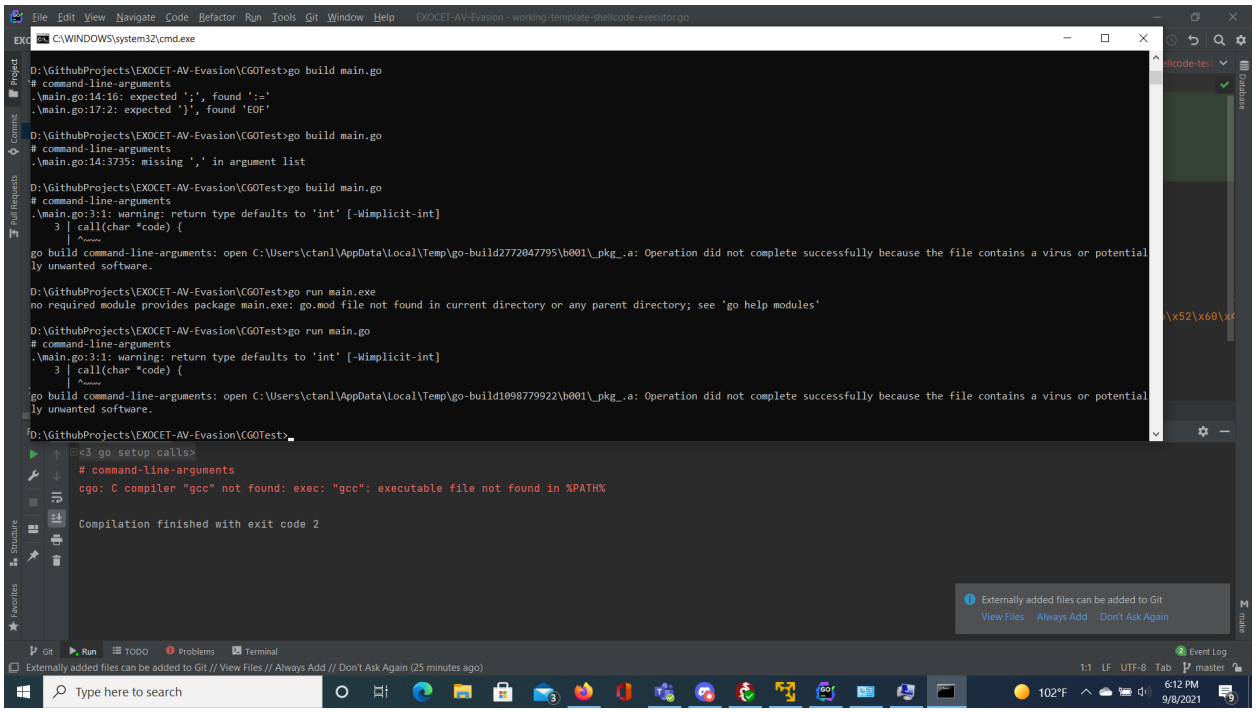
1. One byte flip trick doesn't work anymore. We should try to make a shellcode encoder-decoder in c++
2. Second option, we can use EXOCET again but try to execute inline assembly using CGO. EXOCET already encrypts the buffer. All we need to do is import "C" and add "C" code on top of the stub. The problem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



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

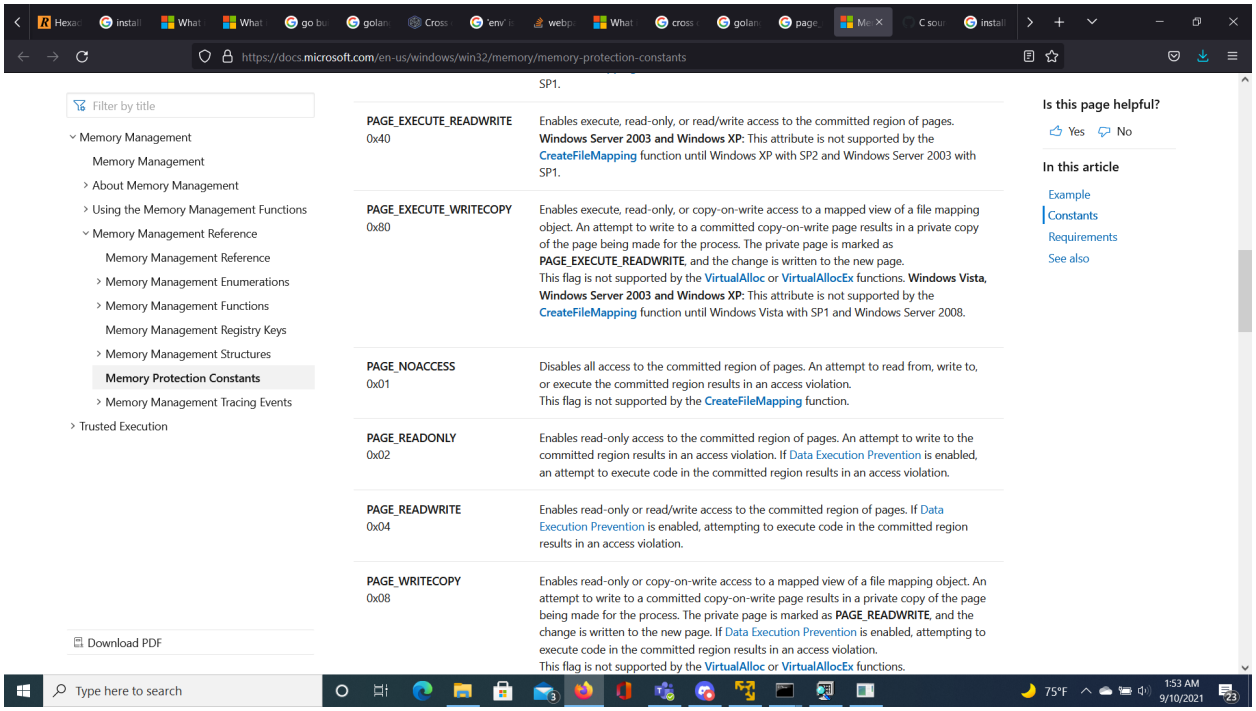
What it has to do is take a cobalt strike beacon C payload output, ask the user to just submit the bytes in a text file, have EXOCET encrypt it and then DECRYPT it on the fly to execute in memory.

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

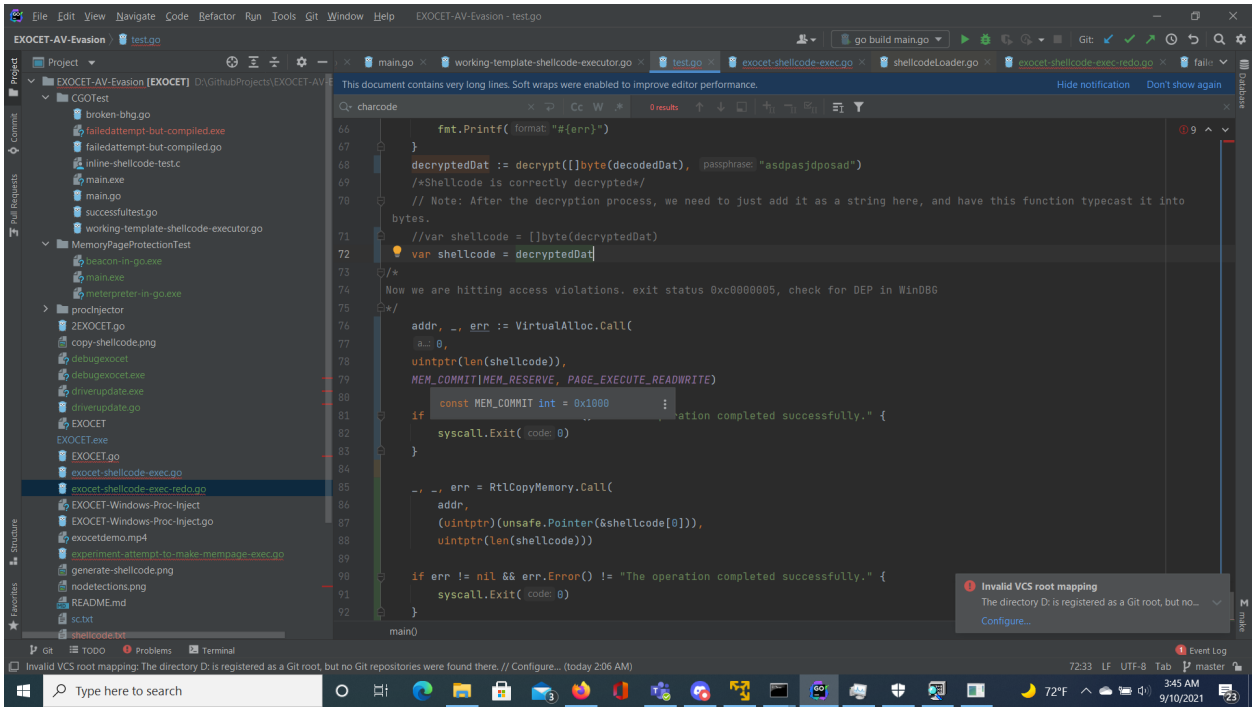
We found that for some reason the shellcode lies in a memory range where it is not RWX.

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



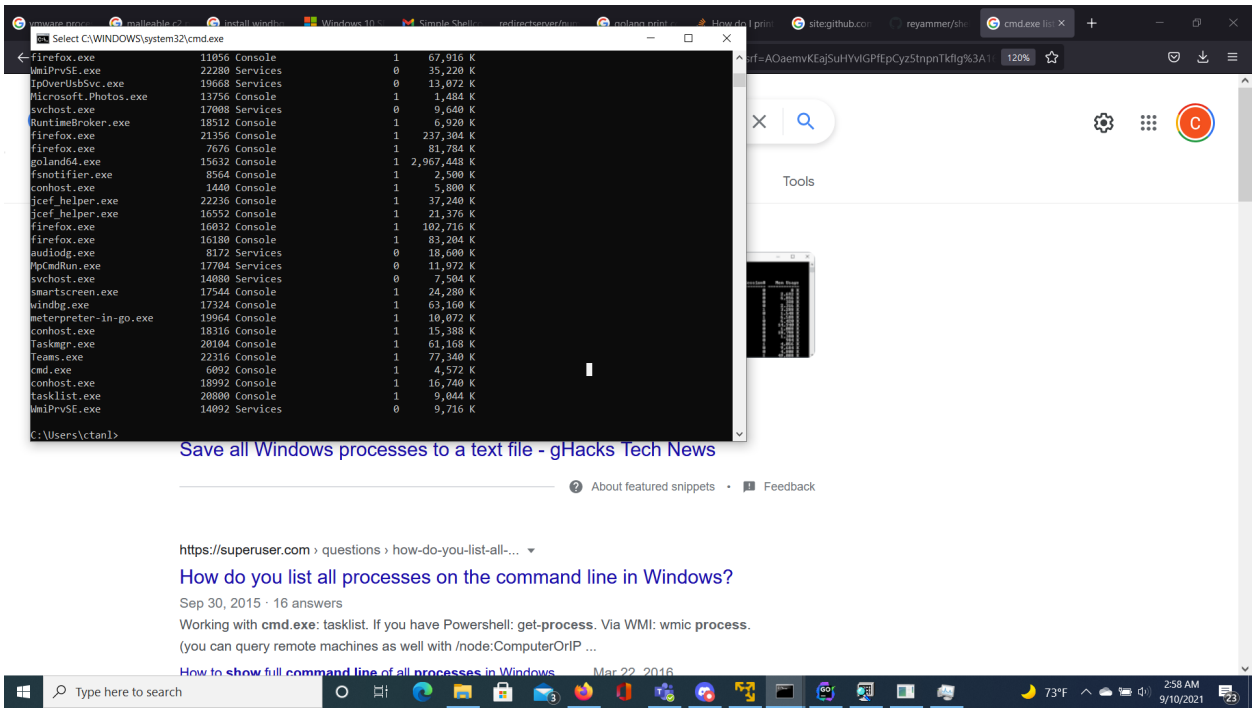
As it turns out we need to explicitly allocate memory and make it RWX using go itself with VirtualAlloc

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



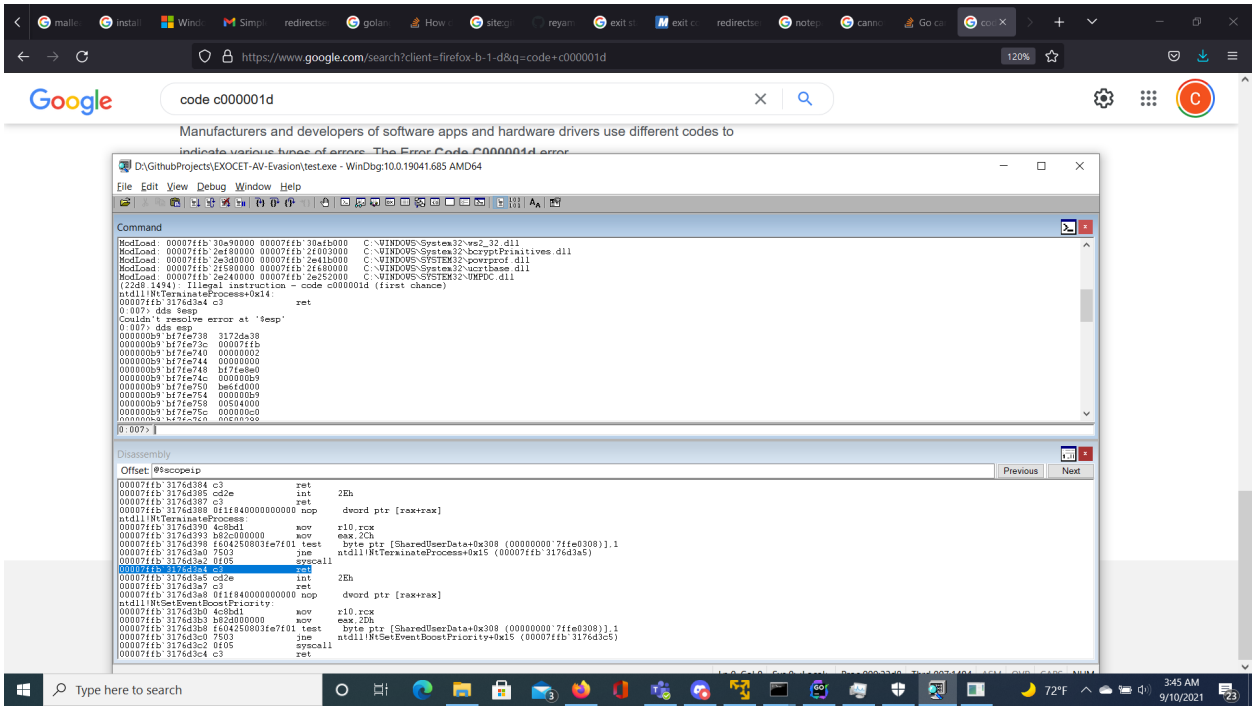
Furthermore, using msfvenom's num-transform ability, msfvenom -p payload -f num works and runs the payload just fine

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



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.