16. Iniciar procesos

Tuesday, November 17, 2020 5:06 PM

Para hacer un script que nos permite iniciar (spawn) procesos, necesitamos definir 2 estructuras y usar una función de la WinAPI:

- Estructura STARTUPINFOA - Especifica el estado de ventana, el escritorio, los identificadores estándar y el aspecto de la ventana principal para un proceso en el momento de la creación.
- Estructura PROCESS_INFORMATION - Contiene información sobre un proceso recién creado y su subproceso principal.
- Función CreateProcessW

  Crea un nuevo proceso y su subproceso principal. El nuevo proceso se ejecuta en el contexto de seguridad del proceso de llamada. Si el proceso de llamada está suplantando a otro usuario, el nuevo proceso usa el token para el proceso de llamada, no el token de suplantación.

```
BOOL CreateProcessW(
  LPCWSTR               lpApplicationName,
  LPWSTR                lpCommandLine,
  LPSECURITY_ATTRIBUTES lpProcessAttributes,
  LPSECURITY_ATTRIBUTES lpThreadAttributes,
  BOOL                  bInheritHandles,
  DWORD                 dwCreationFlags,
  LPVOID                lpEnvironment,
  LPCWSTR               lpCurrentDirectory,
  LPSTARTUPINFOW        lpStartupInfo,
  LPPROCESS_INFORMATION lpProcessInformation
);
```

ProcKiller.py

ProcKiller

Sección	Código
1	`import ctypes` `from ctypes.wintypes import DWORD,LPWSTR,WORD,LPBYTE,HANDLE`
2	`k_handle = ctypes.WinDLL("Kernel32.dll")`

3	`class STARTUPINFO(ctypes.Structure):`
	`  _fields_ = [`
	`  ("cb", DWORD),`
	`  ("lpReserved", LPWSTR),`
	`  ("lpDesktop", LPWSTR),`
	`  ("lpTitle", LPWSTR),`
	`  ("dwX", DWORD),`
	`  ("dxY", DWORD),`
	`  ("dwXSize", DWORD),`
	`  ("dwYSize", DWORD),`
	`  ("dwXCountChars", DWORD),`
	`  ("dwYCountChars", DWORD),`
	`  ("dwFillAttribute", DWORD),`
	`  ("dwFlags", DWORD),`
	`  ("wShowWindow", WORD),`
	`  ("cbReserved2", WORD),`

```
    ("lpReserved2", LPBYTE),
    ("hStdInput", HANDLE),
    ("hSTDOutput", HANDLE),
    ("hStdError", HANDLE),
    ]
```

```
class PROCESS_INFORMATION(ctypes.Structure):
    _fields_ = [
    ("hProcess", HANDLE),
    ("hThread", HANDLE),
    ("dwProcessId", DWORD),
    ("dwThreadId", DWORD),
    ]
```

```
lpApplicationName = "C:\\Windows\\System32\\cmd.exe"
lpCommandLine = None
lpProcessAttributes = None
lpThreadAttributes = None
lpEnviroment = None
lpCurrentDirectory = None
dwCreationFlags = 0x00000010 #Nueva consola
bInheritHandle = False
lpProcessInformation = PROCESS_INFORMATION()
lpStartupInfo = STARTUPINFO()
lpStartupInfo.wShowWindow = 0x1
lpStartupInfo.dwFlags = 0x1
```

```
response = k_handle.CreateProcessW(
    lpApplicationName,
    lpCommandLine,
    lpProcessAttributes,
    lpThreadAttributes,
    bInheritHandle,
    dwCreationFlags,
    lpEnviroment,
    lpCurrentDirectory,
    ctypes.byref(lpStartupInfo),
    ctypes.byref(lpProcessInformation))
```

```
if response > 0:
    print("Proc is running")
else:
    print("Failed. Error Code: {0}".format(k_handle.GetLastError()))
```

Explicación

1. Importamos ctypes y los tipos de datos de ctypes
2. Creamos un handle para el Kernel32.dll
3. Creamos la clase STARTUPINFO que emula la estructura procedente de C++.
4. Creamos la clase PROCESS_INFORMATION que emula la estructura procedente de C++.
5. Definimos todas las variables que necesitamos.
6. Invocamos el método CreateProcessW.
7. Si la respuesta no es cero, significa que la función si realizó tu tarea con éxito.