

1. One byte flip trick doesn't work anymore. We should try to make a shellcode encoder-decoder in c++
2. Second option, we can use EXOCET again but try to execute inline assembly using CGO. EXOCET already encrypts the buffer. All we need to do is import "C" and add "C" code on top of the stub. The problem is that at decryption, the malware is now represented in the stack

```

D:\GithubProjects\EXOCET-AV-Evasion\CGOTest>go build main.go
# command-line-arguments
.\main.go:14:16: expected '\', found '\', EOF
.\main.go:17:2: expected '\', found '\', EOF

D:\GithubProjects\EXOCET-AV-Evasion\CGOTest>go build main.go
# command-line-arguments
.\main.go:14:3735: missing '\', in argument list

D:\GithubProjects\EXOCET-AV-Evasion\CGOTest>go build main.go
# command-line-arguments
.\main.go:3:1: warning: return type defaults to 'int' [-Wimplicit-int]
3 | call(char *code) {
  | ~~~~~
go build command-line-arguments: open C:\Users\ctan1\AppData\Local\Temp\go-build2772847795\b001\_pkg_.a: Operation did not complete successfully because the file contains a virus or potentially unwanted software.

D:\GithubProjects\EXOCET-AV-Evasion\CGOTest>go run main.exe
no required module provides package main.exe: go.mod file not found in current directory or any parent directory; see 'go help modules'

D:\GithubProjects\EXOCET-AV-Evasion\CGOTest>go run main.go
# command-line-arguments
.\main.go:3:1: warning: return type defaults to 'int' [-Wimplicit-int]
3 | call(char *code) {
  | ~~~~~
go build command-line-arguments: open C:\Users\ctan1\AppData\Local\Temp\go-build1098779922\b001\_pkg_.a: Operation did not complete successfully because the file contains a virus or potentially unwanted software.

D:\GithubProjects\EXOCET-AV-Evasion\CGOTest>
  
```

We successfully found Go and CGO code that can successfully execute shellcode and malware. Now we need to add it to our crypter function.

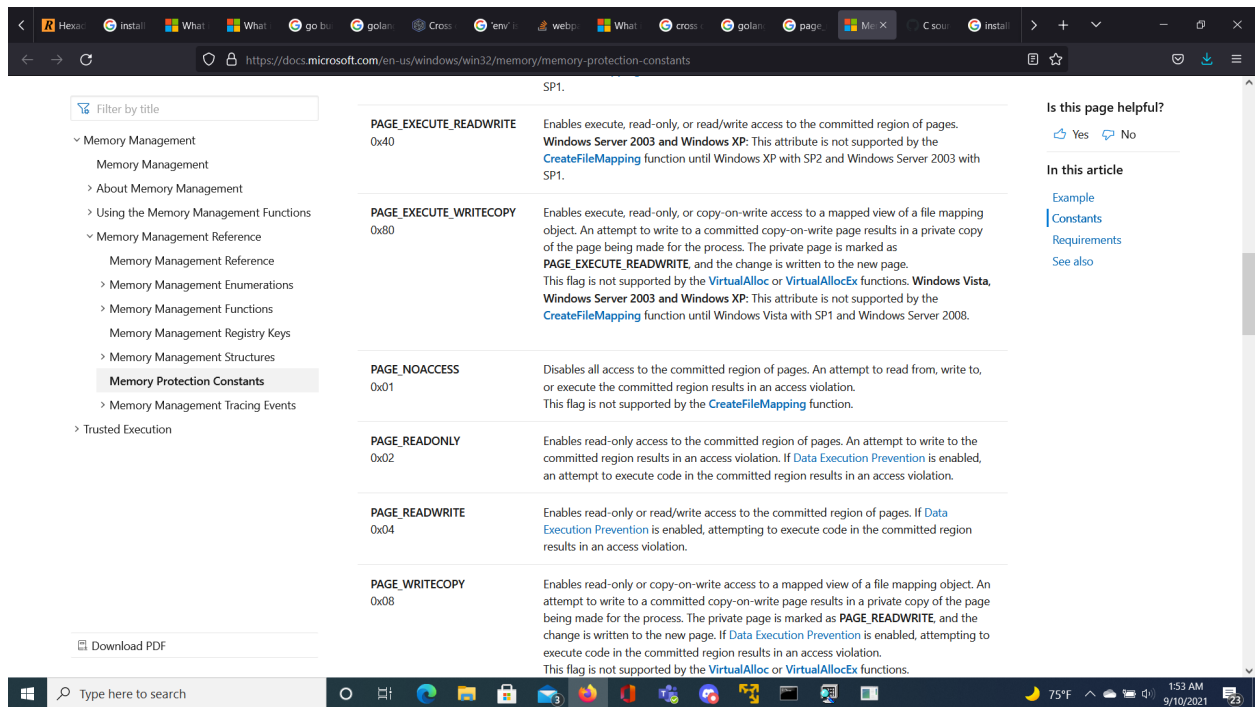
What it has to do is take a cobalt strike beacon C payload output, ask the user to just submit the bytes in a text file, have EXOCET encrypt it and then DECRYPT it on the fly to execute in memory.

We find some sort of memory access violation issue. Not sure if its the operating system, but the previous payload without encryption executed and got picked up by Windows Defender.

3. SharpShooter could be used
4. TrustedSec Unicorn payloads can be used
5. If the target was Linux, Unix, or Mac OSX, we can resort to DarkLordObama

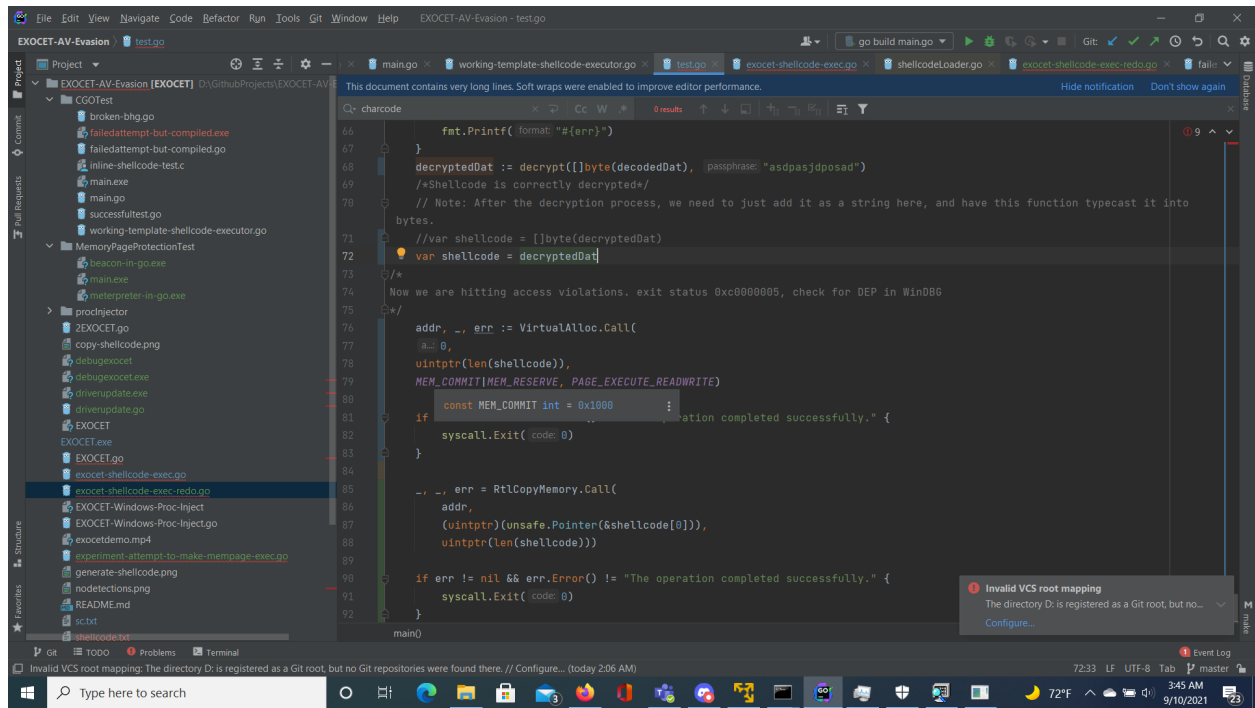
We found that for some reason the shellcode lies in a memory range where it is not RWX.

We ran the decrypted C payload from both Beacon and Metasploit and we came across the same memory access violation in WinDBGx64, !vprot shows page noaccess



As it turns out we need to explicitly allocate memory and make it RWX using go itself with **VirtualAlloc**

We then combine unsafe pointers and syscalls to allow direct execution of shellcode.



Furthermore, using msfvenom's num-transform ability, msfvenom -p payload -f num works and runs the payload just fine

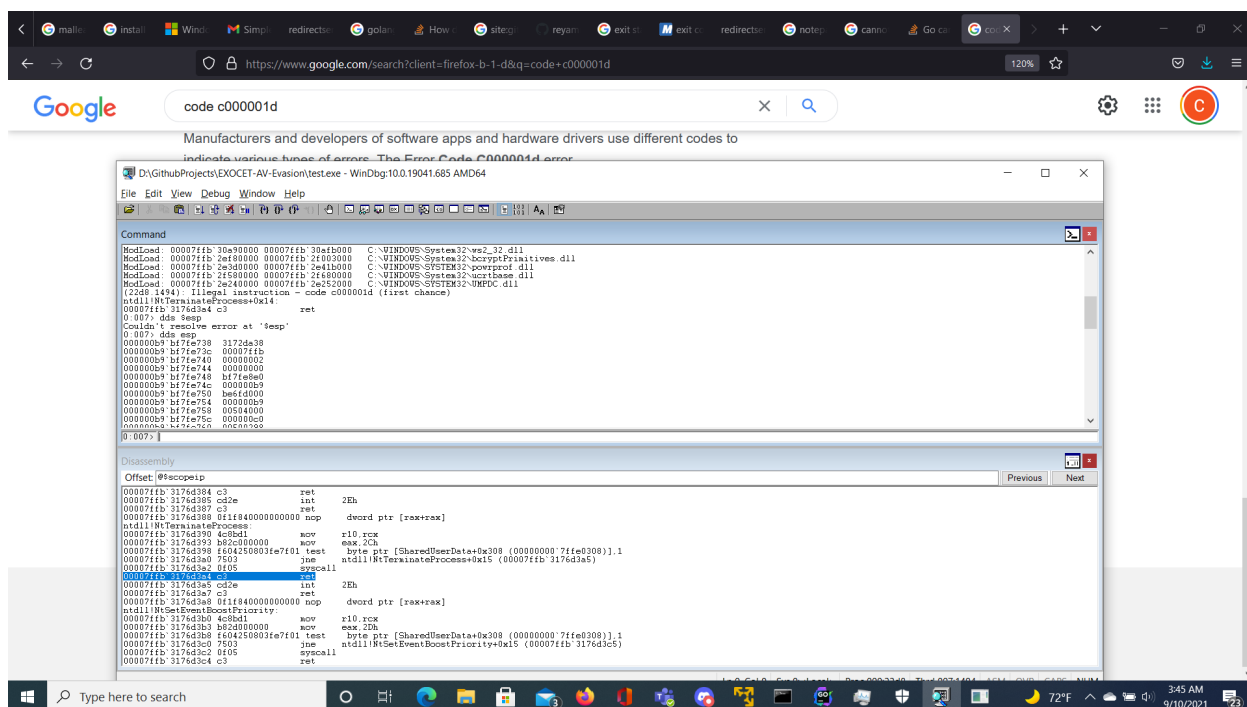
The tens of thousands of lines of Meterpreter shellcode was in num-transform format and then added as a []byte slice by copy-and-paste

The screenshot shows a Windows desktop environment. In the foreground, a command prompt window is open, displaying a list of system processes with columns for process name, PID, session ID, architecture, and private bytes. The processes listed include Firefox.exe, WmiPrvSE.exe, IpoverUsbSvc.exe, Microsoft.Photos.exe, svchost.exe, RuntimeBroker.exe, and many others. The command prompt is titled 'Select C:\WINDOWS\system32\cmd.exe'. In the background, a web browser window is open, showing a search result for 'How do you list all processes on the command line in Windows?'. The search result is from 'https://superuser.com' and includes a link to 'How to show full command line of all processes in Windows'. The desktop taskbar at the bottom shows various application icons and the system clock indicating 2:58 AM on 9/10/2021.

Process Name	PID	Session ID	Architecture	Private Bytes
Firefox.exe	11050	Console	1	67,916 K
WmiPrvSE.exe	22280	Services	0	35,220 K
IpoverUsbSvc.exe	19668	Services	0	13,072 K
Microsoft.Photos.exe	13756	Console	1	1,484 K
svchost.exe	17008	Services	0	9,640 K
RuntimeBroker.exe	18512	Console	1	6,920 K
Firefox.exe	21356	Console	1	237,304 K
Firefox.exe	7676	Console	1	81,784 K
goland.exe	15632	Console	1	2,967,448 K
fsnotifier.exe	8564	Console	1	2,500 K
conhost.exe	1440	Console	1	5,800 K
jcef_helper.exe	22236	Console	1	37,240 K
jcef_helper.exe	16552	Console	1	21,376 K
Firefox.exe	16032	Console	1	182,716 K
Firefox.exe	16180	Console	1	83,204 K
audiodg.exe	8172	Services	0	18,600 K
MpCmdRun.exe	17704	Services	0	11,072 K
svchost.exe	14080	Services	0	7,504 K
smartscreen.exe	17544	Console	1	24,280 K
windbg.exe	17324	Console	1	63,160 K
meterpreter-in-go.exe	19964	Console	1	10,072 K
conhost.exe	18316	Console	1	15,388 K
Taskmgr.exe	20104	Console	1	61,168 K
Teams.exe	22316	Console	1	77,340 K
cmd.exe	6092	Console	1	4,572 K
conhost.exe	18992	Console	1	16,740 K
tasklist.exe	20800	Console	1	9,044 K
WmiPrvSE.exe	14092	Services	0	9,716 K

However, attempting to read from and decrypt from a file of num-transformed shellcode from Metasploit does not work, and results in illegal instructions.

This applies to both msfvenom and Cobalt Strike Beacons.



Conclusions

CGO can be used, but it...

1. Limits the development of the final payload to Windows users of Golang who have installed the MinGW toolchain. That means you can't run EXOCET from Kali Linux or make Windows malware from a Linux distro. All malware development must be done on Windows because CGO cannot cross-compile
2. Cannot be cross-compiled. You cannot turn CGO code from x64 to i386, it will throw compilation errors even if you ran the command `CGO_ENABLED=1`
3. Will support C shellcode generated by Metasploit or Cobalt Strike in `\x00` format

Syscalls and unsafe pointers can be used instead but...

1. Will only support num-transformed shellcode, which is a `0x00, 0x0a` type of formatting
2. Can be cross-compiled across different architectures
3. Allows calling `VirtualAlloc` and giving it `RWX` permissions on memory pages, allowing you to execute shellcode
4. Requires interaction with Windows APIs, including but not limited to `kernel32.dll` and `ntdll.dll`
5. Since Go only supports num-transformed shellcode, we need to make a decoder to convert classic C or Python shellcode into num-transformed format automatically, and also remove whitespaces and line breaks.

The simple way, is to fix all of the formatting issues and incorporate Go-compatible shellcode to allow cross-platform compilation of payloads.