

## Détection Manuelle du Débogueur (PEB)

**Localisation:** 0x004044DF    **Fonction:** sub\\_4044B0

**Type:** Anti-debug / Evasion    **Sévérité:** High

### Code Assembleur

```
1 ; Acces manuel au TEB puis au PEB pour verifier le flag BeingDebugged
2 mov     ebx, 28h
3 mov     ecx, 2
4 mov     eax, fs:[ebx+ecx*4]    ; EAX = Adresse du PEB (FS:[30h])
5 mov     edx, eax
6 cmp     byte ptr [edx+2], 0    ; Verifie l'offset +2 (BeingDebugged)
7 jnz     loc_4045F0            ; Saute vers la sortie d'echec si
                                detecte
```

### Code Décompilé (Reconstitution)

```
1 // Verification combinee API et acces direct memoire
2 CheckRemoteDebuggerPresent(GetCurrentProcess(), &pbDebuggerPresent);
3
4 // Lecture directe du segment FS
5 PEB* peb = (PEB*)__readfsdword(0x30);
6 if (peb->BeingDebugged == 1) {
7     return 0; // Echec silencieux
8 }
```

### Analyse

Le binaire utilise une technique d'anti-débogage classique mais obfusquée par des calculs d'index sur le segment FS. Il récupère l'adresse du **Process Environment Block (PEB)** à l'adresse FS:[0x30] et vérifie le champ **BeingDebugged** (offset 0x02). Si ce champ est à 1 (présence d'un débogueur), le programme branche vers loc\_4045F0, qui nettoie la pile et retourne 0, empêchant l'exécution de la logique de décodage.

## Opaque Predicate (Saut Impossible)

**Localisation:** 0x00404540    **Fonction:** sub\\_4044B0

**Type:** Obfuscation    **Sévérité:** Medium

### Code Assembleur

```
1 ; Calcul pr alable: 0x2A (42) * 0x11 (17) = 0x2CA (714)
2 mov     eax, [esp+6Ch+var_30] ; Charge 714
3 test    eax, eax
4 js      loc_4046D8            ; "Jump if Sign" (ne saute jamais car
                                positif)
```

## Analyse

Une structure de contrôle trompeuse (Opaque Predicate) est utilisée pour masquer le chemin vers la routine de succès. Le programme effectue une multiplication dont le résultat (714) est toujours positif. L'instruction `js` (Jump if Sign) teste si le résultat est négatif. Dans un flux normal, ce saut n'est jamais pris, cachant ainsi le bloc de code situé en `loc_4046D8` aux outils d'analyse statique. Ce bloc caché contient l'appel à `puts` permettant d'afficher la chaîne secrète.

## Chaîne Chiffrée et Sortie

**Localisation:** 0x004046D8    **Fonction:** `loc\_4046D8`

**Type:** Data Hiding    **Sévérité:** Medium

## Code Assembleur

```
1 ; Bloc atteint uniquement si l'Opaque Predicate est patch
2 mov     [esp+6Ch+hProcess], offset off_47F307 ; Chaîne Base64
3 call    ds:puts                               ; Affiche le résultat
```

## Analyse

La chaîne de caractères située à l'offset `0x47F307` contient une donnée encodée en Base64 : `"V1XTXhXWGhhU0ZaV1lsaFNWVlZxUmt0WGJGcDBU"`. L'analyse dynamique a révélé que cette chaîne n'est pas déchiffrée par la fonction `puts` standard, mais dépend probablement des constantes calculées précédemment (`0x2A`, `0x11`) pour un déchiffrement XOR manuel ou une validation via l'algorithme récursif identifié plus loin. En modifiant le `.exe` nous n'avons pas réussi à obtenir le déchiffrement de la chaîne.

## IOC

- **String** : `V1XTXhXWGhhU0ZaV1lsaFNWVlZxUmt0WGJGcDBU` (Base64)
- **Offset** : `0x47F307`