15. "Asesinar" procesos

Monday, November 16, 2020 6:26 PM

Para hacer un script que nos permite "asesinar" procesos necesitamos manejar diversas funciones de la WinAPI:

- FindWindowA
  Recupera un identificador en la ventana de nivel superior cuyo nombre de clase y nombre de ventana coinciden con las cadenas especificadas. Esta función no busca ventanas secundarias. Esta función no realiza una búsqueda que distingue mayúsculas de minúsculas.
  Parámetros de entrada:
  - lpClassName - Especifica el nombre de la clase de ventana. Si su valor es NULL, busca la ventana cuyo nombre empata con el parámetro lpWindowName.
  - lpWindowName - Es el nombre de la ventana. Si su valor es NULL, todas los nombres de ventana empatan.

  Valor de retorno - Si la función es exitosa, el valor de retorno es el handle de la ventana especificada. Si la función falla, su valor de retorno es NULL.
- GetWindowThreadProcessId
  Recupera el identificador del subproceso (thread) que creó la ventana especificada y, opcionalmente, el identificador del proceso que creó la ventana.
  Parámetros de entrada:
  - hWnd - Un handle a la ventana
  - lpdwProcessId - Un puntero a una variable que recibe el identificador del proceso. Si este valor es NULL, GetWindowThreadProcessId copia el identificador del proceso a la variable, de otra forma no.

  Valor de retorno: El identificador del subproceso (thread) que creó la ventana.
- OpenProcess
  Abre un objeto existente de proceso local.
  Parámetros de entrada:
  - dwDesiredAccess - Es el acceso al objeto de proceso.
  - bInheritHandle - Si su valor es True, procesos creados por este prcoeso van a heredar el handle, de otro forma, el proceso no heredará el handle.
  - dwProcessId - El identificador del proceso local que será abierto.

  Valor de retorno: Si la función se realiza correctamente, el valor devuelto es un identificador abierto para el proceso especificado. Si se produce un error en la función, el valor devuelto es NULL.
- TerminateProcess
  Termina el proceso especificado y todos sus subprocesos (threads).
  Parámetros de entrada:
  - hProcess - Un handle del proceso que será terminado.
  - uExitCode - El código de salida que será usado por el proceso y los subprocesos terminados como resultado de la llamada. Para conocer más sobre estos valores revise GetExitCodeProcess.

ProcKiller.py

ProcKiller

Sección	Código
1	`import ctypes`

```
2  k_handle = ctypes.WinDLL("Kernel32.dll")
   u_handle = ctypes.WinDLL("User32.dll")
3  PROCESS_ALL_ACCESS = (0x000F0000 | 0x00100000 | 0xFFF)
4  lpWindowName = ctypes.c_char_p(input("Enter Window Name to Kill: ").encode('utf-
   8'))
   hWnd = u_handle.FindWindowA(None, lpWindowName)
5  if hWnd == 0:
       msgError = "Error Code: {0} - Could Not Grab Handle"
       print(msgError.format(k_handle.GetLastError()))
       exit(1)
6  else:
       print("Got the Handle!")
       lpdwProcessId = ctypes.c_ulong()
       response = u_handle.GetWindowThreadProcessId(hWnd, ctypes.byref(lpdwProcessId))
7      if response == 0:
           msgError = "Error Code: {0} - Could Not Grab PID"
           print(msgError.format(k_handle.GetLastError()))
           exit(1)
8      else:
           print("Got the PID!")
           dwDesiredAccess = PROCESS_ALL_ACCESS
           bInheritHandle = False
           dwProcessId = lpdwProcessId
           hProcess = k_handle.OpenProcess(dwDesiredAccess, bInheritHandle,
   dwProcessId)
9          if hProcess <= 0:
               msgError = "Error Code: {0} - Could Not Grab Priv Handle"
               print(msgError.format(k_handle.GetLastError()))
10         else:
               print("Got our Handle...")
               uExitCode = 0x1
               response = k_handle.TerminateProcess(hProcess, uExitCode)
11             if response == 0:
                   msgError = "Error Code: {0} - Could Not Terminate Process"
                   print(msgError.format(k_handle.GetLastError()))
12             else:
                   print("Process Went Bye Bye!")
```

Explicación

1. Importamos ctypes
2. Creamos un handle para User32.dll y para Kernel32.dll
3. Es una variable que nos da acceso de todos los modos a las funciones que vamos a realizar (estándar, sincronizado, error).
4. Capturamos el nombre de la ventana que queremos cerrar e intentamos obtener un handle de esa ventana.
5. Si el resultado de la función anterior es 0, resulta que ocurrió un error, por lo que informamos del error al usuario y cerramos el script.
6. Si no se dieron errores implica que si obtuvimos el handle y buscaremos obtener el ID del proceso asociado a esa ventana.
7. Si la respuesta de GetWindowThreadProcessId es 0, significa que ocurrió un error, por lo que informamos al usuario y terminamos el script.
8. Si no hubo error, tenemos el ID del proceso (PID), por lo que definimos el tipo de acceso, limitamos la herencia de los subprocesos y copiamos a una nueva variable el PID. Con estos datos intentamos obtener el handle del proceso abierto en la variable hProcess.
9. Si la variable hProcess tiene un valor menor o igual a cero, significa que hubo un error. Actuaremos en consecuencia para terminar el script.
10. Si no hubo error, significa que tenemos el handle del proceso y definiremos el código de salida como 1 (podemos usar otro valor). Usaremos la función TerminateProcess para terminar el proceso sobre el cual tenemos privilegios que está en hProcess.

11. Si la respuesta obtenida de TerminateProcess es 0, significa que hubo un error, por lo que actuaremos en consecuencia.
12. Si no hubo error, significa que logramos terminar el proceso y la ventana se cerró. Informamos al usuario con un mensaje.