Malware Groupe 8

Type de programme : 2 (Il y a bien une clé mais elle ne rentre pas dans les conditions de l'énoncé, voir la partie Fausse Clé).

Action malveillante :
Lorsqu'un debugger est présent ou que l'argument donné en entrée ne respecte pas la grammaire de l'énoncé, nous déconnectons l'user de la session avec un timer de 30 secondes. Cela nous semblait être le plus dérangeant pour un utilisateur essayant de contourner l'utilisation de notre programme.

Antidebug utilisés :
On utilise la fonction IsDebuggerPresent, du code assembleur personnalisé pour check le flag PEB, et également Checkdebuggerpresent. Ces fonctions ont été réparties tout au long de notre programme et plusieurs fois dans chaque fonction essentielle. Nous avons également inséré des debugger trap (INT 3) qui sont confondu en breakpoint par un débugger et un anti-step over qui permet de détecter si un breakpoint à été mis après une fonction et de le supprimer le cas échéant.

Fausse clé:
Nous avons décidé de faire une clé qu'il soit impossible d'atteindre : nous avons en effet inséré une clé avec un caractère non valide (ici z en l'occurrence) tout en conservant une fonction de comparaison de chaîne de caractères. Le but est de laisser penser que le programme est de type 1 et qu'il contient une clef valide utilisée, alors qu'il est de type 2.
Nous avons hésité à insérer un hash supplémentaire mais avons considéré qu'il serait impossible pour les attaquants de prouver qu'il n'y pas de clé.
Cette chaîne de caractères a été chiffrée avec une cascade de "xor" : la chaîne en entrée va être découpée en 3 sous chaines de tailles inégales et ces chaînes vont être chiffrées avec un xor et un tableau (pour chaque sous chaîne). Ensuite nous découpons ces 3 chaînes en 2 sous chaînes de tailles inégales chacunes et nous les chiffrons avec un xor et un tableau.
Au lieu d'utiliser directement la fonction xor, nous avons décidé d'utiliser une fonction substitutive à base de moins et de and . Pour obfusquer ces différentes fonctions, nous avons rajouté des lignes d'assembleur. Ces lignes ne modifient cependant pas les fonctionnalités du code.

Obfuscation syntaxique :
Dans l'ensemble de l'exécutable, nous avons dispersé du code mort (appelé, pour veiller à ce qu'il soit compilé, mais inutile) comprenant des fonctions récursives et des boucles.

Le string de résultat à été xor pour ne pas être facilement identifiable dans les strings.

Nous avons aussi cherché à cacher les fonctions "squelettes" essentielles du programme : les fonctions qui vérifient que argv[1] respecte bien la grammaire (3 fonctions), et la fonction censée comparer argv[1] avec notre clef (1 fonction). Nous les avons dispersées dans plusieurs autres sous-fonctions.
Nous avions aussi prévu des fonctions remplaçant certaines opérations de base (exemple: une fonction "add(x1,x2)" qui remplacerait l'opération de multiplication x1+x2), rendant le code plus compliqué à comprendre. Nous avons oublié de les utiliser après implémentation.