Chapitre 8
La sécurité

1. La sécurité : pour qui ? Pourquoi ?

L'arrivée des réseaux locaux et d'Internet a changé beaucoup de choses dans la manière de protéger son PC. Il ne suffit plus d'attacher son disque dur au radiateur et de fermer la porte du bureau le soir pour ne pas se faire voler ou pirater des données. Maintenant, protéger son poste de travail est devenu essentiel pour ne pas faire les frais d'intrusions ou de malveillances.

Mais alors contre qui se prémunir ? Eh bien, contre tout ce qui bouge… et même ce qui ne bouge pas. En effet, que ce soient des programmes malveillants, des utilisateurs mal intentionnés, voire des utilisateurs inexpérimentés, tous peuvent être considérés comme une menace. C'est pour cela que vous devez verrouiller votre système en établissant des règles de sécurité, en les appliquant et en vous assurant que les autres en font tout autant.

2. Les risques liés au scripting

Vous allez vite deviner que ce qui fait la force du scripting en fait aussi sa faiblesse. La facilité avec laquelle vous pouvez tout faire, soit en cliquant sur un script, soit en l'exécutant depuis la fenêtre de commande, peut vous mettre dans l'embarras si vous ne faites pas attention.

Imaginez un script de logon qui dès l'ouverture de la session la verrouille aussitôt ! Alors oui, c'est sympa entre copains, mais en entreprise, nous doutons que cela soit de bon ton. Plus grave encore, un script provenant d'une personne mal intentionnée ou vraiment peu expérimentée en PowerShell (dans ce cas, nous vous conseillons de lui

acheter un exemplaire de ce livre…) peut parfaitement vous bloquer des comptes utilisateurs dans Active Directory, vous formater un disque, vous faire rebooter sans cesse. Enfin, vous l'avez compris, un script peut tout faire. Car même si aujourd'hui des alertes sont remontées jusqu'à l'utilisateur pour le prévenir de l'exécution d'un script, elles ne sont pas capables de déterminer à l'avance si un script est nuisible au bon fonctionnement du système.

Les risques liés au scripting se résument à une histoire de compromis : soit vous empêchez toute exécution de scripts, c'est-à-dire encourir le risque de vous « pourrir la vie » à faire et à refaire des tâches basiques et souvent ingrates, soit vous choisissez d'ouvrir votre système à PowerShell, en prenant soin de prendre les précautions qui s'imposent.

Mais ne vous laissez pas démoraliser car même si l'exécution de scripts vous expose à certains problèmes de sécurité, PowerShell se dote de certains concepts qui font de lui l'un des langages de script les plus sûrs. Il ne faut pas non plus oublier qu'en cas de problème de sécurité, c'est l'image tout entière de Microsoft qui en pâtit…

3. Optimiser la sécurité PowerShell

3.1 La sécurité PowerShell par défaut

Vous l'avez compris, la sécurité est une chose très importante, surtout dans le domaine du scripting. C'est pour cela que les créateurs de PowerShell ont inclus deux règles de sécurités par défaut.

Des fichiers ps1 associés au bloc-notes

L'extension « **`.ps1`** » des scripts PowerShell, est par défaut associée à l'éditeur de texte bloc-notes (ou Notepad). Ce procédé permet d'éviter de lancer des scripts potentiellement dangereux sur une mauvaise manipulation. Le bloc-notes est certes un éditeur un peu classique, mais a le double avantage d'être inoffensif et de ne pas bloquer l'exécution d'un script lorsque celui-ci est ouvert avec l'éditeur. Vous remarquerez cependant que l'édition (**Clic droit** + **Modifier**) des fichiers **`.ps1`** est associée à l'éditeur ISE.

Remarque

Ce type de sécurité n'était pas mis en place avec les scripts VBS dont l'ouverture était directement associée au Windows Script Host.

Une stratégie d'exécution restreinte

La seconde barrière de sécurité est l'application de la stratégie d'exécution « **Restricted** » par défaut (cf. section suivante : Les stratégies d'exécution).

Cette stratégie est la plus restrictive. C'est-à-dire qu'elle bloque systématiquement l'exécution de tout script. Seules les commandes tapées dans le shell seront exécutées. Pour remédier à l'inexécution des scripts, PowerShell requiert que l'utilisateur change le mode d'exécution avec la commande **Set-ExecutionPolicy <mode d'exécution>**. Mais pour ce faire il faut être administrateur de la machine.

Remarque

Peut-être comprenez-vous mieux pourquoi l'utilisation de PowerShell sur vos machines ne constitue pas un accroissement des risques, dans la mesure où certaines règles sont bien respectées.

3.2 Les stratégies d'exécution

PowerShell intègre un concept de sécurité que l'on appelle les stratégies d'exécution (execution policies) pour qu'un script non autorisé ne puisse pas s'exécuter à l'insu de l'utilisateur. Il existe sept configurations possibles : **Restricted**, **RemoteSigned**, **AllSigned**, **UnRestricted**, **Bypass**, **Default** et **Undefined**. À chacune d'elles correspond un niveau d'autorisation d'exécution de scripts particulier, et vous pourrez être amené à en changer en fonction de la stratégie que vous souhaitez appliquer.

3.2.1 Les différentes stratégies d'exécution

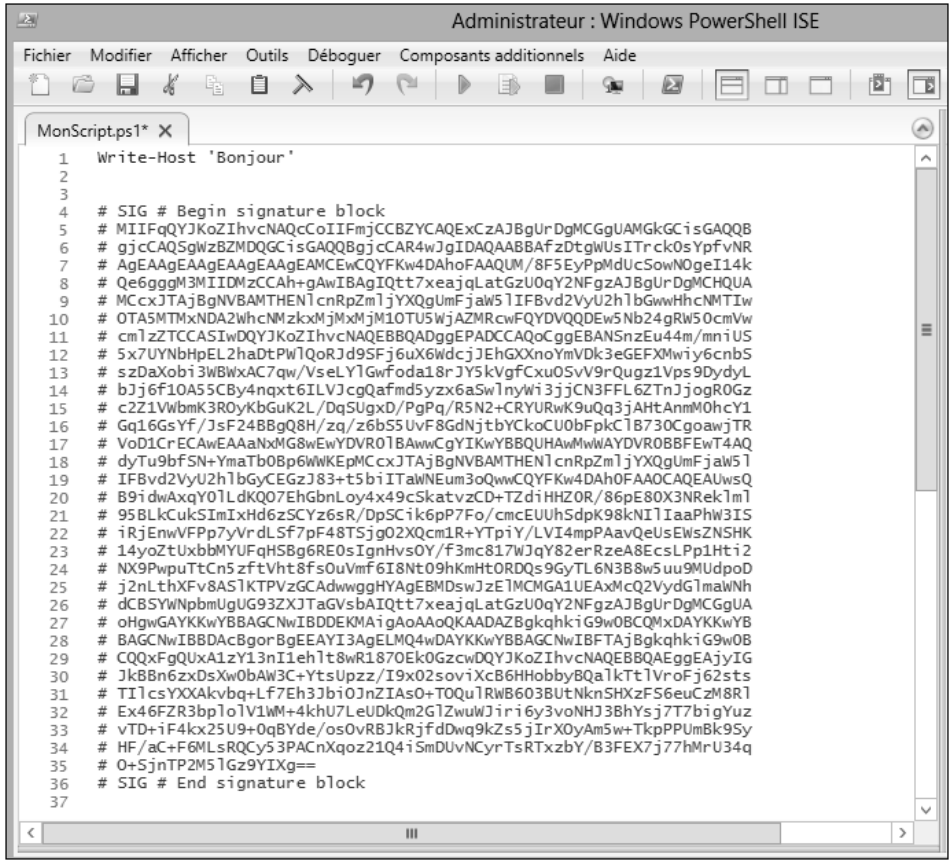
Restricted : c'est la stratégie la plus restrictive, et c'est aussi la stratégie par défaut. Elle ne permet pas l'exécution de script mais autorise uniquement les instructions en ligne de commande tapées dans la console. Cette stratégie peut être considérée comme la plus radicale étant donné qu'elle protège contre l'exécution involontaire des fichiers « **.ps1** ».

Lors d'une tentative d'exécution de script avec cette stratégie, un message de ce type est affiché dans la console :

```
Impossible de charger le fichier C:\script.ps1,
car l'exécution de scripts est désactivée sur ce système.
```

Comme cette stratégie est celle définie par défaut lors de l'installation de PowerShell, il vous faudra donc la changer pour l'exécution de votre premier script.

AllSigned : c'est la stratégie permettant l'exécution de script la plus « sûre ». Elle autorise uniquement l'exécution des scripts signés. Un script signé est un script comportant une signature numérique comme celle présentée sur la figure ci-dessous.



Exemple de script signé

Avec la stratégie **AllSigned**, l'exécution de scripts signés nécessite que vous soyez en possession des certificats correspondants (cf. section Signature des scripts).

RemoteSigned : cette stratégie se rapporte à **AllSigned** à la différence près que seuls les scripts ayant une origine autre que locale nécessitent une signature. Par conséquent, cela signifie que tous vos scripts créés localement peuvent être exécutés sans être signés.

Si vous essayez d'exécuter un script provenant d'Internet sans que celui-ci soit signé, le message suivant sera affiché dans la console.

```
Impossible de charger le fichier C:\script.ps1.
Le fichier C:\script.ps1 n'est pas signé numériquement.
Le script ne sera pas exécuté sur le système.
```

Remarque

Vous vous demandez sûrement comment PowerShell fait pour savoir que notre script provient d'Internet ? Réponse : Grâce aux « Alternate Data Streams » qui sont implémentés sous forme de flux cachés depuis des applications de communication telles que Microsoft Outlook, Internet Explorer, Outlook Express et Windows Messenger (voir partie traitant des Alternate Data Streams). En gros, lorsqu'un script est téléchargé à partir d'un client Microsoft, la provenance de celui-ci lui est attachée.

`Unrestricted` : c'est la stratégie la moins contraignante, et par conséquent la moins sûre. Avec elle, tout script, peu importe son origine, peut être exécuté sans demande de signature. C'est donc la stratégie où le risque d'exécuter des scripts malveillants est le plus élevé.

Cette stratégie affiche tout de même un avertissement lorsqu'un script téléchargé d'Internet tente d'être exécuté.

```
PS > .\script.ps1

Avertissement de sécurité
N'exécutez que des scripts que vous approuvez. Bien que les scripts
en provenance d'Internet puissent être utiles, ce
script est susceptible d'endommager votre ordinateur.
Voulez-vous exécuter C:\script.ps1 ?
[N] Ne pas exécuter [O] Exécuter une fois
[S] Suspendre [?] Aide (la valeur par défaut est « N ») :
```

`Bypass` : rien n'est bloqué et aucun message d'avertissement ne s'affiche.

`Undefined` : pas de stratégie d'exécution définie dans l'étendue courante. Si toutes les stratégies d'exécution de toutes les étendues sont non définies alors la stratégie effective appliquée sera la stratégie **`Restricted`**.

`Default` : positionne la stratégie par défaut, à savoir **`Restricted`**.

Remarque

Microsoft a mis en place ces mécanismes afin de tenter de limiter les risques liés à l'exécution de scripts provenant de l'extérieur de l'entreprise et donc potentiellement malveillants. La configuration par défaut et permet d'atteindre cet objectif mais elle ne garantit en aucun cas une sécurité parfaite.

3.2.2 Les étendues des stratégies d'exécution

PowerShell permet de gérer l'étendue des stratégies. L'ordre d'application est le suivant :

– **Étendue `Process`** : la stratégie d'exécution n'affecte que la session courante (processus Windows PowerShell). La valeur affectée à l'étendue **`Process`** est stockée en mémoire uniquement ; elle n'est donc pas conservée lors de la fermeture de la session PowerShell.

– **Étendue `CurrentUser`** : la stratégie d'exécution appliquée à l'étendue **`CurrentUser`** n'affecte que l'utilisateur courant. Le type de stratégie est stocké de façon permanente dans la partie du registre **`HKEY_CURRENT_USER`**.

– **Étendue `LocalMachine`** : la stratégie d'exécution appliquée à l'étendue **`LocalMachine`** affecte tous les utilisateurs de la machine. Le type de stratégie est stocké de façon permanente dans la partie du registre **`HKEY_LOCAL_MACHINE`**.

La stratégie ayant une priorité 1 est plus propriétaire que celle ayant une priorité 3. Par conséquent, si l'étendue **`LocalMachine`** est plus restrictive que l'étendue **`Process`**, la stratégie qui s'appliquera sera quand même la stratégie de l'étendue **`Process`**. À moins que cette dernière soit de type **`Undefined`** auquel cas PowerShell appliquera la stratégie de l'étendue **`CurrentUser`** puis tentera d'appliquer la stratégie **`LocalMachine`**.

À noter que l'étendue **`LocalMachine`** est celle par défaut lorsque l'on applique une stratégie d'exécution sans préciser d'étendue particulière.

3.2.3 Identifier la stratégie d'exécution courante

La stratégie d'exécution courante s'obtient avec la commandelette **`Get-ExecutionPolicy`**.

Exemple

```
PS > Get-ExecutionPolicy
Restricted
```

Avec cette commande, nous bénéficions du commutateur **`-List`**. Grâce à lui, nous allons savoir quelles stratégies s'appliquent à nos étendues.

Par exemple

```
PS > Get-ExecutionPolicy -List

        Scope ExecutionPolicy
        ----- ---------------
MachinePolicy       Undefined
   UserPolicy       Undefined
      Process       Undefined
  CurrentUser       AllSigned
 LocalMachine      Restricted
```