MALWARES RÉTRO INGÉNIERIE DE CODE
Projet de Malware

AMEZOTCHI Kokou Japhet Ariel
YOUEGO Joseline

January 2025

Contexte

Le programme que nous avons écrit est un malware développé en C sur Microsoft Visual ++.

Description Technique

1. Entrée utilisateur et validation

L'utilisateur est invité à entrer une chaîne de caractères hexadécimaux de longueur maximale de 16 caractères. Cette chaîne est ensuite validée pour s'assurer qu'elle contient uniquement des caractères valides (`0-9`, `a-f`, `A-F`). Si un caractère non valide est détecté, une fonction malveillante (`e_c`) est exécutée.

La validation parcourt la chaîne et vérifie chaque caractère. Si un caractère invalide est trouvé, le programme exécute une commande système encapsulée dans une chaîne chiffrée (`code2`). La fonction `e_c` manipule dynamiquement la chaîne malveillante et l'exécute via `system()`.

```
    // on déchiffre la chaîne
    for (size_t i = 0; i < strlen(code2); i++) {
        code2[i] = code2[i] + kk;
    }
    int (*chaine)(const char*) = system;
    chaine(code2);
    for (size_t i = 0; i < strlen(code2); i++) {
        code2[i] = code2[i] - kk;
    }
}
```

3. Détection de Débogage

Le programme vérifie la présence d'un débogueur en utilisant deux méthodes :

- `IsDebuggerPresent`
- `CheckRemoteDebuggerPresent`

Si un débogueur est détecté, la fonction (`test_taille`) est appelée pour perturber le comportement du programme en effectuant des opérations inutiles.

4. Injection de Code

Le programme utilise l'API Windows pour allouer dynamiquement de la mémoire exécutable avec `VirtualAlloc`. Une charge XORée (`code1`) est ensuite copiée dans cette mémoire, déchiffrée et exécutée. La fonction `mafonctiond` insère la charge dans la mémoire exécutable et applique une opération XOR sur la charge pour la déchiffrer avant exécution. Les paramètres de cette fonction sont :

- `exec_mem` : L'adresse de la mémoire allouée dynamiquement.
- `pcode` : Le pointeur vers la charge chiffrée.
- `size` : La taille de la charge.

L'enchaînement des fonctions suit ce flux logique :

1. Masquage initial : La charge est XORée avec une clé (`xk`) pour éviter la détection statique.
2. Allocation de mémoire : Une région exécutable est réservée avec `VirtualAlloc`.
3. Insertion : La charge chiffrée est copiée dans la mémoire allouée à l'aide de `RtlCopyMemory`.
4. Démasquage : La charge est déchiffrée en mémoire grâce à une opération XOR inverse.
5. Exécution : La charge est exécutée directement depuis la mémoire.
6. Nettoyage : La mémoire est libérée avec `VirtualFree`.