cette partie du projet avec cette petite quantité d'informations, une arborescence plus importante pourra être utilisée. (Environ 2-3 dossiers, avec 30 fichiers environ) Ceci est mis en place pour voir si la quantité d'informations n'a pas d'impact sur la véracité des données affichées, et si la mise en page est toujours acceptable.

Une fois que les groupes/utilisateurs sont affichés sur les résultats et que les données sont correctes, une bien plus grande arborescence peut être utilisée, avec des données réelles cette fois (un serveur live par exemple). Le but de ces tests est de repérer des problèmes qui n'apparaissent pas avec des arborescences plus petites. De plus, cela offrira un nombre de cas différents bien plus importants. La véracité des résultats pourra difficilement être vérifiée avec un résultat trop important, mais quelques modifications sur des fichiers/dossiers facilement repérables pourraient être utilisées pour vérifier le bon fonctionnement de tout le système.

Les tests ne sont pas limités à exemple mentionnés ci-dessus, mais ceux-ci doivent être effectués.

Le candidat, Neal van Rooij, sera le seul testeur de ce travail.

Budget

Le TPI est effectué au sein de l'entreprise Practeo SA, au cours de l'année de stage de 4^{ème} année. Ce travail ne nécessite pas de coût supplémentaire au niveau matériel ou humain, si ce n'est l'utilisation d'un ESX déjà en cours d'utilisation et le coût humain nécessaire pour un TPI.

Étude de faisabilité

Risques techniques : Le candidat n'effectue pas un stage dans une entreprise de développement, mais de système. Les habitudes et la logique de développement ne sont pas exercées régulièrement, pouvant ralentir le projet.

De plus, le PowerShell est un langage pour lequel il a des connaissances basiques. L'apprentissage de ce langage est à prendre en compte dans le planning initial.

Risque pour le planning et les ressources humaines : Il n'y a pas de spécialiste PowerShell dans l'entreprise, mais plusieurs développeurs Ingénieurs qui peuvent intervenir en cas de besoin. Le temps d'une recherche de solution peut donc être allongée.

Risque budgétaire : Aucune notion d'argent n'a été mentionnée avant ou durant le projet. Le matériel et l'infrastructure nécessaires au travail sont déjà présents.

Le budget n'a pas d'impact sur la faisabilité du TPI.

Compléments d'analyse

Il existe déjà des programmes offrant un résultat attendu partiellement similaire, mais modifier le travail d'un autre n'est pas l'objectif de ce travail. La réalisation de ce projet permet d'avoir un logiciel fait-maison plus facilement modifiable après-coup grâce à la documentation présente. De plus, le faire soit même offre une flexibilité plus importante face aux potentielles demandes du client.

Planification finale

Le graphique de Gantt avec l'attribution du temps véridique du projet est disponible à la page suivante.

Activité	Durée	15.03	16.03	17.03	20.03	21.03	22.03	23.03	24.03	27.03	28.03	29.03	30.03	31.03	03.04	04.04
Documentation + Journal de travail	27															
	45.5															
Présentation TPI initiale avec M. Pretre et M. Clerc + planning initial	4															
	5															
Discussion avec le client	1															
	1															
Configuration Windows Serveur 2012 R2	2															
	1.5															
Configuration serveur simple (AD simple, arborescence simple)	3															
	1															
Recherches + tests d'obtentions droits	3															
	8															
Affichage résultats sur support bureautique	12															
	11.75															
Affichage groupes de l'AD sur le résultat	3															
	4															
Ajout des utilisateurs de groupes dans les résultats	3															
	12															
Affichage des droits plus précis de manière concise et visible	3															
	7.5															
Gestion de l'héritage	3															
	0															
Gestion de l'arborescence affichée	6															
	5															
Suppressions doublons (Chemins/Droits)	6															
	3.5															
Historique des rapports	6															
	10															
Gestion des dossiers non-accessibles	6															
	3															
Automatisation des scripts	6															
	1.5															
Fusion des scripts	12															
	0															
Mise en page bureautique	6															
	0															
Finalisation du projet, des documents, envois et autres	8															
	2															
Total d'heures passées (sur 120)	122.3															

5 - Planification finale

Historique

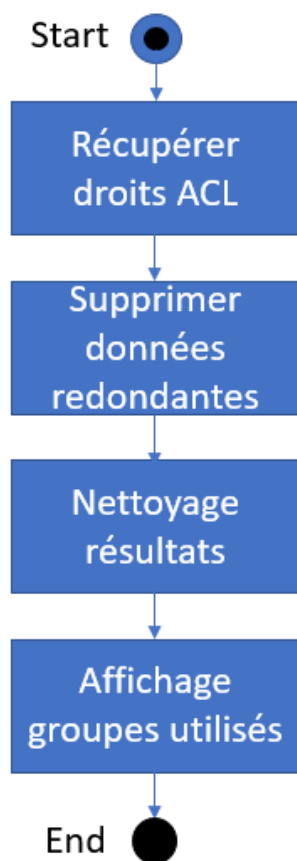
Il n'y a pas eu de modification demandée par le client au cours du projet, ni une différence entre le budget estimé et le budget véritable.

Conception

Dossier de conception

Ce projet est réalisé sur une machine ESX avec Windows Server 2012 R2, et les scripts sont écrits dans le langage PowerShell. Ce choix est fait car les serveurs mis en place par Practeo SA sont tous des Windows Server et que les plus récents sont de la version 2012 R2.

Il n'y a pas de restriction pour les logiciels à utiliser, si ce n'est que le résultat doit pouvoir être lu par le logiciel Excel 2013 pour présenter les résultats.



6 - Schéma des modules du programme

La première partie du travail consiste à mettre en place un domaine, avec arborescence/utilisateurs/groupes/droits allow/deny.

Concernant les scripts, le dossier départ de l'arborescence où il faut récupérer les droits ACL est noté dans un fichier settings.txt, pour que le client puisse choisir lui-même sur quel dossier il souhaite obtenir un rapport.

Chaque tâche est séparée pour pouvoir effectuer des changements plus facilement par la suite, si des changements sont nécessaires.

« Récupérer les droits ACL » est une récupération des droits où il faut effectuer un premier tri et n'afficher que le chemin des fichiers ayant les mêmes droits que leur dossier parent.

« Supprimer les données redondantes » doit supprimer les chemins semblables qui sont redondants, ainsi que les FileSystemRights (Read, Write, FullControl)

« Nettoyage résultat » est là pour n'afficher que les lignes contenant des données.

« Affichage groupes utilisés » doit afficher les groupes et leurs contenus qui sont présents dans les droits ACL.

Ces informations doivent être exportées dans des rapports structurés dans une structure semblable à la maquette proposée à la figure 3.

Règles pour l'écriture des scripts

Les noms de variables essaient d'être le plus représentatif de leur contenu. Ils commencent toujours par une minuscule et chaque mot supplémentaire a la première lettre en majuscule. De plus, toutes les variables sont écrites en anglais.

Les commentaires sont en français quant à eux.

Historique

La conception du projet a dû être modifiée durant le travail à plusieurs reprises, afin d'adapter les potentielles modifications. En premier lieu, un seul script avait été prévu pour extraire, mais suite à diverses modifications nécessaires, les tâches ont été réparties sur différents scripts pour principalement permettre un meilleur débogage ainsi que des modifications plus simples sur quelques points si besoin.

Réalisation

Domaine

Un domaine TPI.local a été configuré, avec plusieurs utilisateurs, groupe et gestion des droits sur les dossiers (allow et deny).

Prérequis

Afin que le projet fonctionne, il faut que la machine qui l'utilise ait l'autorisation d'exécuter des scripts. Il faut également que la machine ait un rôle d'Active Directory, et par conséquent le module Active directory PowerShell qui va avec.

Il faut également que l'utilisateur ait le droit d'exécuter les scripts. Pour autoriser les scripts non-signés par un organisme reconnu à être lancés, il faut entrer la commande suivante :

```
Set-ExecutionPolicy UnRestricted -Scope CurrentUser
```

Les valeurs par défauts n'autorisent que l'exécution de ligne, et non pas de scripts.

Pour plus de précision à ce sujet, le lien <https://technet.microsoft.com/fr-FR/library/hh847748.aspx> est disponible.

Dossier de réalisation

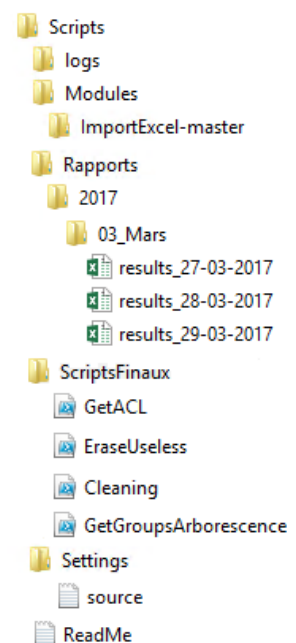
Répertoire du projet

Le répertoire contenant le travail est construit comme montré à la figure 6 ci-contre.

Un fichier README.txt sera inclus dans le travail, contenant les liens et la marche à suivre suivante. L'objectif de ce fichier ReadMe.txt est d'indiquer qu'il faut installer le module additionnel pour que les scripts fonctionnent tout en fournissant des informations brèves quant au projet.

Le lien qui suit permet de télécharger l'archive zip contenant le module supplémentaire <https://github.com/dfinke/ImportExcel/archive/master.zip>

Pour installer le module, il faut démarrer le script InstallModule.ps1 présent dans le dossier principal avec Powershell. Si l'exécution de scripts est autorisée, le module s'installera.



7 - Arborescence du projet

Le fichier ReadMe.txt contient le texte suivant :

ListingACL Script

Those scripts list the arborescence security of a given path and print them on a Excel sheet.

It works with the added module Import-Excel, already added in the package.

You can read more about it here: <https://github.com/dfinke/ImportExcel>

To install Import-Excel, start .\ImportExcel-master\InstallModule.ps1

There is no need to install anything, running the script GetACL.ps1 launches everything

Use the Task Scheduler to plan the automatization of it.

The path analysed can be change in .\Settings\source.txt

The results of those four scripts is being saved in . \Rapports

This project version: 1.0

Last update: 04.04.2017

Author: Neal van Rooij, Practeo SA

Scripts

Différents scripts sont produits, chacun effectuant une tâche définie. Les scripts suivants sont présents dans le projet.

- Un script de récupération de droits dossiers et son affichage dans un document Excel
GetACL.ps1

Ce script récupère le contenu du fichier .\Settings\source.txt contenant le chemin de destination dont l'arborescence sera analysée. Il va aussi vérifier si le dossier où le rapport ira s'enregistrer existe, et le crée si ce n'est pas le cas.

L'arborescence sera récupérée via un *Get-Childitem*. Les droits de chaque entrée de l'arborescence sont récupérés par le Cmdlet *Get-Acl*, et un tri est effectué pour n'avoir que les champs qui nous intéressent. Soit le chemin, le type de contrôle, le niveau de droits et les utilisateurs/groupes qui bénéficient de ces droits.

Ces données sont exportées sur un premier fichier Excel.

- Un script de modification du fichier Excel, pour épurer les résultats
EraseUseless.ps1

Ce script récupère les données du premier fichier Excel, vérifie pour chaque ligne si celle du dessus contient les mêmes informations. Si c'est le cas, le chemin et le niveau de contrôle sont rendu vide.

Le choix de ne pas vider le type de contrôle s'il est redondant est un choix personnel. Je trouve que les droits sont plus lisibles si leur type de contrôle est affiché.

Les données sont alors réexportées vers un second fichier Excel.

- Un script de nettoyage pour un résultat lisible et ordré
Cleaning.ps1

Cleaning.ps1 va récupérer le contenu du second document Excel pour supprimer toutes les lignes vides qu'il y a, va supprimer les deux premiers fichiers Excel et exportera les données vers un dernier fichier Excel.

- Un script de récupération des utilisateurs de l'Active Directory en fonction des groupes et utilisateurs listés par les précédents scripts.

GetGroupsArborescence.ps1

Ce script récupère les droits sur l'arborescence donnée, pour lister tous les utilisateurs différents listés dans ces droits. Les groupes de cette liste seront utilisés pour extraire les utilisateurs qu'ils contiennent, et sont exportés sur une nouvelle feuille du fichier Excel précédent.

Chaque script appelle son suivant pour que la chaîne dans sa totalité soit effectuée.

Version du produit

Le produit est encore à la version 1.0 puisqu'elle n'a pas encore été sortie.

Module supplémentaire

Le projet fonctionne avec un module supplémentaire qui permet d'exporter en fichier Excel, format xlsx. Le cmdlet Export-Csv est moins permissif que le module [Import-Excel](#) que j'ai ajouté. Ce module, a été mis à disposition de la communauté, crée par dfinke et mis à jour par la communauté, sur GitHub. Il permet d'avoir une plus grande liberté créative quant à l'aspect visuel d'une feuille Excel.

L'installation du module est disponible dans la rubrique Mise en service.

Fonctionnement

Chaque script commence par récupérer des informations qu'il devra utiliser. Il peut s'agir de données à modifier, de paramètres à prendre en compte ou de récupérer la date pour l'enregistrement.

Dans la figure ci-dessous, nous récupérons le dossier parent pour les rapports, qui est utilisé après par la suite, l'année, le mois sous format (01_Janvier, 02_février) pour faciliter la navigation dans les rapports et nous enregistrons le chemin de destination du dossier pour des conditions plus tard.

```
# Chemin pour tester l'arborescence
$sourcePath = Get-Content ".\Settings\source.txt"
# Récupération année et mois pour vérification des dossiers.
$year = get-date -f yyyy # Année en chiffre
$monthName = (Get-culture).TextInfo.ToTitleCase((Get-Culture).DateTimeFormat.GetMonthName((get-date -f MM)))
$monthNumber = get-date -f MM # Numéro du mois
$month = "$monthNumber" + "_" + "$monthName" # Égal à 01_Janvier, 02_Février, ...
$destinationPath = ".\Rapports\$year\$month\" # Chemin de destination pour enregistrer les rapports
```

8 - Récupération de données

Même si les variables au même nom sont enregistrées et conservées pour plus longtemps que simplement un script, je tiens à récupérer ces informations à chaque début de scripts qui en ont besoin pour conserver un système de récupération de données systématique, même s'il y a des changements à faire sur plusieurs scripts en cas de besoin.

Cette portion de script a nécessité plusieurs changements pour s'adapter à l'avancement du projet. Au début, les données étaient entrées à la main, puis récupérer automatiquement pour instaurer un système un peu plus automatique.

Après ces données obtenues, il faut récupérer l'arborescence et les droits de celle-ci. La commande ci-dessous les récupère, ordonné par le chemin du fichier. Cela permet d'avoir tout le contenu d'un dossier listé sous lui, et d'avoir un ordre un peu plus ordonné que par les données reçues par défauts.

```
$table = Get-ChildItem -Recurse -Path $sourcePath `
| Sort-Object FullName
```

9 - Récupération de l'arborescence

Les données récupérées sont passées dans une boucle foreach où les données subissent un tri pour que les résultats obtenus soient déjà triés une première fois. Par exemple, le script va vérifier si le chemin de l'entrée actuelle correspond à un fichier, et si c'est le cas, suivant si elle a les mêmes droits que son dossier parent, afficher ses droits ou non.

```
# Si l'entrée est un fichier
if ((Get-Item $item.FullName) -is [System.IO.FileInfo]) {
```

10 - Vérification si l'entrée est un fichier

Je choisis de ne faire ceci que pour les fichiers car il est important selon moi de toujours savoir quels droits sont attribués à un dossier, et par la même occasion à tous les documents qui en héritent.

Pour vérifier si les droits du fichier et de son dossier parent, les droits sont enregistrés dans deux variables différentes puis comparé grâce au cmdlet Compare-Object, comme montré ci-dessous.

```
if ((Compare-Object $StoredParentACL $StoredItemACL) -eq $null) {
```

11 - Comparaison entre deux objets

Avant d'arriver à ce point-là, j'étais bloqué sur le résultats de cette comparaison. Sur la console, le cmdlet liste les points en indiquant dans lequel des deux objets tel ou tel entrée n'est pas présente. Ce que je ne savais pas et que j'ai appris durant mes recherches, c'est qu'une condition qui attend un résultat True ou False doit utiliser une variable \$true ou \$false. C'est lors de la discussion avec M. Olivier Clerc que la solution m'est venue, en connaissant cette histoire de variable, concernant les résultats null. Il fallait utiliser une variable \$null.

Pour l'enregistrement des résultats, j'ai dû poser une question sur un forum pour obtenir une solution. (Le lien du sujet est dans la bibliographie). Le résultat suivant m'a été proposé, et les résultats que j'obtiens par cette solution convient à ce dont j'ai besoin.

```
$ACLs = (Get-Acl $item.FullName).Access
$results += $ACLs `
| Add-Member -Name Path -Value $item.FullName -MemberType NoteProperty -PassThru `
| Select-Object Path, AccessControlType, FileSystemRights, IdentityReference
```

12 - Ajout d'un résultat

Cette partie de script est placée dans un boucle foreach, dont on prend une variable \$item pour un objet de la liste de résultats. La seconde partie du code ajoute une entrée aux résultats contenu dans la variable \$ACLs, entrée qui contient le chemin, le type de contrôle, le niveau de contrôle et les utilisateurs/groupes concernés. L'Array \$results est l'endroit où toutes les informations seront stockées avant d'être exportées.

Le résultat de ce script est un fichier Excel avec un nombre de lignes équivalent au nombre d'entrées utilisateurs/groupes pour ce fichier/dossier.

Dans le script suivant, EraseUseless.ps1, le contenu du fichier Excel est récupéré grâce à la fonction Import-Excel du module du même nom.

```
$fileContent = import-excel "..\Rapports\$year\$month\results1_$(get-date -f dd-MM-yyyy).xlsx"
```

13 - Importation de données depuis le fichier Excel

Grâce à une boucle foreach une nouvelle fois, le contenu des lignes est comparé. Si les données de la ligne précédente sont les mêmes que celles de la ligne actuelle, le chemin et le niveau de contrôle sont effacés. Il est important de ne pas supprimer le contenu d'une première ligne, qui donne les informations pour toutes celles qui suivent.

Pour faire ceci, une première condition regarde si le chemin de la ligne d'avant ne sont pas égales à celui de l'actuelle, cela signifie qu'il s'agit de la première ligne pour une nouvelle entrée. Si c'est le cas, la valeur 1 est attribué à une variable.

La condition suivante compare les niveaux de contrôle ou si la valeur est égale à 1. Si c'est le cas, la variable retrouve une valeur de 0 et ne fait pas d'autre action qu'enregistrer les données pour le tour d'après. Sinon, les niveaux de contrôle sont remplacés par un espace vide.

```
foreach ($item in $fileContent) {
    # Si le chemin enregistré n'est pas égal à celui de l'objet actuel, l'enregistrer
    # et afficher les informations normalement
    if ($storedPath -ne $item.Path) {
        $storedPath = $item.Path
        $storedFileSystemRights = $item.FileSystemRights
        $firstLine = "1"
    } else {
        # Sinon, effacer le chemin
        $item.Path = ""
    }
    # Si les types de droits comparés ne sont pas égaux,
    # ou s'il s'agit de la première ligne d'un fichier,
    # afficher les informations normalement
    if ($storedFileSystemRights -ne $item.FileSystemRights -or $firstLine -eq "1") {
        $storedFileSystemRights = $item.FileSystemRights
        $firstLine = "0"
    } elseif ($storedFileSystemRights -eq $item.FileSystemRights) {
        # Sinon, les effacer
        $item.FileSystemRights = ""
    }
}
```

14 - Modifications des entrées en fonction der de celles d'au-dessus

Après ces changements, les données sont réexportées vers un nouveau fichier Excel et le prochain script est appelé comme montré ci-dessous.

```
Invoke-Expression "..\Scripts\Cleaning.ps1"
```

15 - Appel du script suivant

Le troisième script, du nom de Cleaning.ps1, récupère les données du second fichier Excel, et uniquement les lignes où il y a des données pour l'enregistrer dans une variable. Le module Import-Excel permet de faire ceci grâce à l'option -DataOnly comme indiqué sur la figure suivante.

```
Import-Excel "..\Rapports\$year\$month\results2_$(get-date -f dd-MM-yyyy).xlsx" -DataOnly
```

16 - Récupération des données uniquement

Pour une question de visibilité, la variable n'est pas incluse dans la figure.

Chaque entrée est ajoutée dans un nouveau Array qui sera exporté à la fin du recueil des données. Cette exportation est différente des autres puisqu'elle inclut le nom de la feuille également. C'est encore une fonctionnalité proposée par le module Import-Excel.

```
$array | Export-Excel "..\Rapports\$year\$month\results_$(get-date -f dd-MM-yyyy).xlsx" `
-Worksheetname "Rapport" -AutoSize
```

17 - Exportation Excel avec nom de feuille

Après l'enregistrement des données, les deux feuilles Excel précédentes sont supprimées, puis le dernier script est appelé.

```
Remove-Item "..\Rapports\$year\$month\results1_$(get-date -f dd-MM-yyyy).xlsx"
Remove-Item "..\Rapports\$year\$month\results2_$(get-date -f dd-MM-yyyy).xlsx"
Invoke-Expression "..\Scripts\GetGroupsArborescence.ps1"
```

18 - Suppression des fichiers Excel + prochain script

La création de ce script a posé quelques problèmes. J'ai relativement rapidement trouvé comment exporter les utilisateurs de tous les groupes présents de l'Active Directory, mais il y a de nombreux résultats qui ne sont pas utiles. J'ai plutôt voulu récupérer les groupes listés dans les droits de l'arborescence, et suis resté bloqué sur cette étape plusieurs jours. J'ai finalement trouvé une solution qui utilise un nouvel objet pour stocker les différentes variables.

Le premier point de ce script est de récupérer les groupes dans les droits de l'arborescence, et de fournir une liste de ces groupes. Pour ce faire, je récupère les droits ACL pour chaque entrée de l'arborescence. Si une entrée n'est pas stockée dans le tableau prévu à cet effet, elle y est ajoutée, sinon il ne se passe rien.

```
$groups = @()
foreach ($line in $arborescence) {
    $acls = (get-ACL $line.FullName).Access
    foreach ($a in $acls) {
        try {
            # Sépare la ligne en deux par un \ et sélectionne le
            #second bloc de résultat avec [1], soit le nom du groupe
            #ou de l'utilisateur
            $entry = ($a.identityReference.toString().split("\")[1])
            # Enregistre l'entrée s'il n'est pas encore présent dans
            #l'Array $groups
            if (!$groups -contains $entry) {
                $groups += $entry
            }
        } catch {
            continue
        }
    }
}
```

19 - Récupération des groupes dans l'arborescence

Cette liste est ensuite utilisée pour récupérer les informations des utilisateurs et des groupes. Chaque entrée de la liste est utilisée pour trouver les membres de ce groupe. Toutes les entrées ne sont pas des groupes, et cela produit des erreurs, qui n'influent pas sur le fonctionnement du système.

La boucle foreach débouche sur une variable avec comme données ce qui est récupéré par l'objet.


```
# Array pour stocker les différents utilisateurs
$groupsOutput = @()
$groupsOutput = foreach ($group in $groups) {
    $results = Get-ADGroupMember -Identity $group.ToString() `
    | Get-ADUser -Properties displayname, ObjectClass, name, samAccountName
    # Pour chaque entrée de membres de groupes dans $results,
    # récupérer les informations sur les utilisateurs
    # et stockage de ces informations dans un objet pour exporter le tout
    foreach ($result in $results) {
        New-Object PSObject -Property @{
            GroupeName = $group
            Username = $result.Name
            SAN = $result.SamAccountName
            DisplayName = $result.DisplayName
        }
    }
}
```

20 - Enregistrement des données par l'objet

Après ceci, les données sont exportées une dernière fois vers le fichier créé par le script précédent, mais sur une feuille différente.

Lecture des résultats

Les résultats qui sont recherchés par le TPI sont FullControl, Read, Write, List, mais ils ne sont pas tous affichés de cette manière sur le résultat. Il faut également savoir qu'il y a 14 droits différents dans les permissions avancées, mais les droits les plus communs étant ceux configurés via l'interface simple, je vais indiquer ceux-ci.

- FullControl = FullControl
- Read = ReadAndExecute
- Write = Write
- List = ReadData

Descriptions des tests effectués

Des tests ont été effectués comme mentionné dans la rubrique Stratégie de test.

Les premiers tests consistaient à vérifier la bonne récupération des droits et de vérifier leur véracité. Par les résultats obtenus sur la console, affichés comme sur la figure suivante, j'ai pu voir que les données étaient correctes. Des droits étaient modifiés sur certains dossiers afin de vérifier que les données étaient correctement modifiées dans les résultats, ce qui était le cas.

FileSystemRights	AccessControlType	IdentityReference
ReadAndExecute, Synchronize	Allow	TPI\nvr
FullControl	Allow	NT AUTHORITY\SYSTEM
FullControl	Allow	BUILTIN\Administrators
FullControl	Allow	TPI\Administrator

21 - Résultats obtenus sur la console

Mais la partie suivante qui consistait à exporter les résultats dans un fichier CSV à ce moment-là posait plus de problème puisque les résultats obtenus ne correspondaient nullement à ce qui était prévu, comme montrent ces extraits de résultats de tests obtenus le 16 mars. L'exportation des données d'un seul fichier ne pose pas de problème, c'est lorsqu'il faut exporter une arborescence que cela pose problème.


```
System.Security.AccessControl.DirectorySecurity
System.Security.AccessControl.DirectorySecurity
System.Security.AccessControl.FileSecurity
System.Security.AccessControl.FileSecurity
```

```
@{Access=System.Security.AccessControl.AuthorizationRuleCollection}
@{Access=System.Security.AccessControl.AuthorizationRuleCollection}
@{Access=System.Security.AccessControl.AuthorizationRuleCollection}
@{Access=System.Security.AccessControl.AuthorizationRuleCollection}
```

22 - Résultats exportations du 16 mars

Suite à ça, certaines potentielles solutions sont données, mais l'exportation des données reste toujours un problème lorsqu'il y a plus d'un fichier à exporter. Les données récupérées suite aux tests effectués le 20 mars sont visibles dans la figure suivante.

#TYPE System.Security.AccessControl.FileSecurity							
PSPPath,"PSParentPath","PSChildName","PSDrive","PSProvider","CentralAccessPolicyId","CentralAccess							
Microsoft.PowerShell.Core\FileSystem::C:\Users\Administrator\Desktop\test\testing.txt,"Microsoft.Po							
NT AUTHORITY\SYSTEM Allow FullControl							
BUILTIN\Administrators Allow FullControl							
TPI\Administrator A\0x1200a9			S-1-5-21-5 ID	FA			
Microsoft.PowerShell.Core\FileSystem::C:\Users\Administrator\Desktop\test\txt.txt,"Microsoft.Power							
BUILTIN\Administrators Allow FullControl							
TPI\Adminir ID	FA		SY)(A ID	FA			

23 - Résultats d'exportation du 20 mars

C'est suite à une réponse obtenue sur un forum que les résultats obtenus correspondent plus à un résultat attendu, visibles sur la figure suivante.

C:\Users\administrator\Desktop\New folder,"FullControl","Allow","NT AUTHORITY\SYSTEM","True","ContainerInherit, ObjectInherit","None"							
,,"FullControl","Allow","BUILTIN\Administrators","True","ContainerInherit, ObjectInherit","None"							
,,"FullControl","Allow","TPI\Administrator","True","ContainerInherit, ObjectInherit","None"							
C:\Users\administrator\Desktop\test,"FullControl","Allow","NT AUTHORITY\SYSTEM","True","ContainerInherit, ObjectInherit","None"							
,,"FullControl","Allow","BUILTIN\Administrators","True","ContainerInherit, ObjectInherit","None"							
,,"FullControl","Allow","TPI\Administrator","True","ContainerInherit, ObjectInherit","None"							

24 - Résultats obtenus au 21 mars

Suite à l'ajout du module Import-Excel, la mise ne page n'est plus un problème. Les résultats se mettent en page seul. Le test suivant a servi à vérifier que les données récupérées soient correctes. Pour faire ça, différents droits ont été ajoutés sur certains fichiers/dossiers, et les données affichées sur le résultat du script ont été vérifiées, sachant quel droit avait été modifié. Sur la figure suivante, nous pouvons voir certaines de ces modifications correctement récupérées par le script, mises en avant par un coloriage gris.

Path	FileSystemRights	AccessCor	IdentityReference
C:\Users\administrator\desktop	FullControl	Allow	NT AUTHORITY\SYSTEM
C:\Users\administrator\desktop	FullControl	Allow	BUILTIN\Administrators
C:\Users\administrator\desktop	FullControl	Allow	TPI\Administrator
C:\Users\administrator\desktop\New_folder	FullControl	Allow	NT AUTHORITY\SYSTEM
C:\Users\administrator\desktop\New_folder	ReadData, Execute	Allow	BUILTIN\Administrators
C:\Users\administrator\desktop\New_folder	FullControl	Allow	TPI\Administrator
C:\Users\administrator\desktop\New_folder	ReadAndExecute, !	Allow	TPI\Administration_tpi
C:\Users\administrator\desktop\test	FullControl	Allow	NT AUTHORITY\SYSTEM
C:\Users\administrator\desktop\test	FullControl	Allow	BUILTIN\Administrators
C:\Users\administrator\desktop\test	FullControl	Allow	TPI\Administrator
C:\Users\administrator\desktop\AD.txt	FullControl	Deny	TPI\nvr
C:\Users\administrator\desktop\AD.txt	FullControl	Allow	NT AUTHORITY\SYSTEM

25 - Résultats suite à modifications des droits

Les tests de gestion de l'affichage n'ont pas de résultats visuels, car le fichier utilisé était écrasé au fur et à mesure de l'avancement du projet. Mais comme dans les autres cas, des données étaient modifiées afin de vérifier le bon fonctionnement du projet après les modifications, ce qui était le cas.

Un nouveau test a été effectué avec une très grande arborescence. Le fichier contenant 23'000 ligne Excel, et ne peut être montré pour cause de confidentialité. Ce test a permis de voir que le résultat affiché n'est pas des plus optimale avec un grand nombre de fichiers, même si l'utilisateur peut naviguer relativement facilement dans la liste de résultats puisqu'elle est triée par chemin.

Le deniers test a consisté à vérifier les résultats obtenus par le script de récupération des utilisateurs. Pour se faire, un premier rapport est effectué, avec une vérification des groupes utilisés dans l'arborescence ainsi que les utilisateurs y faisant part. Une modification des membres des groupes est faite et un second rapport est effectué, puis les résultats comparés une nouvelle fois. Les modifications sont correctement prises en compte, et affichées dans le résultat.

Parmi les 6 figures utilisées pour montrer les résultats de test, seul la figure 7 (n'étant pas dans un fichier) et la figure 11 (C'est une reproduction du résultat), ils sont tous disponibles dans des fichiers disponibles dans le dossier fournis, dans le dossiers resultsTest.

Erreurs restantes

Un système de log d'erreurs a été mis en place durant le projet, et celui-ci récupère toutes les erreurs qui se produisent durant le déroulement du script. Or, certaines erreurs sont connues mais sont quand même listées dans ce fichier de log. Ce problème n'a pas de conséquences sur l'utilisation du produit, mais pour trouver les véritables erreurs, tels que « l'accès à ce dossier n'est pas possible », il faut aller naviguer à travers un fichier d'erreur contenant des erreurs superflues. Pour résoudre ce problème, je ferais un tri une fois le fichier de log terminé pour modifier le message en un message plus simple, voir même le supprimer du fichier log. Pour ceci, une recherche d'un mot clef permettrait de sélectionner un message et avoir un comportement particulier suivant la signification du message.

Dossiers d'archivage

Le dossier d'archivage contient

- Un dossier Logs, contenant certains fichiers d'erreurs.
- Un dossier Module, contenant le module additionnel, Import-Excel
- Un dossier Rapports, avec quelques rapports établis
- Un dossier resultsTest qui contient les résultats des tests mentionnés plus tôt
- Un dossier Scripts, contenant les 4 scripts du projet
- Un dossier Settings, où le fichier settings.txt est enregistré
- Le fichier ReadMe.txt, avec le contenu mentionné plus tôt

Mise en service

Installation du produit

L'installation du produit doit être effectuée par l'administrateur, qui a normalement les droits d'exécution de scripts. (Mais si ce n'est pas le cas, se référer à la section Réalisation)

Ensuite, il faut installer le module supplémentaire Import-Excel. Pour se faire, il faut lancer le fichier *Modules\ImportExcel-master\InstallModule.ps1* avec PowerShell pour qu'il soit correctement installé. Le module entier est dans le dossier de projet.

Il faut sélectionner le chemin du dossier qui doit être rapporté dans le fichier *Settings\settings.txt*.

Je conseille de modifier le script une première fois, et de lancer le script depuis la console Microsoft PowerShell ISE pour vérifier qu'aucune erreur ne se produise. (Certains fonctionnements de chemin de fichier n'ont pas bien accepté le changement de la zone de test et la zone de production. Il est possible que le chemin nécessite un point supplémentaire ou en moins.)

Ensuite, configurer le script dans Task Scheduler pour qu'il soit lancé toutes les nuits lorsque personne n'a besoin des ressources. 00h30 est une bonne heure, pour éviter que le changement d'heures puisse poser problème par la suite.

Dans les premiers jours, ne pas hésiter à regarder les fichiers de logs pour voir si tout se passe bien ou si certains dossiers n'autorisent pas le passage du script par manque de droits.

Liste de documents fournis

Les documents fournis au client sont les suivants :

- Le rapport du projet
- Une archive Zip contenant le projet de scripts, nettoyée de tout fichier de résultats

Conclusion

Objectifs atteints / non-atteints

Parmi les tâches listées dans le cahier des charges, il me semble toutes les avoir effectuées. La mise en service également, même si son importance était moindre à mes yeux puisque la partie importante du projet était plus les scripts que l'installation d'une machine servant à développer l'outil au client.

Mais par rapport aux points techniques évalués, je ne pense pas que tous les points soient entièrement satisfaits. Le script fonctionne, mais certaines erreurs inconnues apparaissent malgré tout. L'éventail d'erreurs qui peut se produire est large, mais je n'ai pas pu implémenter un système de modification de message d'erreur parmi les messages d'erreurs les plus connus comme je l'avais prévu initialement s'il restait du temps.

J'estime que le résultat est utilisable et relativement bien lisible, même s'il faut aller chercher l'information dans la liste du rapport, et les utilisateurs présents dans chaque groupe sur une autre feuille.

Problèmes rencontrés et résolus

Affichage des droits selon arborescence : (jeudi 16 mars - Lundi 20 mars) - Résolu

Je n'arrivais pas exporter le résultat de la récupération de droits avec Get-ChildItem et Get-Acl. Un sujet a été fait sur le forum PowerShell de Microsoft afin d'obtenir de l'aide. ([lien](#) 11) Là où moi j'enregistrais les résultats dans un tableau simple, la personne m'ayant répondu l'a fait dans un Array. Le résultat fonctionne et les données sont exploitables.

Récupération des utilisateurs et groupes : (mardi 21 mars – lundi 03 avril) – Résolu

Je n'arrivais pas exporter les résultats récupérés pour n'afficher que les groupes utilisés dans l'arborescence. Au final, une solution que j'avais repéré durant mes recherches est celle qui a pu venir à mon secours. Les données du groupe et de chaque utilisateur qu'il contient sont enregistrées dans un objet pour ensuite être exporté vers le fichier Excel.

Affichage plus clair des résultats : (Vendredi 24 mars) – Résolu

Je souhaite afficher uniquement le chemin en cas de droits semblables au dossier parent. Mais les règles IF que j'essaie ne fonctionnent pas. Après discussion avec mon maître de stage durant laquelle j'ai trouvé la solution, c'est un problème résolu.

Une comparaison des droits de l'entrée avec son dossier parent est effectuée. S'ils sont semblables, uniquement son chemin est affiché, sinon sont affichés le chemin, le type de contrôle (Allow/deny), le niveau de contrôle (FullControl, Read, Write, ...) et les utilisateurs/groupes ayant ces droits.

Points positifs/négatifs

Le point positif majeur selon moi est que les données du résultat, même si très nombreuses, sont assez facilement lisibles. Si les droits d'un dossier en particulier doit être recherché, la fonction Rechercher sur Excel permet de rapidement naviguer à travers les différentes lignes pour trouver le dossier concerné.

Les points négatifs les plus importants à mes yeux sont la mauvaise gestion du temps alloué aux tâches et les objectifs évalués partiellement atteints. Je pense que le temps nécessaire à la recherche d'informations et de solutions à des problèmes qui peuvent être rapidement résolus avec plus d'expérience en développement m'ont beaucoup ralenti dans l'avancement du projet.

De plus, les décalages suite au blocage sur l’affichage des groupes/utilisateurs, et la détermination que j’ai eu à vouloir trouver une solution ont fait que le temps alloué à la documentation et au journal de travail n’était pas suffisant, et j’ai dû rattraper ce retard à la fin en supprimant certains points que je voulais essayer de réaliser.

Difficultés particulières

La principale difficulté que j’ai eu avec le développement des scripts est l’affichage des données, qui a occupé près de 6 jours de recherches et tests pour trouver une solution.

Suites possibles pour le projet (évolutions & améliorations)

Je vois de nombreuses améliorations que j’apporterais au projet si j’en ai l’opportunité ainsi que le temps. Un exemple parmi d’autres est l’affichage actuel qui ne me convient pas lorsque la quantité d’informations est trop importante. Je souhaiterais effectuer un affichage qui s’indente pour bien mieux détecter à quel dossier un fichier appartient. Je souhaiterais également, sans avoir le stress d’une date butoir, mettre en place un système ressemblant à des sauvegardes incrémentales qui pourraient sortir les différences entre les deux derniers rapports.

Bilan de planification

Le projet comptait 120 heures de travail, mais le temps pris pour compléter et relire la documentation, le projet s’approche des 125 heures environ. En passant en revue les différentes tâches de mon projet, j’ai pu en ressortir différents points qui semblent importants d’être pris en compte.

Le manque de pratique dans le développement m’a fait perdre du temps, avec un temps plus long pour la compréhension de certains points par exemple.

La tâche ayant causé le retard le plus important est la documentation, avec le double d’heures passées sur la documentation, soit 45.5 heure au lieu de 27. Cette grande différence est certainement dû à une mauvaise planification initiale.

La seconde tâche qui a demandé beaucoup de temps est l’affichage des groupes et des utilisateurs, 16 heures cumulées pour les deux tâches concernant les groupes et utilisateurs à la place de 6 comme planifié initialement. Cette différence est due à l’échec des différentes tentatives et des différents tests que j’effectuais pour arriver au résultat qui me convenait.

Bilan personnel

Ce projet m’a plu, j’ai apprécié retrouver le développement et devoir trouver un moyen pour surmonter des obstacles, même s’il faut de l’aide par moments. Je suis un peu déçu de ne pas avoir pu répondre à tous les objectifs que je m’étais fixé moi, mais je pense que je visais peut-être un peu trop haut sachant que je n’avais que des connaissances de bases en PowerShell et qu’il me fallait apprendre énormément.

Avec ce projet, j’ai appris que le temps nécessaire pour faire un rapport de qualité est important, et le temps passé à l’analyse d’un projet doit être plus conséquent afin de pouvoir correctement attribué les tâches à l’avance.

Remerciements

Je remercie le client, et collègue, Jérémie Heiniger qui a proposé cette idée pour mon TPI.

Annexes

Bibliographie

1. Créé un domaine : <https://social.technet.microsoft.com/wiki/contents/articles/12370.windows-server-2012-set-up-your-first-domain-controller-step-by-step.aspx>
2. Ajouter un domaine : <https://blogs.technet.microsoft.com/canitpro/2013/05/05/step-by-step-adding-a-windows-server-2012-domain-controller-to-an-existing-windows-server-2003-network/>
3. Get-ChildItem : http://www.computerperformance.co.uk/powershell/powershell_file_gci_include.htm
4. Export-Csv : <https://blogs.technet.microsoft.com/heyscriptingguy/2014/02/04/use-powershell-to-create-csv-file-to-open-in-excel/>
5. Get-Acl, droits étendus : <http://stackoverflow.com/questions/28119096/how-to-list-folder-permissions-located-on-a-different-server>
6. Trier objet : <https://blogs.technet.microsoft.com/heyscriptingguy/2012/01/10/order-your-output-by-easily-sorting-objects-in-powershell/>
7. Export-Csv avec un objet : <https://blogs.technet.microsoft.com/heyscriptingguy/2011/09/23/use-powershell-to-work-with-csv-formatted-text/>
8. Ajouter Membre dans Array : <https://msdn.microsoft.com/en-us/powershell/reference/5.1/microsoft.powershell.utility/add-member>
9. Sujet sur le forum Microsoft : <https://social.technet.microsoft.com/Forums/scriptcenter/en-US/2aa95771-f24a-4b54-b24d-40be952acf6e/getacl-from-a-getchilditem-list?forum=ITCG>
10. Supprimer les espaces blancs : <https://blogs.technet.microsoft.com/heyscriptingguy/2014/07/18/trim-your-strings-with-powershell/>
11. Vérifier l'existence d'un fichier : <https://msdn.microsoft.com/en-us/powershell/reference/5.1/microsoft.powershell.management/test-path?f=255&MSPPError=-2147217396>
12. Time-Stamp : <http://stackoverflow.com/questions/1954203/timestamp-on-file-name-using-powershell>
13. Séparer un String par un caractère : <https://blogs.technet.microsoft.com/heyscriptingguy/2014/07/17/using-the-split-method-in-powershell/>
14. Sélectionner String spécifique avec un caractère : <https://blogs.technet.microsoft.com/heyscriptingguy/2014/07/17/using-the-split-method-in-powershell/>
15. Fonctionnement des conditions IF : <https://4sysops.com/archives/powershell-comparison-operators-eq-lt-gt-contains-like-match/>
16. Trouver ou remplacer des guillemets, ou autres caractères : <http://stackoverflow.com/questions/14816571/powershell-remove-or-replace-quote-marks-from-variable>
17. Gestion des erreurs : <https://blogs.msdn.microsoft.com/kebab/2013/06/09/an-introduction-to-error-handling-in-powershell/>
18. Invoquer script suivant : <https://technet.microsoft.com/en-us/library/ee176880.aspx>
19. Comparer des objets pour trouver des différences : <https://technet.microsoft.com/en-us/library/ee156812.aspx?f=255&MSPPError=-2147217396>
20. Récupérer groupes sur l'AD : <https://technet.microsoft.com/en-us/library/ee617196.aspx?f=255&MSPPError=-2147217396>
21. Récupérer les membres d'un groupe : <https://technet.microsoft.com/en-us/library/ee617193.aspx>
22. Enregistrer les membres d'un groupe dans un Array : <http://www.jonathanmedd.net/2014/01/adding-and-removing-items-from-a-powershell-array.html>
23. Import-Excel GitHub : <https://github.com/dfinke/ImportExcel>
24. Test si chemin mène à fichier ou dossier : <https://blogs.technet.microsoft.com/heyscriptingguy/2014/08/31/powertip-using-powershell-to-determine-if-path-is-to-file-or-folder/>

Journal de travail

Le journal de travail est à retrouver en annexe

Archive du projet

Le projet peut être retrouvé sur le lien GitHub suivant : <https://github.com/NealvanRooij/TPI2017>