Analyse de malware

Bastien Pesme, Youssef Ifri

9 mars 2023

Introduction

Ce document est un court rapport de la réalisation de notre malware.

Test anti-debug

Nous avons commencé l'exécution du programme par un test anti-debug. Nous n'en avons pas mis beaucoup puisqu'ils sont relativement faciles à contourner. Nous aurions aimé cependant en rajouter et en dissimuler plus.

Nous vérifions juste les retours des fonctions IsDebuggerPresent(), CheckRemoteDebuggerPresent() et l'utilisation du PEB.

Chiffrement des strings

Premièrement, nous avons chiffré la plupart des chaînes de caractères utilisées. Celles-ci sont déchiffrées à la volée et le résultat est stocké dans un flux temporaire (en temps normal, aucun texte n'est affiché pendant l'exécution.).

Chiffrement des fonctions

Nous avons également embarqué des fonctions que nous avons chiffrées. Celles-ci sont également déchiffrées à la volée et brouillées après utilisation.

Pour générer le code à embarquer, nous avons rédigé ces fonctions dans un fichier source que nous avons compilé en fichier objet (avec cl.exe), puis nous les avons extraites avec dumpbin.exe. Il nous a suffi ensuite de copier leur code binaire et de l'importer dans le code source final.

Les fonctions embarquées peuvent utiliser d'autres fonctions, mais qui pourraient avoir été compilées à un autre endroit que lors de la génération du code embarqué. Nous passons donc l'adresse des fonctions utilisées en tant qu'argument des fonctions embarquées.

Les fonctions de génération de clé et de comparaison de clé ont été embarquées et chiffrées. La fonction permettant de cacher l'usage de XOR, elle, a été embarquée, mais non chiffrée. Ce, pour pouvoir modifier le code binaire et y cacher le XOR.

Appel dynamique des fonctions

Presque tous les appels de fonctions sont cachés lors de l'exécution du programme. On utilise un pointeur dont l'adresse de la fonction pointée est calculée à l'exécution.

Génération de la clé

Pour générer la clé, nous avons donc chiffré la fonction (cf. fonctions chiffrées). L'appel à cette fonction chiffrée est également caché (cf. appel dynamique).

La génération de la clé est faite par génération de nombres pseudo-aléatoires, à partir d'une graine fixe (0xdeadbeef).

Dissimulation du XOR

Puisque le chiffrement est très utilisé pour les chaînes de caractères et des fonctions, nous avons dissimulé l'usage du XOR.

Pour ce faire, nous avons caché l'opérande XOR dans un opérande plus grande, nous faisons ensuite un saut à l'adresse de l'opérande XOR, ce qui la dissimule un peu dans le désassemblage.

Attente d'une minute avant d'afficher quoi que ce soit

Enfin, comme dit plus haut, en temps normal, rien n'est affiché sur la console pendant l'exécution. À la fin du programme, celui-ci attend jusqu'à ce qu'il ait été exécuté depuis un peu moins d'une minute, puis affiche toutes les chaînes de caractères qui ont été mises en attente depuis.

Ajout de faux-flags

Nous avons rajouté quelques chaînes de caractères inutiles, par exemple "00deadbeef00". Nous avons également ajouté une fausse clé, qui est bien comparée avec l'entrée et qui provoque l'affichage d'un message particulier.

Ce qui était prévu initialement

- Utiliser un hash plutôt que comparer avec la clé directement.
- Vérifier l'intégrité du code
- Ajouter des contre-mesures plus efficaces contre le debug, etc.
- Ajouter plus de test anti-debug/les cacher mieux
- À l'origine, nous avions pensé cacher la clé dans une image ressource de l'exécutable, mais cela n'a pas été faisable.
- Complexifier le programme en usant de multithreading pour par exemple générer la clé ou plusieurs parties de clés dans un autre thread.
- modifier une fonciton anti-debug pour tromper l'analyse, par exemple mettre la fonction de comparaison de clé à la place.

Exécution du malware

malware.exe <a-f0-9>

Clé

La clé du programme : a1182c3a235f33cdb11efc9ef85e3e1cf73387266293c87f4467147ce9e3a500

il n'y en a qu'une.