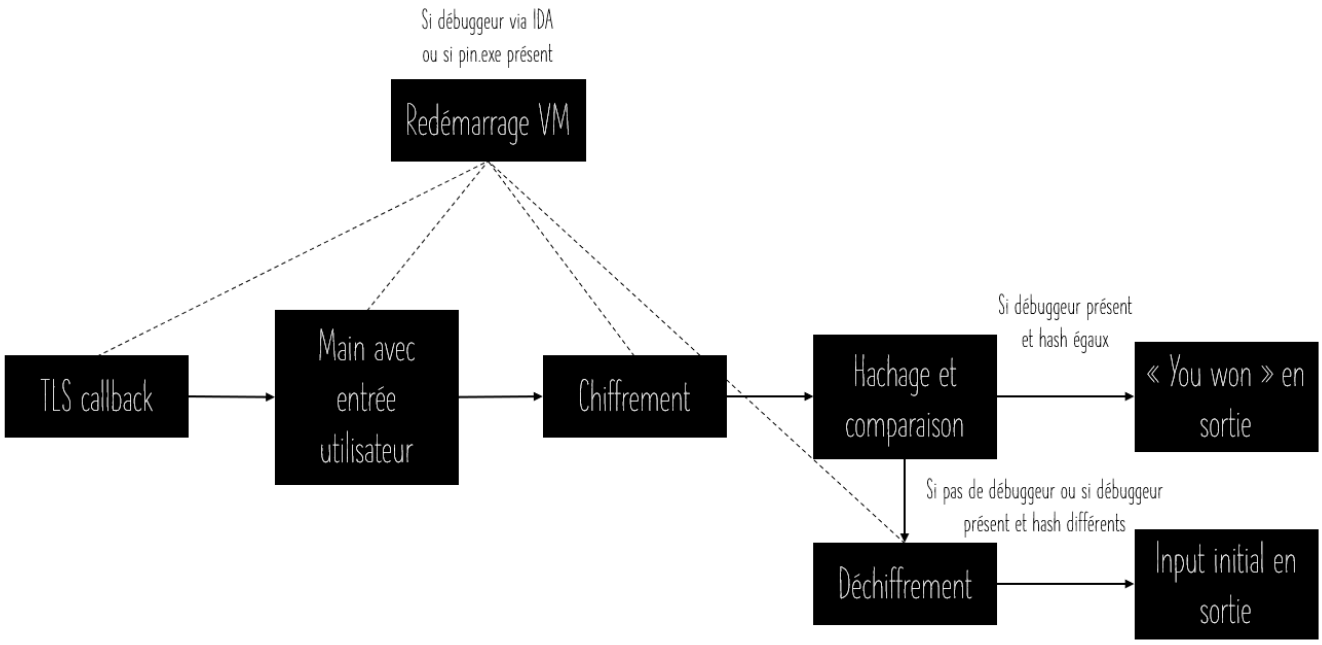
Résumé projet Malwares Attack Victor Dumange Margaux Voiselle

Structure générale du malware :



> Notre volonté avec ce malware était de piéger le binôme ennemi en leur faisant croire qu'il faisait face à un malware de type "méchant" alors qu'en réalité, il avait un comportement "gentil".

Les techniques que nous avons utilisées sont les suivantes :

- Création de deux fonctions `reboot` et `findNeedToReboot` en charge de lister les processus actuels sur la machine virtuelle et de la faire redémarrer en terminant le processus `services.exe` si jamais le processus `pin.exe` est présent dans la liste ou si jamais les deux processus `idaq64.exe` et `Malware_Dumange_Voiselle.exe` sont présents en même temps.
- Création d'une fonction `XOR` permettant de déchiffrer une chaîne de caractère chiffrée à partir d'une clé en incrémentant la clé à chaque nouveau caractère.
- Stockage des chaînes de caractère des processus dont la présence est à vérifier de manière chiffrée (avec des xor) avec déchiffrement des chaînes directement dans les fonctions `reboot` et `findNeedToReboot` quand celles-ci doivent être utilisées.
- Création d'une fonction dans le TLS callback avec génération des clés de chiffrement et déchiffrement (par simple xor) et vérification de la présence du débuggeur avec le PEB pour lancer ou non la fonction `reboot` et lancer dans tous les cas la fonction `findNeedToReboot` afin d'empêcher autant que possible une analyse dynamique.
- Obfuscation des fonctions importantes des librairies appelées (par exemple `TerminateProcess`, `IsDebuggerPresent`, `CheckRemoteDebuggerPresent`, `CryptEncrypt`) grâce à des décalages

calculés avec des fonctions existantes de la même librairie. Dans quelques cas, le décalage n'est pas exact de telle manière à arriver dans les instructions `nop` avant la fonction voulue ou alors après le début de cette fonction en ajoutant les instructions assembleur à la main dans ce cas avant l'appel de la fonction obfusquée.
- Stockage des variables globales importantes (clés de chiffrement AES, clé de chiffrement par simple xor) dans une zone de mémoire spécialement allouée dont l'accès est restreint en lecture avec des `VirtualProtect` quand elles ne sont pas utilisées.
- Ajout d'instructions (assembleur ou C++) inutiles pour obfusquer le code important avec par exemple allongement des opérations de calcul des clés pour les calculs des xor.
- Vérifications régulières dans le code du `main` de la présence d'un débuggeur avec les fonctions `CheckRemoteDebuggerPresent` et `IsDebuggerPresent` obfusquées.
- Chiffrement, hachage et déchiffrement des entrées utilisateurs avec la librairie `Wincrypt.h` et les fonctions `CryptAcquireContext`, `CryptGenKey`, `CryptEncrypt`, `CryptHashData`...
- Stockage de la chaîne de caractère qu'on fait croire "gagnante" en clair mais chiffrée de telle manière à compliquer la vérification.
- Auto modification du code du `main` au cours de l'exécution du programme avec l'appel de la fonction `dechiffre` pour compliquer l'analyse statique et cacher la vérification de la présence du débuggeur au niveau du point critique du code entraînant l'affichage de "You won, congratulations!" et faisant donc croire qu'il existe effectivement une clé dans le code.
- Ajout de code assembleur incrémentant la valeur de `eax` juste avant la vérification de la présence du débuggeur de l'auto modification pour faire croire qu'on peut effectivement passer dans la branche de l'affichage du "You won, congratulations!".

Nous avons rencontré quelques difficultés lors du développement de notre malware :

- Avec la même entrée, génération de hash SHA256 différents entre deux exécutions du programme si on faisait appel à la fonction de hash dans les fonctions `reboot` et `findNeedToReboot` ce qui nous a empêché de stocker les hash des noms des processus recherchés dans nos anti-debugs (donc moins d'obfuscation car noms des processus seulement obfusqués avec un xor donc réversibles).
- Décalage de `esp` à prévoir lors de l'obfuscation d'une fonction d'une DLL avec un décalage car sinon, le programme crashait (d'où les nombreuses instructions `__asm{}` après les appels de fonctions obfusquées).