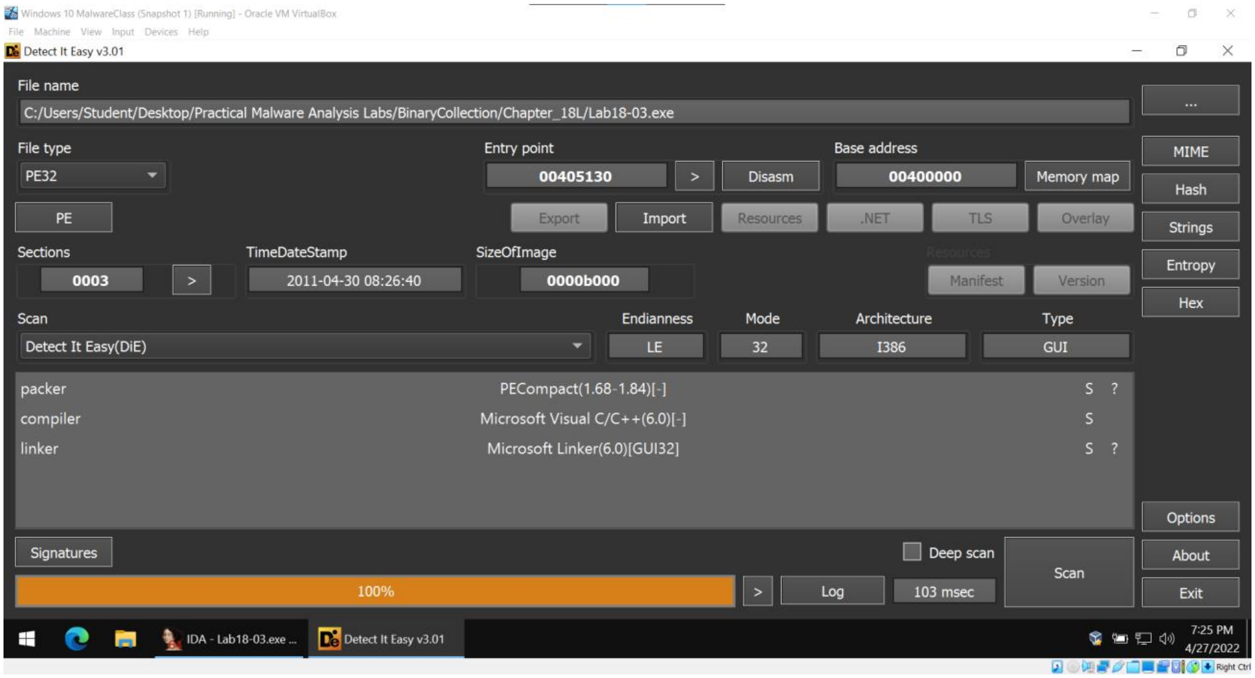
The first thing we should do is take a look at the malware in Detect it Easy:


Windows 10 MalwareClass (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Detect It Easy v3.01
...
MIME
Hash
Strings
Entropy
Hex
Options
About
Exit
>

File name: C:/Users/Student/Desktop/Practical Malware Analysis Labs/BinaryCollection/Chapter_18L/Lab18-03.exe

File type: PE32

Entry point: 00405130 > Disasm

Base address: 00400000 Memory map

PE Export Import Resources .NET TLS Overlay

Sections: 0003 >

TimeDateStamp: 2011-04-30 08:26:40

SizeOfImage: 0000b000

Resources: Manifest Version

Scan: Detect It Easy(DiE)

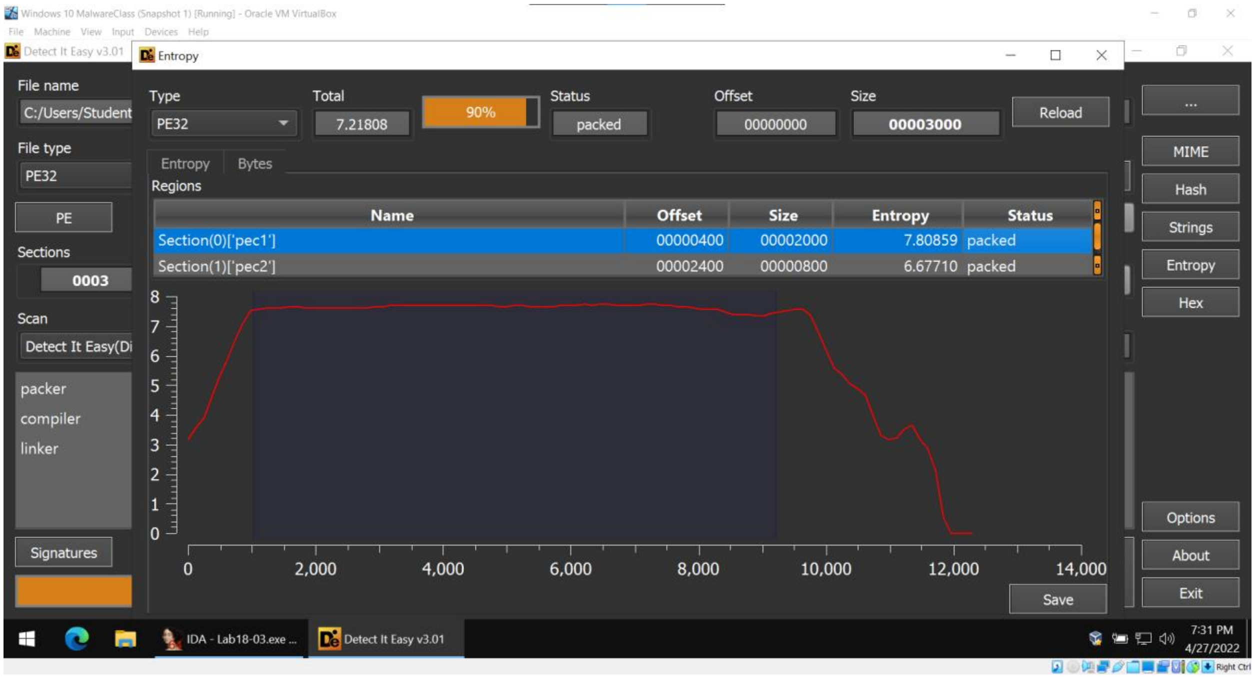
Endianness	Mode	Architecture	Type
LE	32	I386	GUI

packer	PECompact(1.68-1.84)[-]	S	?
compiler	Microsoft Visual C/C++(6.0)[-]	S	
linker	Microsoft Linker(6.0)[GUI32]	S	?

Signatures ☐ Deep scan Scan

100% > Log 103 msec

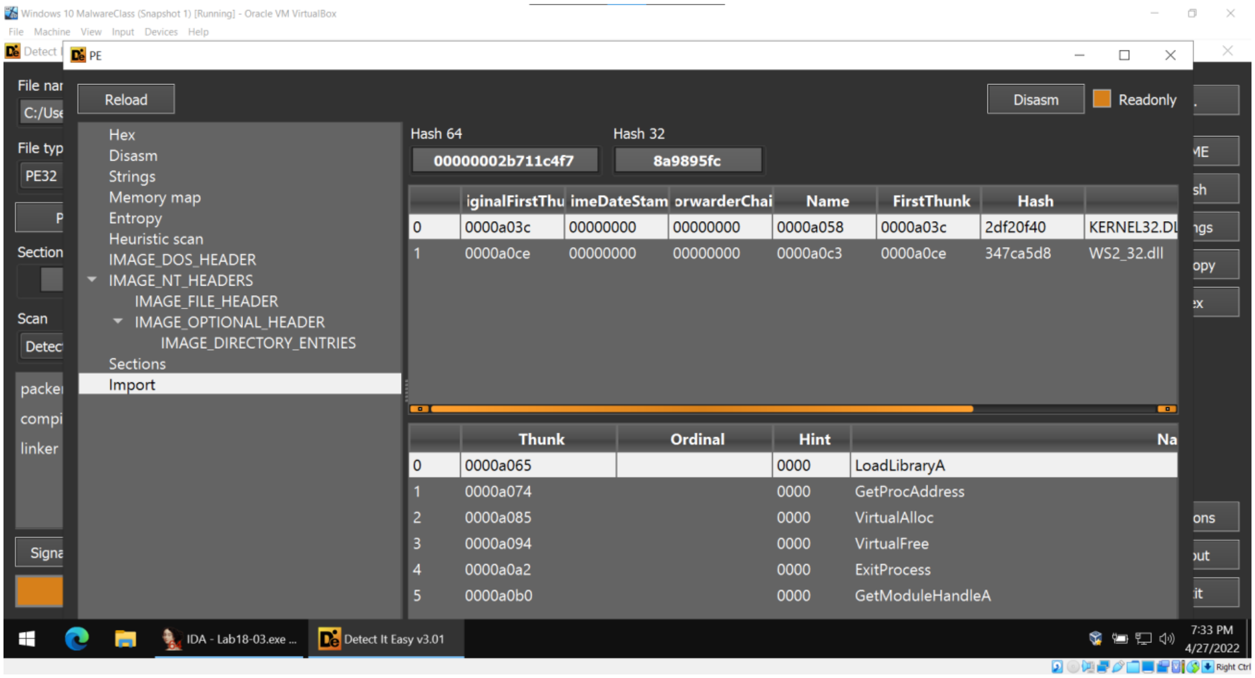
Right off the bat, we can tell that this is most probably packed, since DiE has detected that the PECompact packer is likely being used. On top of this automatic detection, there's a few other key indicators that suggest packing:



Type: PE32 Total: 7.21808 90% Status: packed Offset: 00000000 Size: 00003000

Name	Offset	Size	Entropy	Status
Section(0)['pec1']	00000400	00002000	7.80859	packed
Section(1)['pec2']	00002400	00000800	6.67710	packed

A rather large section of the executable has a very high entropy – almost 8! This suggests that this section is packed. Since most packers pack the *code* of an executable, we should suspect that a large section with an extremely high entropy contains the packed code. There are other indicators as well:



Hash 64: 00000002b711c4f7

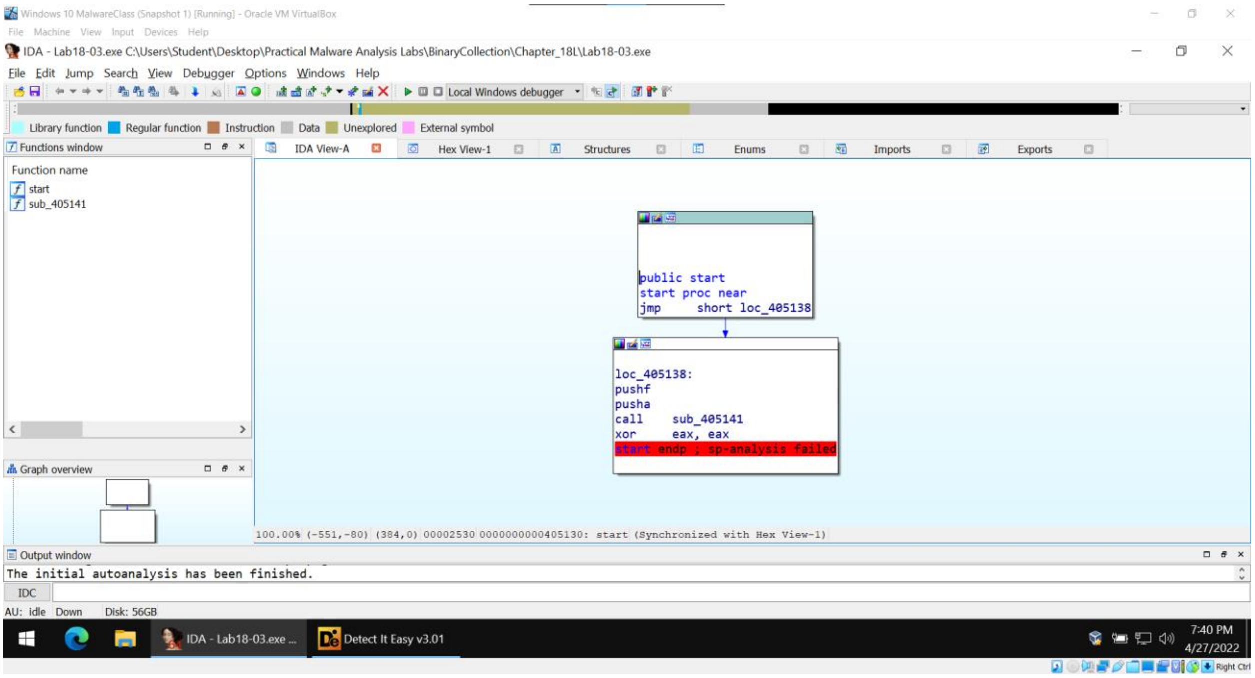
Hash 32: 8a9895fc

	iginalFirstThu	imeDateStam	orwarderChai	Name	FirstThunk	Hash	
0	0000a03c	00000000	00000000	0000a058	0000a03c	2df20f40	KERNEL32.DL
1	0000a0ce	00000000	00000000	0000a0c3	0000a0ce	347ca5d8	WS2_32.dll

	Thunk	Ordinal	Hint	Na
0	0000a065		0000	LoadLibraryA
1	0000a074		0000	GetProcAddress
2	0000a085		0000	VirtualAlloc
3	0000a094		0000	VirtualFree
4	0000a0a2		0000	ExitProcess
5	0000a0b0		0000	GetModuleHandleA

Note that there are very few imports for this executable, which means that the executable will need to dynamically load and link various DLLs during runtime if it wants to do anything particularly interesting. In fact, of the few functions that are actually important on load, we see LoadLibraryA and GetProcAddress – these allow the program to do that kind of dynamic loading and linking. This also strongly suggests packing.

Attempting to disassemble (or decompile) this executable yields less than desirable results:



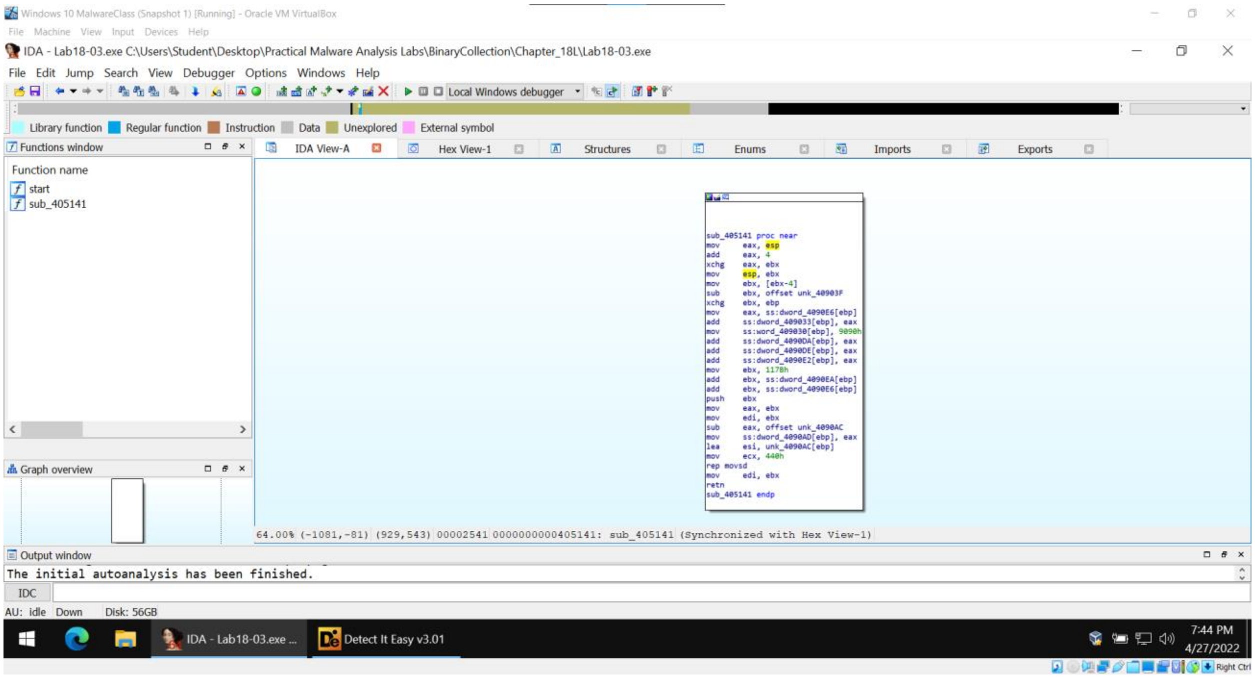
```
public start
start proc near
jmp     short loc_405138
```

```
loc_405138:
pushf
pusha
call    sub_405141
xor     eax, eax
start endp ; sp-analysis failed
```

```
100.00% (-551,-80) (384,0) 00002530 0000000000405130: start (Synchronized with Hex View-1)
```

```
The initial autoanalysis has been finished.
```

IDA very quickly runs into bytes that it can't interpret as assembly code and its analysis fails. Even worse, there doesn't appear to be any obvious tail jump instruction, even if we check the call to the function 405141:



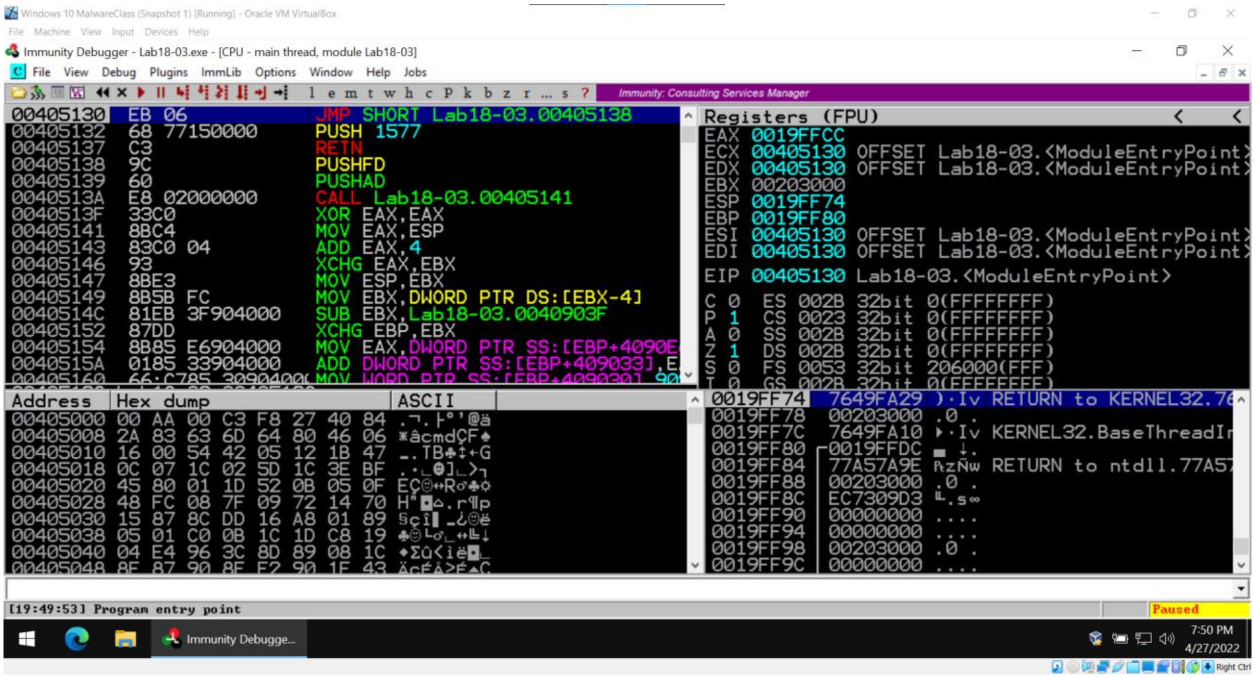
```
sub_405141 proc near
mov     eax, esp
add     eax, 4
xchg    eax, ebx
mov     esp, ebx
mov     ebx, [ebx-4]
sub     ebx, offset unk_40903F
xchg    ebx, ebp
mov     eax, ss:dword_4090E6[ebp]
add     ss:dword_409033[ebp], eax
mov     ss:word_409030[ebp], 9090h
add     ss:dword_4090DA[ebp], eax
add     ss:dword_4090DE[ebp], eax
add     ss:dword_4090E2[ebp], eax
mov     ebx, 1178h
add     ebx, ss:dword_4090EA[ebp]
add     ebx, ss:dword_4090E6[ebp]
push    ebx
mov     eax, ebx
mov     edi, ebx
sub     eax, offset unk_4090AC
mov     ss:dword_4090AD[ebp], eax
lea     esi, unk_4090AC[ebp]
mov     ecx, 440h
rep movsd
mov     edi, ebx
retn
sub_405141 endp
```

```
64.00% (-1081,-81) (929,543) 00002541 0000000000405141: sub_405141 (Synchronized with Hex View-1)
```

```
The initial autoanalysis has been finished.
```

```
AU:  idle   Down    Disk: 56GB
```

If we assume that this malware is actually functional, then we can pretty safely assume that a tail jump *will* eventually exist when the unpacking stub finishes execution: the program's code is probably going to be dynamically modified while its still loaded to insert a jump instruction (among other things) into it. Since this isn't really something that we can discover via static analysis, we shift gears to a more dynamic method:


Windows 10 MalwareClass (Snapshot 1) [Running] - Oracle VM VirtualBox
Immunity Debugger - Lab18-03.exe - [CPU - main thread, module Lab18-03]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Immunity: Consulting Services Manager
Immunity Debugge...
7:50 PM
4/27/2022

```
00405130  EB 06            JMP SHORT Lab18-03.00405138
00405132  68 77150000      PUSH 1577
00405137  C3               RETN
00405138  9C               PUSHFD
00405139  60               PUSHAD
0040513A  E8 02000000      CALL Lab18-03.00405141
0040513F  33C0             XOR EAX,EAX
00405141  8BC4             MOV EAX,ESP
00405143  83C0 04          ADD EAX,4
00405146  93               XCHG EAX,EBX
00405147  8BE3             MOV ESP,EBX
00405149  8B5B FC          MOV EBX,DWORD PTR DS:[EBX-4]
0040514C  81EB 3F904000    SUB EBX,Lab18-03.0040903F
00405152  87DD             XCHG EBP,EBX
00405154  8B85 E6904000    MOV EAX,DWORD PTR SS:[EBP+4090E
0040515A  0185 33904000    ADD DWORD PTR SS:[EBP+409033],E
```

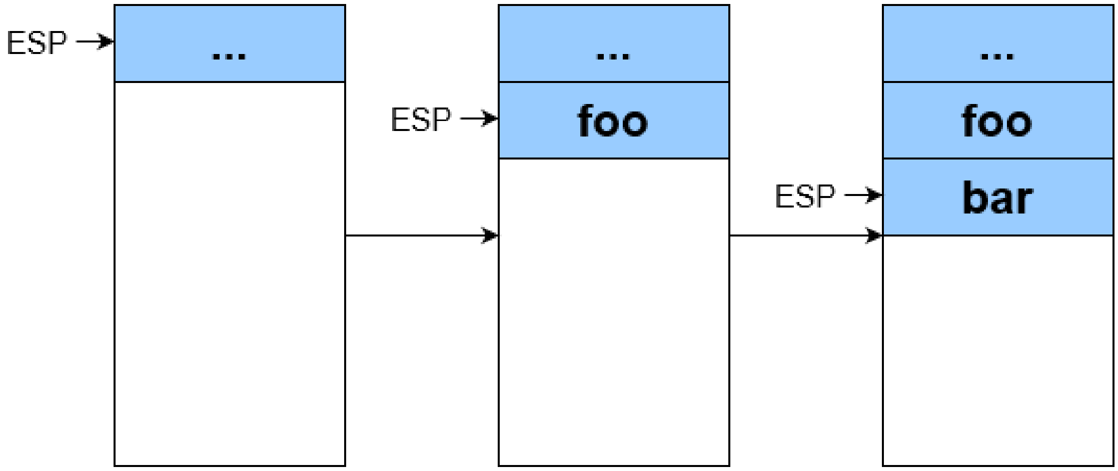
```
Registers (FPU)
EAX 0019FFCC
ECX 00405130 OFFSET Lab18-03.<ModuleEntryPoint>
EDX 00405130 OFFSET Lab18-03.<ModuleEntryPoint>
EBX 00203000
ESP 0019FF74
EBP 0019FF80
ESI 00405130 OFFSET Lab18-03.<ModuleEntryPoint>
EDI 00405130 OFFSET Lab18-03.<ModuleEntryPoint>

EIP 00405130 Lab18-03.<ModuleEntryPoint>

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 206000(FFF)
```

```
Address   Hex dump                 ASCII
00405000  00 AA 00 C3 F8 27 40 84  .¬.├°'@ä
00405008  2A 83 63 6D 64 80 46 06  *âcmdÇF♠
00405010  16 00 54 42 05 12 1B 47  ▬.TB♣↕←G
00405018  0C 07 1C 02 5D 1C 3E BF  .•∟☻]∟>┐
00405020  45 80 01 1D 52 0B 05 0F  EÇ☺↔R♂♣☼
00405028  48 FC 08 7F 09 72 14 70  Hⁿ◘⌂.r¶p
00405030  15 87 8C DD 16 A8 01 89  §çî▌▬¿☺ë
00405038  05 01 C0 0B 1C 1D C8 19  ♣☺└♂∟↔╚↓
00405040  04 E4 96 3C 8D 89 08 1C  ♦Σû<ìë◘∟
```

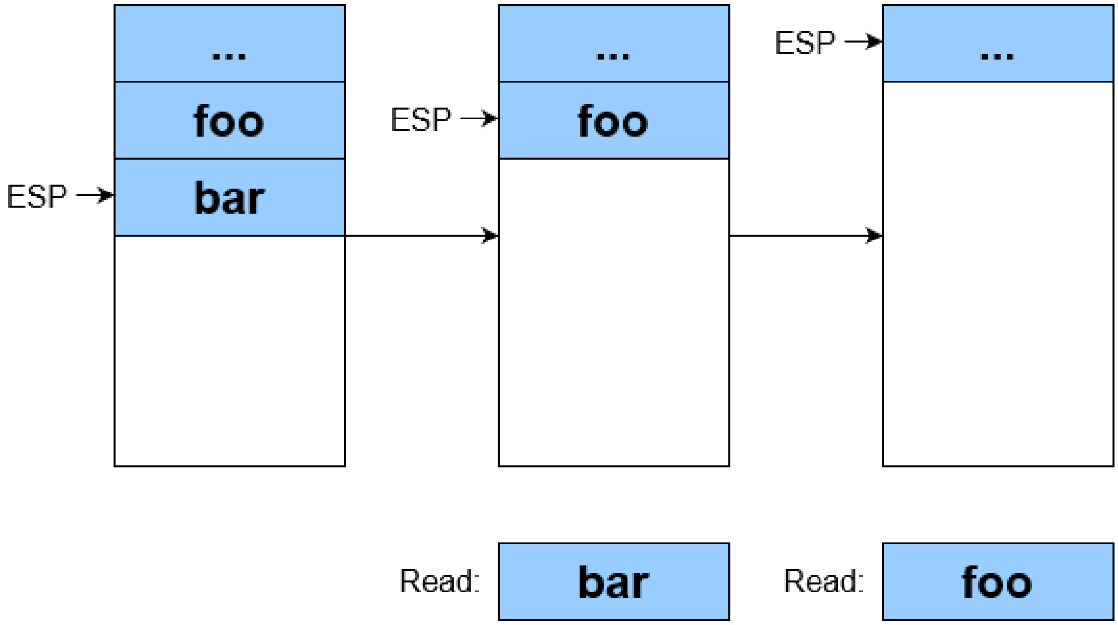
```
0019FF74  7649FA29  )·Iv  RETURN to KERNEL32.76
0019FF78  00203000  .0 .
0019FF7C  7649FA10  ►·Iv  KERNEL32.BaseThreadI
0019FF80  0019FFDC  ▄ ↓.
0019FF84  77A57A9E  ₧zÑw  RETURN to ntdll.77A57
0019FF88  00203000  .0 .
0019FF8C  EC7309D3  ╙.s∞
0019FF90  00000000  ....
0019FF94  00000000  ....
0019FF98  00203000  .0 .
0019FF9C  00000000  ....
```

```
[19:49:53] Program entry point                    Paused
```

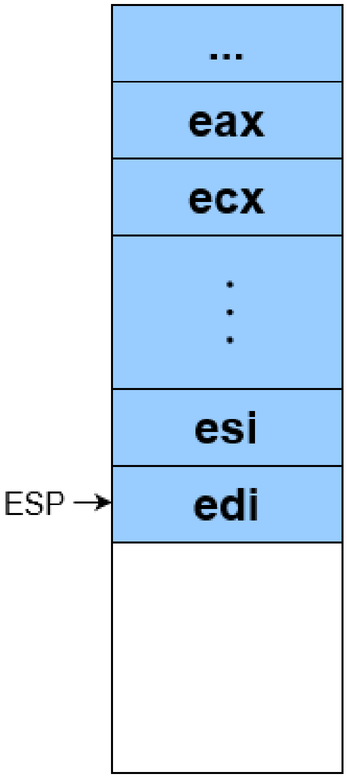
Loading the program in immunity debugger show us mostly the same things we saw when we opened it with IDA: a small jump, a pusha, a pushf, and a function call. The pusha and pushf instructions (which push all registers and flags onto the stack respectively) specifically give us a hint as to something the malware will do after it is unpacked (and give us the means to unpack it and analyze it), but to understand why, we need to review how the stack works. The stack is a data structure that exists in a process's memory and grows "downwards" (towards higher memory addresses) as more data put onto the stack. Data is removed from the stack in a "bottom up" fashion, where data that was written to the stack more recently is read and removed first. The CPU keeps track of where we are in the stack using a register (listed as ESP on the register view of immunity):



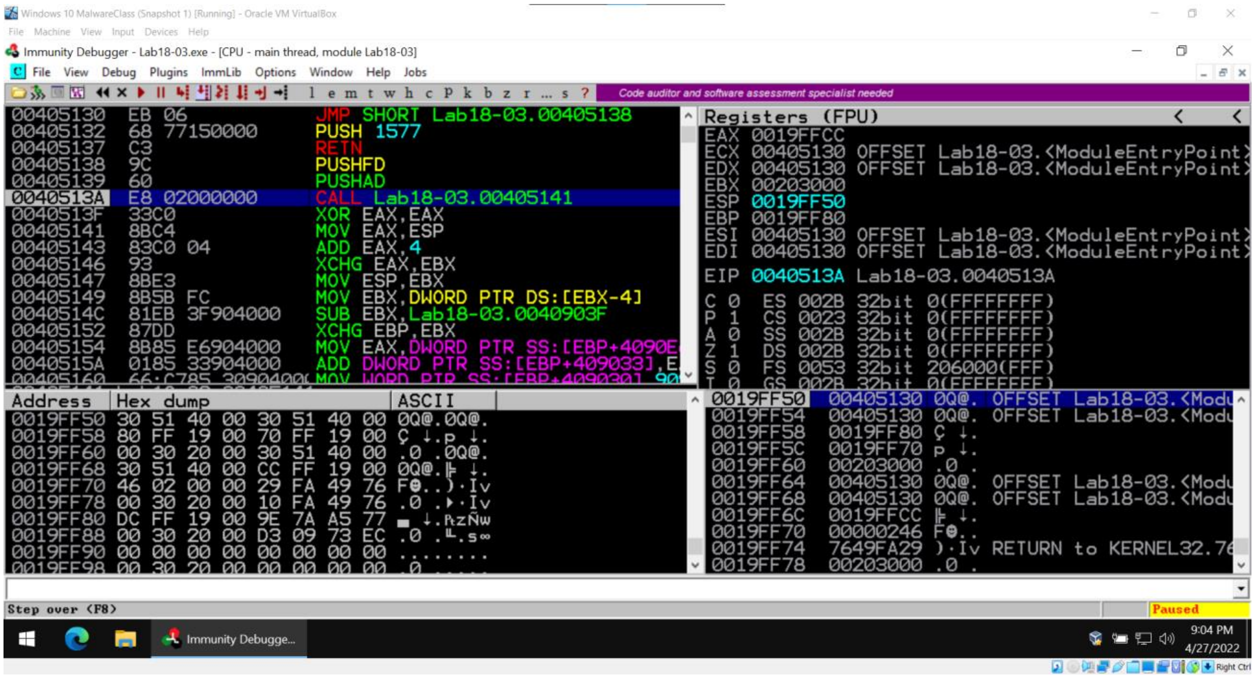
The stack pointer starts at the "top" of the stack – when data is inserted onto the stack, it is placed at the location denoted by ESP (which is then incremented to move on to the next free memory location). When data is read and removed from the stack, we look at the data



One of the first things the malware does is execute the instructions pusha and pushf (displayed as pushad and pushfd in immunity – the “d” suffix indicates that it is operating on a DWORD, which is the size of an entire 32 bit register). Pusha will push each of the CPU registers onto the stack (in a particular order) when it is executed, and pushf does the same with the flags. This suggests that the malware is trying to preserve the state of the CPU right when the program starts, probably to restore it later when the unpacking (which will certainly modify the registers while executing) is finished and it’s about to jump to the newly unpacked code. We can actually exploit this fact using a memory breakpoint. Consider what the stack looks like right after pusha executes: it will have all of the CPU registers placed onto the stack:



If the malware was truly interested in restoring the processor state right before jumping to the actual code, we can pretty safely assume that it will keep these values exactly as they are on the stack until it comes time to restore them – probably using an instruction like popa (which restores all registers from the stack). This means that the memory locations on the stack that are storing the saved registers aren’t going to be accessed (i.e. popped) until that time. A lot is going to happen during the unpacking – registers will change values, and things will probably get put onto and removed from the stack, but the malware will likely be careful not to pop any of those saved registers during that process in order to keep them safe until it runs the popa command. If we could monitor the memory at (for example) the location on the stack that is storing the edi register (which is the last register to be pushed by pusha) to see when it actually gets popped, we will be able to tell the exact moment where popa is called. We can’t just search the disassembled malware code for the popa instruction since we know (from our earlier static analysis) that it hasn’t been unpacked yet (meaning it’s obfuscated and unreadable) - we won’t see it in the program memory at all until the unpacking finishes. Immunity allows us to set memory breakpoints pretty easily – the main challenge is figuring out where in the program memory the registers we're interested in are. Thankfully, this isn't actually too hard if we're clever about stepping through the program. Consider the above diagram showing the stack immediately after pusha executes: the stack pointer (ESP) is pointing at the memory location that is storing edi. If we set a memory access breakpoint at that location, then it will likely trigger right after the popa instruction (which will access that memory address) gets executed. The best way to do this is to step through the program from the beginning until right after we execute pusha:


Windows 10 MalwareClass (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Immunity Debugger - Lab18-03.exe - [CPU - main thread, module Lab18-03]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Code auditor and software assessment specialist needed
Immunity Debugge...
9:04 PM
4/27/2022
Right Ctrl

```
00405130  EB 06            JMP SHORT Lab18-03.00405138
00405132  68 77150000      PUSH 1577
00405137  C3               RETN
00405138  9C               PUSHFD
00405139  60               PUSHAD
0040513A  E8 02000000      CALL Lab18-03.00405141
0040513F  33C0             XOR EAX,EAX
00405141  8BC4             MOV EAX,ESP
00405143  83C0 04          ADD EAX,4
00405146  93               XCHG EAX,EBX
00405147  8BE3             MOV ESP,EBX
00405149  8B5B FC          MOV EBX,DWORD PTR DS:[EBX-4]
0040514C  81EB 3F904000    SUB EBX,Lab18-03.0040903F
00405152  87DD             XCHG EBP,EBX
00405154  8B85 E6904000    MOV EAX,DWORD PTR SS:[EBP+4090E
0040515A  0185 33904000    ADD DWORD PTR SS:[EBP+409033],E
00405160  66:C785 3090400( MOV WORD PTR SS:[EBP+409030],90
```

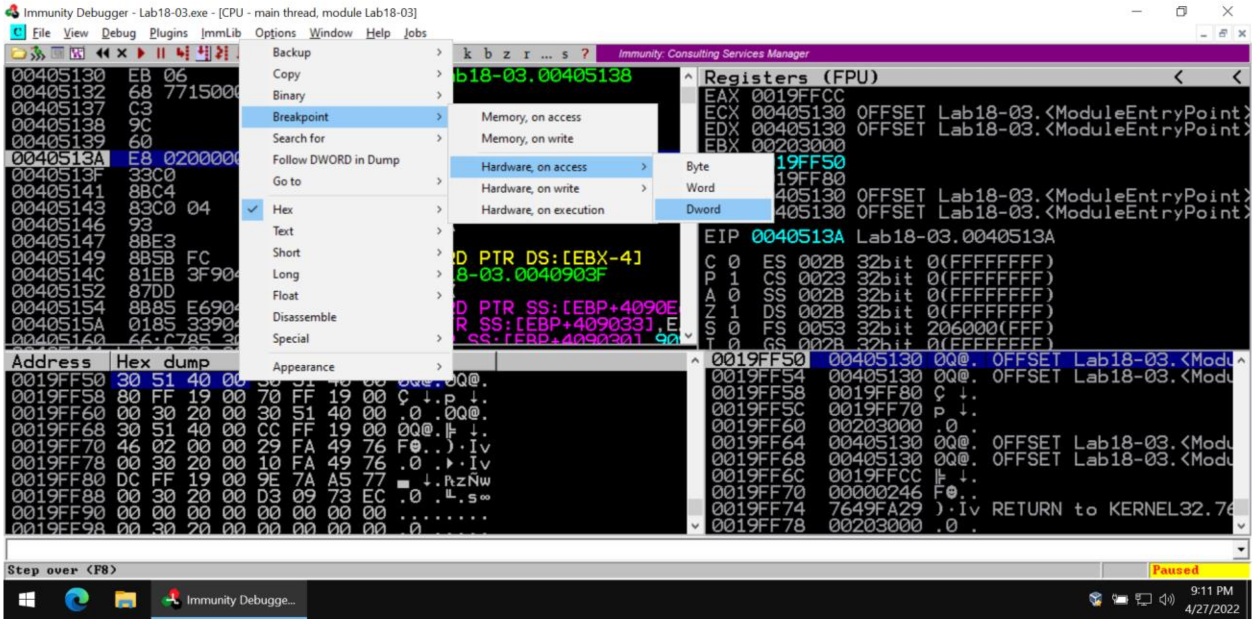
```
Registers (FPU)
EAX 0019FFCC
ECX 00405130 OFFSET Lab18-03.<ModuleEntryPoint>
EDX 00405130 OFFSET Lab18-03.<ModuleEntryPoint>
EBX 00203000
ESP 0019FF50
EBP 0019FF80
ESI 00405130 OFFSET Lab18-03.<ModuleEntryPoint>
EDI 00405130 OFFSET Lab18-03.<ModuleEntryPoint>

EIP 0040513A Lab18-03.0040513A

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 206000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
```

```
Address   Hex dump                 ASCII
0019FF50  30 51 40 00 30 51 40 00  0Q@.0Q@.
0019FF58  80 FF 19 00 70 FF 19 00  Ç ↓.p ↓.
0019FF60  00 30 20 00 30 51 40 00  .0 .0Q@.
0019FF68  30 51 40 00 CC FF 19 00  0Q@.╠ ↓.
0019FF70  46 02 00 00 29 FA 49 76  F☻..)·Iv
0019FF78  00 30 20 00 10 FA 49 76  .0 .►·Iv
0019FF80  DC FF 19 00 9E 7A A5 77  ▄ ↓.₧zÑw
0019FF88  00 30 20 00 D3 09 73 EC  .0 .╙.s∞
0019FF90  00 00 00 00 00 00 00 00  ........
0019FF98  00 30 20 00 00 00 00 00  .0 .....
```

```
0019FF50  00405130 0Q@. OFFSET Lab18-03.<Modu
0019FF54  00405130 0Q@. OFFSET Lab18-03.<Modu
0019FF58  0019FF80 Ç ↓.
0019FF5C  0019FF70 p ↓.
0019FF60  00203000 .0 .
0019FF64  00405130 0Q@. OFFSET Lab18-03.<Modu
0019FF68  00405130 0Q@. OFFSET Lab18-03.<Modu
0019FF6C  0019FFCC ╠ ↓.
0019FF70  00000246 F☻..
0019FF74  7649FA29 )·Iv RETURN to KERNEL32.76
0019FF78  00203000 .0 .
```

```
Step over (F8)                                  Paused
```

Note that the ESP register is blue, meaning it was just changed by the last instruction. This makes sense since we just pushed a whole bunch of registers onto the stack, so ESP should have decreased by quite a few bytes (if you pause right after pushf instead, you'll see that ESP is all the way back at 00199FF70 – we just pushed 32 bytes onto the stack with pusha). The memory location currently pointed to by ESP (0019FF50) is the memory location that is currently storing the value of edi on the stack. In fact, if you look at the memory viewer at the bottom left, you'll see that 0019FF50 has the values 30 51 40 00 in memory – this is storing the value 00405130, which is exactly what we see in the current edi register. This is the memory location we need to monitor. We can tell immunity to do just that by highlighting all four of those bytes, right clicking, then going to Breakpoint->Hardware, on access->Dword:



```
00405130  EB 06          b18-03.00405138
00405132  68 7715000
00405137  C3
00405138  9C
00405139  60
0040513A  E8 0200000
0040513F  33C0
00405141  8BC4
00405143  83C0 04
00405146  93
00405147  8BE3
00405149  8B5B FC        D PTR DS:[EBX-4]
0040514C  81EB 3F904     8-03.0040903F
00405152  87DD
00405154  8B85 E6904     D PTR SS:[EBP+4090E
0040515A  0185 33904     R SS:[EBP+409033],E
```

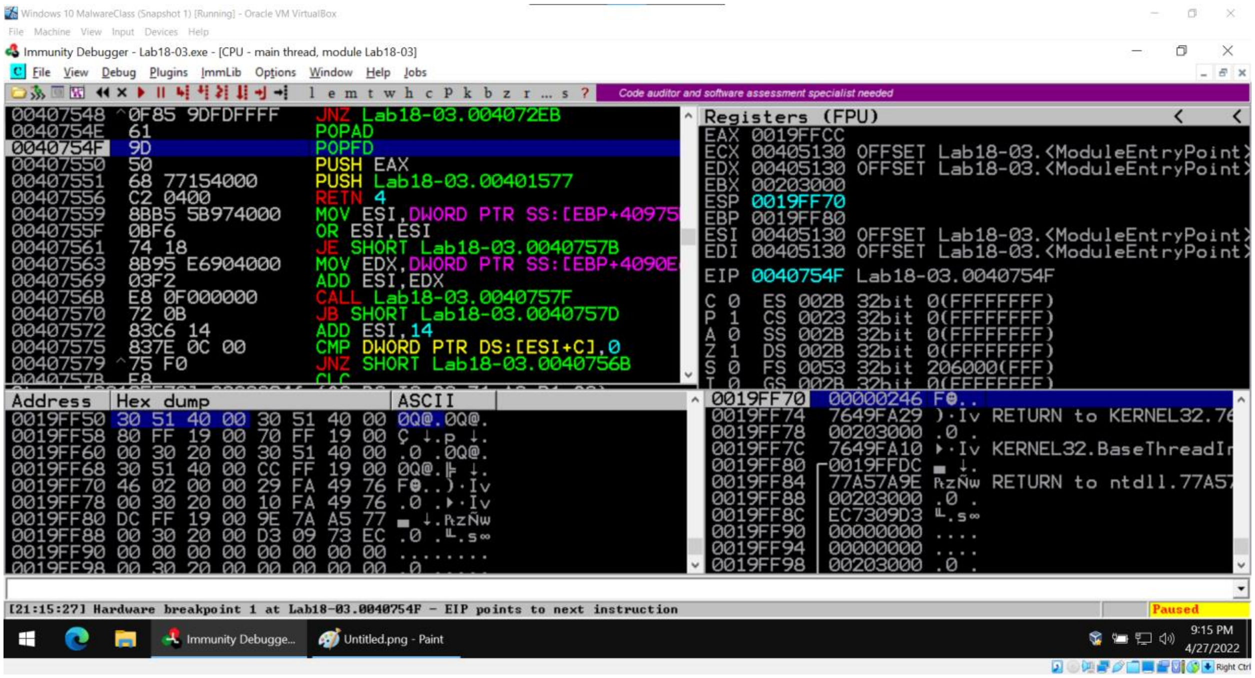
```
Registers (FPU)
EAX 0019FFCC
ECX 00405130 OFFSET Lab18-03.<ModuleEntryPoint>
EDX 00405130 OFFSET Lab18-03.<ModuleEntryPoint>
EBX 00203000
    19FF50
    19FF80
    405130 OFFSET Lab18-03.<ModuleEntryPoint>
    405130 OFFSET Lab18-03.<ModuleEntryPoint>
EIP 0040513A Lab18-03.0040513A
C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 206000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
```

```
Address   Hex dump
0019FF50  30 51 40 00                0Q@.
0019FF58  80 FF 19 00 70 FF 19 00  Ç ↓.p ↓.
0019FF60  00 30 20 00 30 51 40 00  .0 .0Q@.
0019FF68  30 51 40 00 CC FF 19 00  0Q@.╠ ↓.
0019FF70  46 02 00 00 29 FA 49 76  F☻..)·Iv
0019FF78  00 30 20 00 10 FA 49 76  .0 .►·Iv
0019FF80  DC FF 19 00 9E 7A A5 77  ▄ ↓.₧zÑw
0019FF88  00 30 20 00 D3 09 73 EC  .0 .╙.s∞
0019FF90  00 00 00 00 00 00 00 00  ........
0019FF98  00 30 20 00 00 00 00 00  .0 .....
```

```
0019FF50   00405130 0Q@.  OFFSET Lab18-03.<Modu
0019FF54   00405130 0Q@.  OFFSET Lab18-03.<Modu
0019FF58   0019FF80 Ç ↓.
0019FF5C   0019FF70 p ↓.
0019FF60   00203000 .0 .
0019FF64   00405130 0Q@.  OFFSET Lab18-03.<Modu
0019FF68   00405130 0Q@.  OFFSET Lab18-03.<Modu
0019FF6C   0019FFCC ╠ ↓.
0019FF70   00000246 F☻..
0019FF74   7649FA29 )·Iv RETURN to KERNEL32.76
0019FF78   00203000 .0 .
```

```
Step over (F8)                                          Paused
```

Now immunity will automatically pause execution right after that memory location gets accessed, which should only happen when popa gets executed. If we start running the program, we'll find that it does just that:


Windows 10 MalwareClass (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Immunity Debugger - Lab18-03.exe - [CPU - main thread, module Lab18-03]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Code auditor and software assessment specialist needed
Immunity Debugge...
Untitled.png - Paint
9:15 PM
4/27/2022
Right Ctrl

```
00407548 ^0F85 9DFDFFFF    JNZ Lab18-03.004072EB
0040754E  61               POPAD
0040754F  9D               POPFD
00407550  50               PUSH EAX
00407551  68 77154000      PUSH Lab18-03.00401577
00407556  C2 0400          RETN 4
00407559  8BB5 5B974000    MOV ESI,DWORD PTR SS:[EBP+40975
0040755F  0BF6             OR ESI,ESI
00407561  74 18            JE SHORT Lab18-03.0040757B
00407563  8B95 E6904000    MOV EDX,DWORD PTR SS:[EBP+4090E
00407569  03F2             ADD ESI,EDX
0040756B  E8 0F000000      CALL Lab18-03.0040757F
00407570  72 0B            JB SHORT Lab18-03.0040757D
00407572  83C6 14          ADD ESI,14
00407575  837E 0C 00       CMP DWORD PTR DS:[ESI+C],0
00407579 ^75 F0            JNZ SHORT Lab18-03.0040756B
0040757B  F8               CLC
```

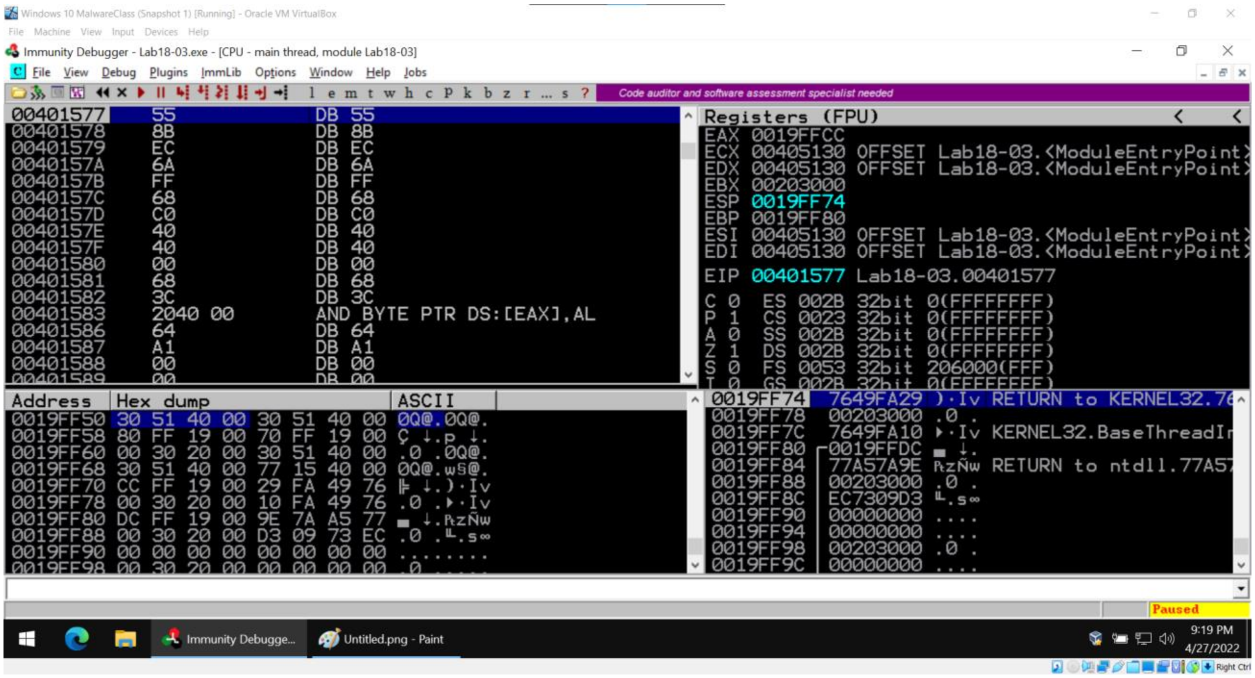
```
Registers (FPU)
EAX 0019FFCC
ECX 00405130 OFFSET Lab18-03.<ModuleEntryPoint>
EDX 00405130 OFFSET Lab18-03.<ModuleEntryPoint>
EBX 00203000
ESP 0019FF70
EBP 0019FF80
ESI 00405130 OFFSET Lab18-03.<ModuleEntryPoint>
EDI 00405130 OFFSET Lab18-03.<ModuleEntryPoint>
EIP 0040754F Lab18-03.0040754F
C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 206000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
```

```
Address   Hex dump                 ASCII
0019FF50  30 51 40 00 30 51 40 00  0Q@.0Q@.
0019FF58  80 FF 19 00 70 FF 19 00  Ç ↓.p ↓.
0019FF60  00 30 20 00 30 51 40 00  .0 .0Q@.
0019FF68  30 51 40 00 CC FF 19 00  0Q@.╠ ↓.
0019FF70  46 02 00 00 29 FA 49 76  F☻..)·Iv
0019FF78  00 30 20 00 10 FA 49 76  .0 .►·Iv
0019FF80  DC FF 19 00 9E 7A A5 77  ▄ ↓.₧zÑw
0019FF88  00 30 20 00 D3 09 73 EC  .0 .╙.s∞
0019FF90  00 00 00 00 00 00 00 00  ........
0019FF98  00 30 20 00 00 00 00 00  .0 .....
```

```
0019FF70   00000246 F☻..
0019FF74   7649FA29 )·Iv RETURN to KERNEL32.76
0019FF78   00203000 .0 .
0019FF7C   7649FA10 ►·Iv KERNEL32.BaseThreadIn
0019FF80  ┌0019FFDC ▄ ↓.
0019FF84  │77A57A9E ₧zÑw RETURN to ntdll.77A57
0019FF88  │00203000 .0 .
0019FF8C  │EC7309D3 ╙.s∞
0019FF90  │00000000 ....
0019FF94  │00000000 ....
0019FF98  │00203000 .0 .
```

```
[21:15:27] Hardware breakpoint 1 at Lab18-03.0040754F - EIP points to next instruction      Paused
```

Popa just executed, and we are now on popf. We suspect that the program has finished its unpacking, and is about to execute the tail jump. Although we don't see a long *jump* instruction specifically, we do see a push followed by a return, which does the same exact thing (return pops a value from a stack then jumps to it). In fact, we see that the value being pushed immediately before the return is 401577, which is very far away from where the program is currently executing in memory (40754F). This is probably the tail jump. Stepping into the jump puts us in a strange place:


Windows 10 MalwareClass (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Immunity Debugger - Lab18-03.exe - [CPU - main thread, module Lab18-03]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Code auditor and software assessment specialist needed
Paused
Immunity Debugge...
Untitled.png - Paint
9:19 PM
4/27/2022
Right Ctrl

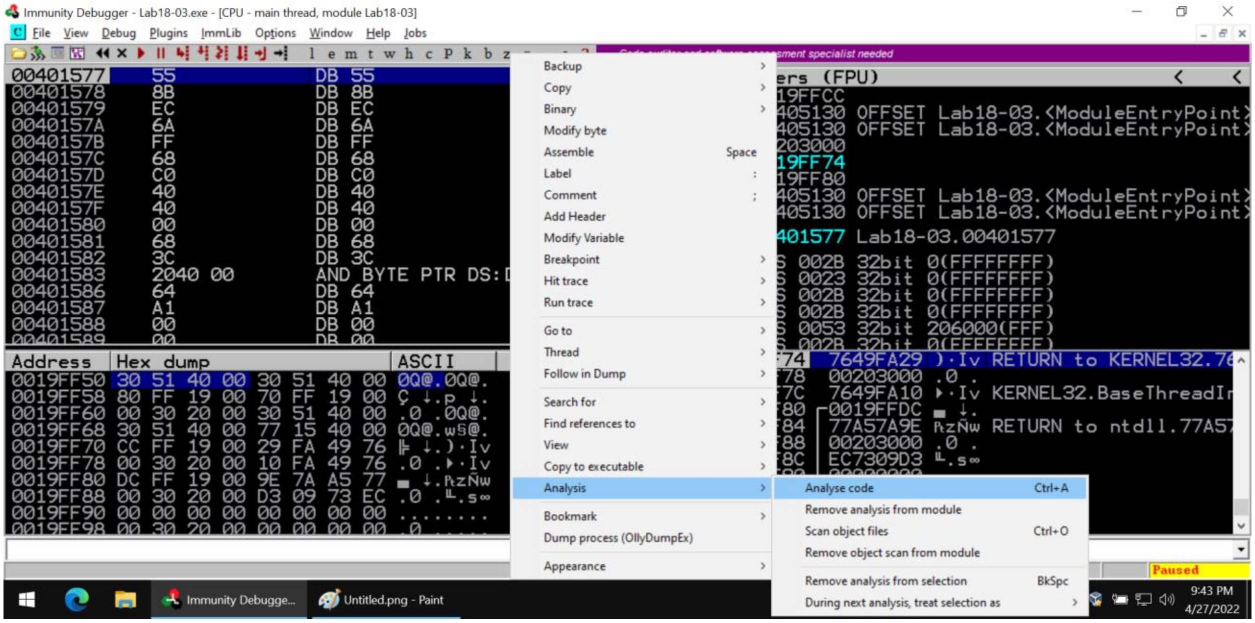
```
00401577  55          DB 55
00401578  8B          DB 8B
00401579  EC          DB EC
0040157A  6A          DB 6A
0040157B  FF          DB FF
0040157C  68          DB 68
0040157D  C0          DB C0
0040157E  40          DB 40
0040157F  40          DB 40
00401580  00          DB 00
00401581  68          DB 68
00401582  3C          DB 3C
00401583  2040 00     AND BYTE PTR DS:[EAX],AL
00401586  64          DB 64
00401587  A1          DB A1
00401588  00          DB 00
00401589  00          DB 00
```

```
Registers (FPU)
EAX 0019FFCC
ECX 00405130 OFFSET Lab18-03.<ModuleEntryPoint>
EDX 00405130 OFFSET Lab18-03.<ModuleEntryPoint>
EBX 00203000
ESP 0019FF74
EBP 0019FF80
ESI 00405130 OFFSET Lab18-03.<ModuleEntryPoint>
EDI 00405130 OFFSET Lab18-03.<ModuleEntryPoint>
EIP 00401577 Lab18-03.00401577
C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 206000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
```

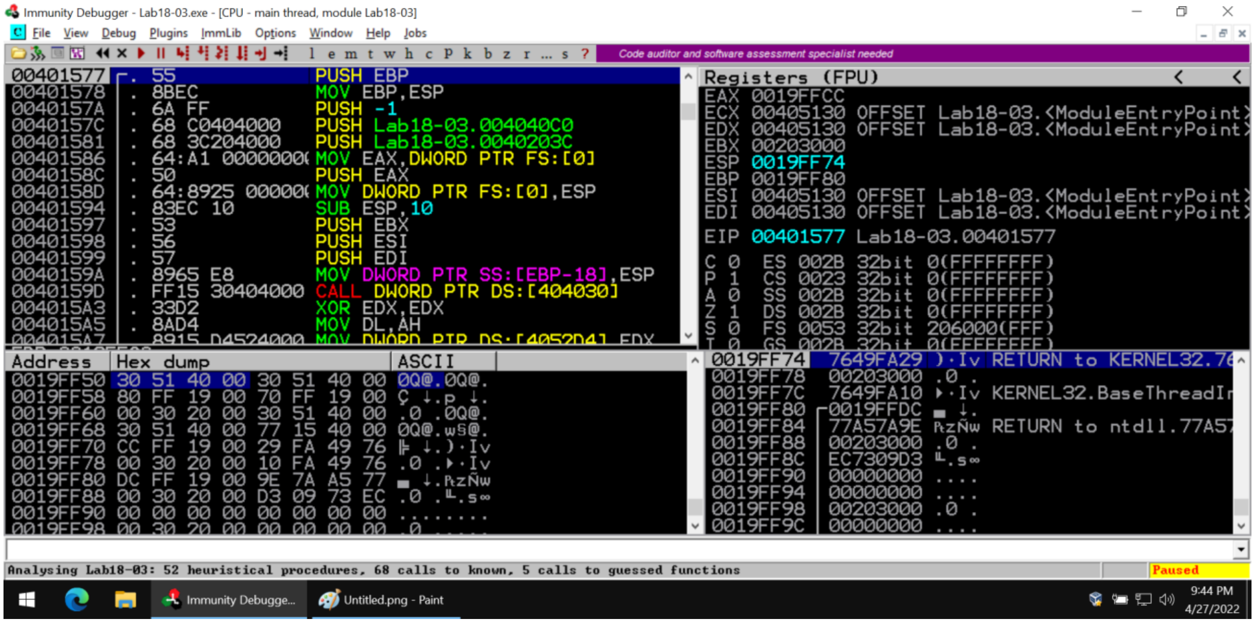
```
Address   Hex dump                 ASCII
0019FF50  30 51 40 00 30 51 40 00  0Q@.0Q@.
0019FF58  80 FF 19 00 70 FF 19 00  Ç ↓.p ↓.
0019FF60  00 30 20 00 30 51 40 00  .0 .0Q@.
0019FF68  30 51 40 00 77 15 40 00  0Q@.w§@.
0019FF70  CC FF 19 00 29 FA 49 76  ╠ ↓.)·Iv
0019FF78  00 30 20 00 10 FA 49 76  .0 .▶·Iv
0019FF80  DC FF 19 00 9E 7A A5 77  ▄ ↓.₧zÑw
0019FF88  00 30 20 00 D3 09 73 EC  .0 .╙.s∞
0019FF90  00 00 00 00 00 00 00 00  ........
0019FF98  00 30 20 00 00 00 00 00  .0 .....
```

```
0019FF74  7649FA29  )·Iv  RETURN to KERNEL32.76
0019FF78  00203000  .0 .
0019FF7C  7649FA10  ▶·Iv  KERNEL32.BaseThreadI
0019FF80 ┌0019FFDC  ▄ ↓.
0019FF84  77A57A9E  ₧zÑw  RETURN to ntdll.77A57
0019FF88  00203000  .0 .
0019FF8C  EC7309D3  ╙.s∞
0019FF90  00000000  ....
0019FF94  00000000  ....
0019FF98  00203000  .0 .
0019FF9C  00000000  ....
```

We just jumped into what immunity is telling us is part of the program that contains *data*, not code. It would be very strange for the malware to make execution jump to something that isn't executable bytecode (which in most cases will cause a crash), so we suspect that this actually IS code that immunity is displaying as data. The reason that immunity is doing this is that when it scans the program when it opens, it tries to classify different sections as code, data, etc. If it finds that a section contains data, it won't automatically change its classification if that section gets modified in memory. Instead, we need to manually tell it that this is probably code now- this can be done by right clicking on the current instruction, going to Analysis, and pressing "Analyse code" (yes, they spelled "Analyze" wrong).



This will make things a bit more readable:



Immunity Debugger - Lab18-03.exe - [CPU - main thread, module Lab18-03]

```
00401577  ┌. 55              PUSH EBP
00401578  │. 8BEC            MOV EBP,ESP
0040157A  │. 6A FF           PUSH -1
0040157C  │. 68 C0404000     PUSH Lab18-03.004040C0
00401581  │. 68 3C204000     PUSH Lab18-03.0040203C
00401586  │. 64:A1 0000000(  MOV EAX,DWORD PTR FS:[0]
0040158C  │. 50              PUSH EAX
0040158D  │. 64:8925 00000(  MOV DWORD PTR FS:[0],ESP
00401594  │. 83EC 10         SUB ESP,10
00401597  │. 53              PUSH EBX
00401598  │. 56              PUSH ESI
00401599  │. 57              PUSH EDI
0040159A  │. 8965 E8         MOV DWORD PTR SS:[EBP-18],ESP
0040159D  │. FF15 30404000   CALL DWORD PTR DS:[404030]
004015A3  │. 33D2            XOR EDX,EDX
004015A5  │. 8AD4            MOV DL,AH
```

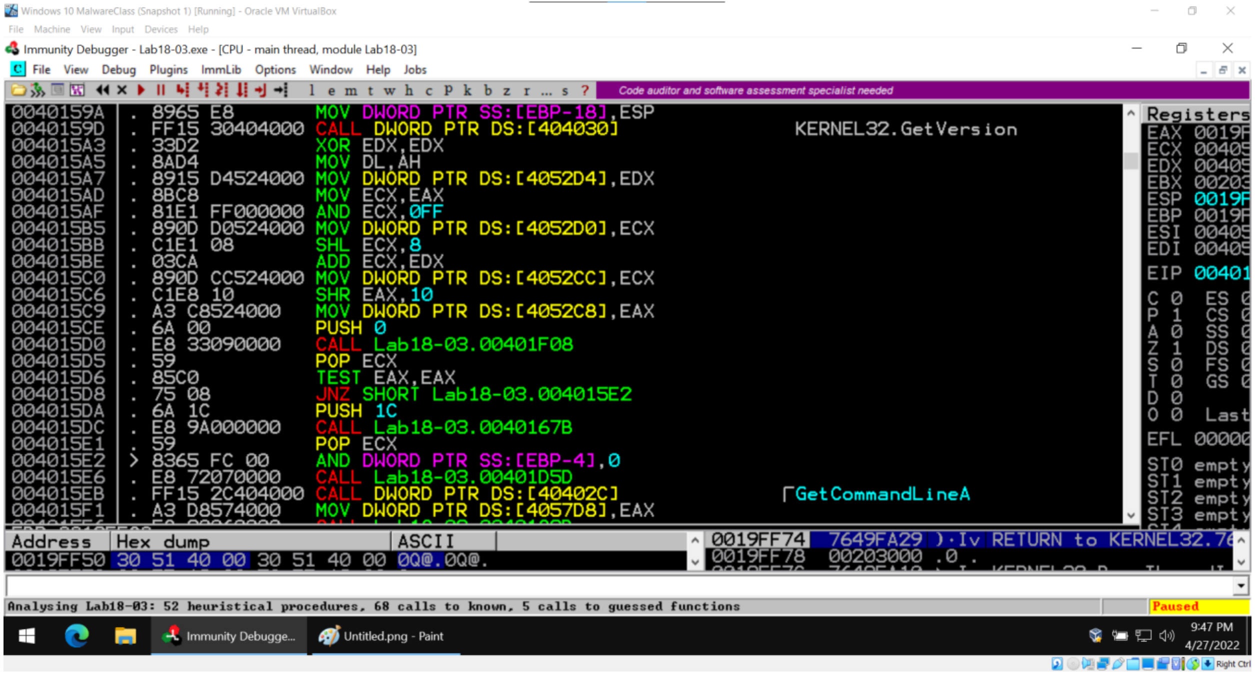
```
Registers (FPU)
EAX 0019FFCC
ECX 00405130 OFFSET Lab18-03.<ModuleEntryPoint>
EDX 00405130 OFFSET Lab18-03.<ModuleEntryPoint>
EBX 00203000
ESP 0019FF74
EBP 0019FF80
ESI 00405130 OFFSET Lab18-03.<ModuleEntryPoint>
EDI 00405130 OFFSET Lab18-03.<ModuleEntryPoint>

EIP 00401577 Lab18-03.00401577

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 206000(FFF)
```

```
Address   Hex dump                  ASCII
0019FF50  30 51 40 00 30 51 40 00  0Q@.0Q@.
0019FF58  80 FF 19 00 70 FF 19 00  Ç ↓.p ↓.
0019FF60  00 30 20 00 30 51 40 00  .0 .0Q@.
0019FF68  30 51 40 00 77 15 40 00  0Q@.w§@.
0019FF70  CC FF 19 00 29 FA 49 76  ╠ ↓.)·Iv
0019FF78  00 30 20 00 10 FA 49 76  .0 .▸·Iv
0019FF80  DC FF 19 00 9E 7A A5 77  ▬ ↓.₧zÑw
0019FF88  00 30 20 00 D3 09 73 EC  .0 .╙.s∞
0019FF90  00 00 00 00 00 00 00 00  ........
```

```
0019FF74  7649FA29  )·Iv  RETURN to KERNEL32.76
0019FF78  00203000  .0 .
0019FF7C  7649FA10  ▸·Iv  KERNEL32.BaseThreadI
0019FF80  ┌0019FFDC  ▬ ↓.
0019FF84  │77A57A9E  ₧zÑw  RETURN to ntdll.77A57
0019FF88  │00203000  .0 .
0019FF8C  │EC7309D3  ╙.s∞
0019FF90  │00000000  ....
0019FF94  │00000000  ....
0019FF98  │00203000  .0 .
0019FF9C  │00000000  ....
```

Analysing Lab18-03: 52 heuristical procedures, 68 calls to known, 5 calls to guessed functions

Now we can use OllyDumpEx to dump the unpacked executable and ImpRec to fix the import table, but before we do that, we should probably look through some of the unpacked code to see if we can figure out some things about it (though we'll need to make the code view a bit bigger).



Windows 10 MalwareClass (Snapshot 1) [Running] - Oracle VM VirtualBox

Immunity Debugger - Lab18-03.exe - [CPU - main thread, module Lab18-03]

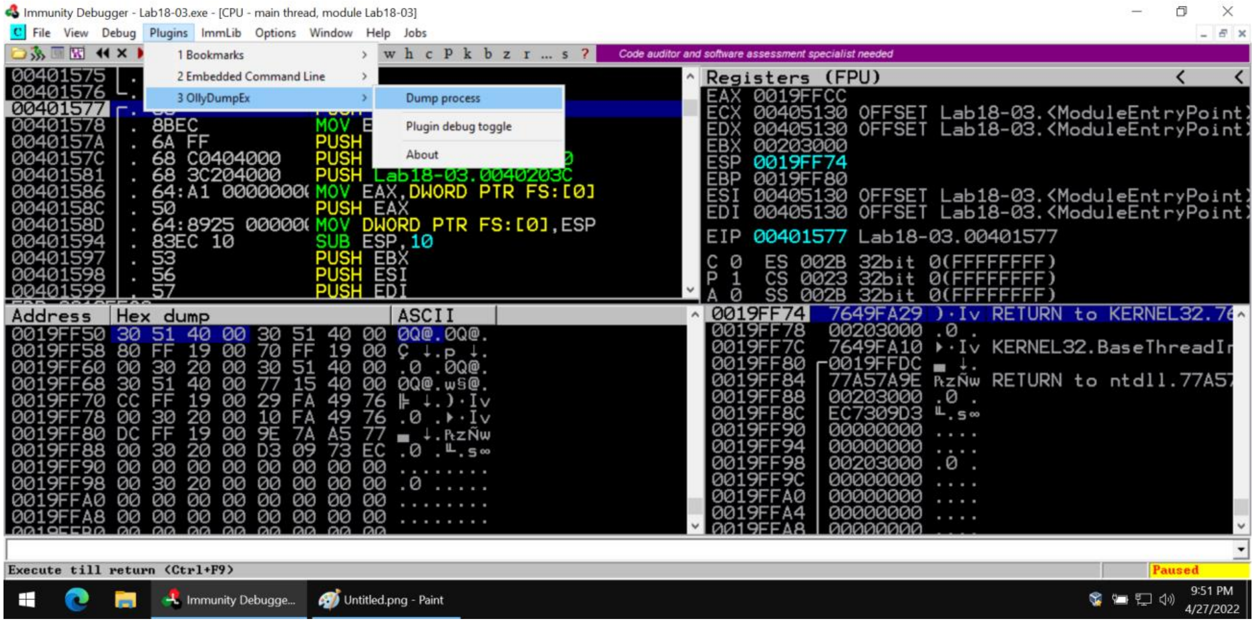
```
0040159A  . 8965 E8         MOV DWORD PTR SS:[EBP-18],ESP
0040159D  . FF15 30404000   CALL DWORD PTR DS:[404030]              KERNEL32.GetVersion
004015A3  . 33D2            XOR EDX,EDX
004015A5  . 8AD4            MOV DL,AH
004015A7  . 8915 D4524000   MOV DWORD PTR DS:[4052D4],EDX
004015AD  . 8BC8            MOV ECX,EAX
004015AF  . 81E1 FF000000   AND ECX,0FF
004015B5  . 890D D0524000   MOV DWORD PTR DS:[4052D0],ECX
004015BB  . C1E1 08         SHL ECX,8
004015BE  . 03CA            ADD ECX,EDX
004015C0  . 890D CC524000   MOV DWORD PTR DS:[4052CC],ECX
004015C6  . C1E8 10         SHR EAX,10
004015C9  . A3 C8524000     MOV DWORD PTR DS:[4052C8],EAX
004015CE  . 6A 00           PUSH 0
004015D0  . E8 33090000     CALL Lab18-03.00401F08
004015D5  . 59              POP ECX
004015D6  . 85C0            TEST EAX,EAX
004015D8  . 75 08           JNZ SHORT Lab18-03.004015E2
004015DA  . 6A 1C           PUSH 1C
004015DC  . E8 9A000000     CALL Lab18-03.0040167B
004015E1  . 59              POP ECX
004015E2  > 8365 FC 00      AND DWORD PTR SS:[EBP-4],0
004015E6  . E8 72070000     CALL Lab18-03.00401D5D
004015EB  . FF15 2C404000   CALL DWORD PTR DS:[40402C]              ┌GetCommandLineA
004015F1  . A3 D8574000     MOV DWORD PTR DS:[4057D8],EAX
```

```
Address   Hex dump                  ASCII
0019FF50  30 51 40 00 30 51 40 00  0Q@.0Q@.
```

```
0019FF74  7649FA29  )·Iv  RETURN to KERNEL32.76
0019FF78  00203000  .0 .
```

Analysing Lab18-03: 52 heuristical procedures, 68 calls to known, 5 calls to guessed functions

Specifically, the calls to GetVersion and GetCommandLineA suggest that this is a command line program. If we saw calls to RegisterClassA, LoadIconA, and FindWindowA instead, then we'd probably be dealing with a GUI based program. Most analysis at this point should be done on an unpacked binary, so let's do that by going to Plugins->OllyDumpEX->Dump process



Immunity Debugger - Lab18-03.exe - [CPU - main thread, module Lab18-03]
File View Debug Plugins ImmLib Options Window Help Jobs
1 Bookmarks
2 Embedded Command Line
3 OllyDumpEx
Dump process
Plugin debug toggle
About
Code auditor and software assessment specialist needed
Paused
Immunity Debugge...
Untitled.png - Paint
9:51 PM
4/27/2022

```
00401575  .
00401576  .
00401577  .
00401578  .  8BEC            MOV E
0040157A  .  6A FF           PUSH
0040157C  .  68 C0404000     PUSH
00401581  .  68 3C204000     PUSH Lab18-03.0040203C
00401586  .  64:A1 0000000(  MOV EAX,DWORD PTR FS:[0]
0040158C  .  50              PUSH EAX
0040158D  .  64:8925 00000(  MOV DWORD PTR FS:[0],ESP
00401594  .  83EC 10         SUB ESP,10
00401597  .  53              PUSH EBX
00401598  .  56              PUSH ESI
00401599  .  57              PUSH EDI
```

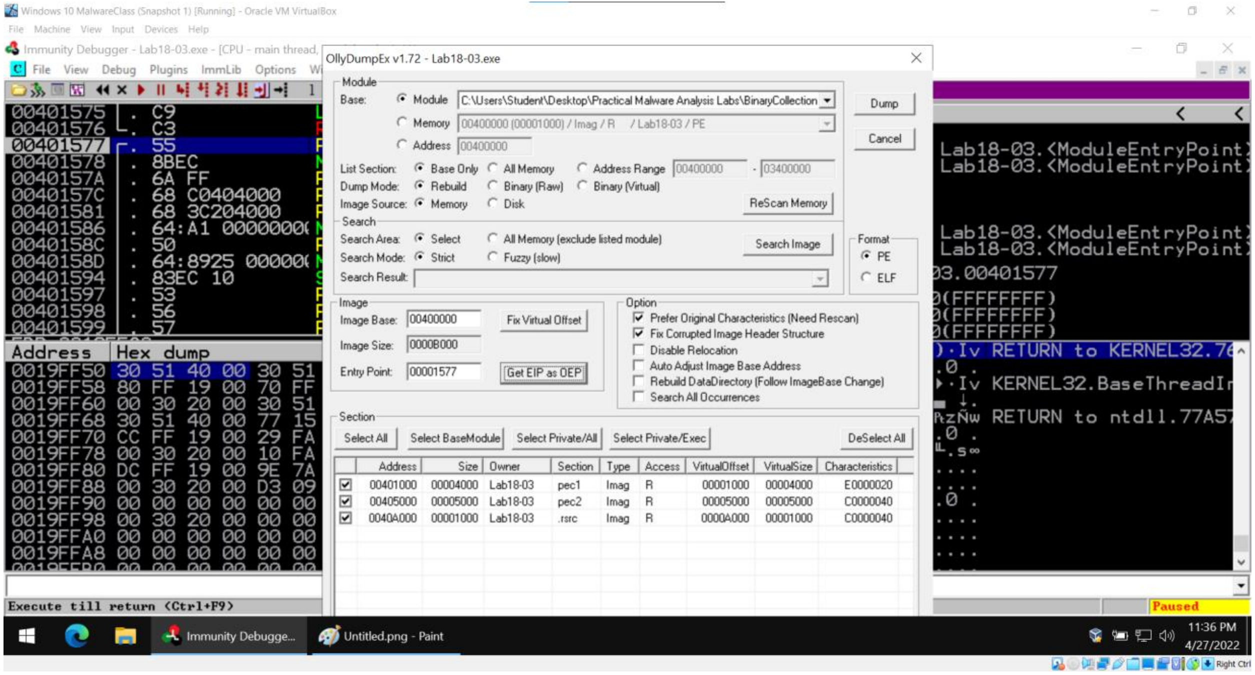
```
Registers (FPU)
EAX 0019FFCC
ECX 00405130 OFFSET Lab18-03.<ModuleEntryPoint>
EDX 00405130 OFFSET Lab18-03.<ModuleEntryPoint>
EBX 00203000
ESP 0019FF74
EBP 0019FF80
ESI 00405130 OFFSET Lab18-03.<ModuleEntryPoint>
EDI 00405130 OFFSET Lab18-03.<ModuleEntryPoint>
EIP 00401577 Lab18-03.00401577
C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
```

```
Address   Hex dump                 ASCII
0019FF50  30 51 40 00 30 51 40 00  0Q@.0Q@.
0019FF58  80 FF 19 00 70 FF 19 00  Ç ↓.p ↓.
0019FF60  00 30 20 00 30 51 40 00  .0 .0Q@.
0019FF68  30 51 40 00 77 15 40 00  0Q@.w§@.
0019FF70  CC FF 19 00 29 FA 49 76  ╠ ↓.)·Iv
0019FF78  00 30 20 00 10 FA 49 76  .0 .▶·Iv
0019FF80  DC FF 19 00 9E 7A A5 77  ▄ ↓.₧zÑw
0019FF88  00 30 20 00 D3 09 73 EC  .0 .╙.s∞
0019FF90  00 00 00 00 00 00 00 00  ........
0019FF98  00 30 20 00 00 00 00 00  .0 .....
0019FFA0  00 00 00 00 00 00 00 00  ........
0019FFA8  00 00 00 00 00 00 00 00  ........
```

```
0019FF74   7649FA29 )·Iv RETURN to KERNEL32.76
0019FF78   00203000 .0 .
0019FF7C   7649FA10 ▶·Iv KERNEL32.BaseThreadIn
0019FF80  ┌0019FFDC ▄ ↓.
0019FF84  │77A57A9E ₧zÑw RETURN to ntdll.77A57
0019FF88  │00203000 .0 .
0019FF8C  │EC7309D3 ╙.s∞
0019FF90  │00000000 ....
0019FF94  │00000000 ....
0019FF98  │00203000 .0 .
0019FF9C  │00000000 ....
0019FFA0  │00000000 ....
0019FFA4  │00000000 ....
0019FFA8  │00000000 ....
```

```
Execute till return (Ctrl+F9)
```

OllyDump needs to know the RVA (relative virtual address) of the OEP. The OEP is the Original Entry Point of the program, which is where the actual malware itself (rather than the unpacking stub) starts executing. Generally, this is the instruction jumped to by the tail jump, which in our case is 00401577. The RVA of an address in program memory is just the address minus the base address of the program (which we can see in our very first screenshot of DiE). Here, this is 401577-400000=1577. We can either manually enter this into the Entry Point field, or if execution is currently paused at the OEP (which it should be!), we can press "EIP as OEP", and OllyDump will calculate it for us.



```
00401575  .  C9
00401576  .  C3
00401577  .  55
00401578  .  8BEC
0040157A  .  6A FF
0040157C  .  68 C0404000
00401581  .  68 3C204000
00401586  .  64:A1 0000000(
0040158C  .  50
0040158D  .  64:8925 00000(
00401594  .  83EC 10
00401597  .  53
00401598  .  56
00401599  .  57
```

OllyDumpEx v1.72 - Lab18-03.exe

Module

Base: ● Module C:\Users\Student\Desktop\Practical Malware Analysis Labs\BinaryCollection

○ Memory 00400000 (00001000) / Imag / R / Lab18-03 / PE

○ Address 00400000

List Section: ● Base Only ○ All Memory ○ Address Range 00400000 - 03400000

Dump Mode: ● Rebuild ○ Binary (Raw) ○ Binary (Virtual)

Image Source: ● Memory ○ Disk

Search

Search Area: ● Select ○ All Memory (exclude listed module)

Search Mode: ● Strict ○ Fuzzy (slow)

Search Result:

Format: ● PE ○ ELF

Image

Image Base: 00400000

Image Size: 0000B000

Entry Point: 00001577

Option

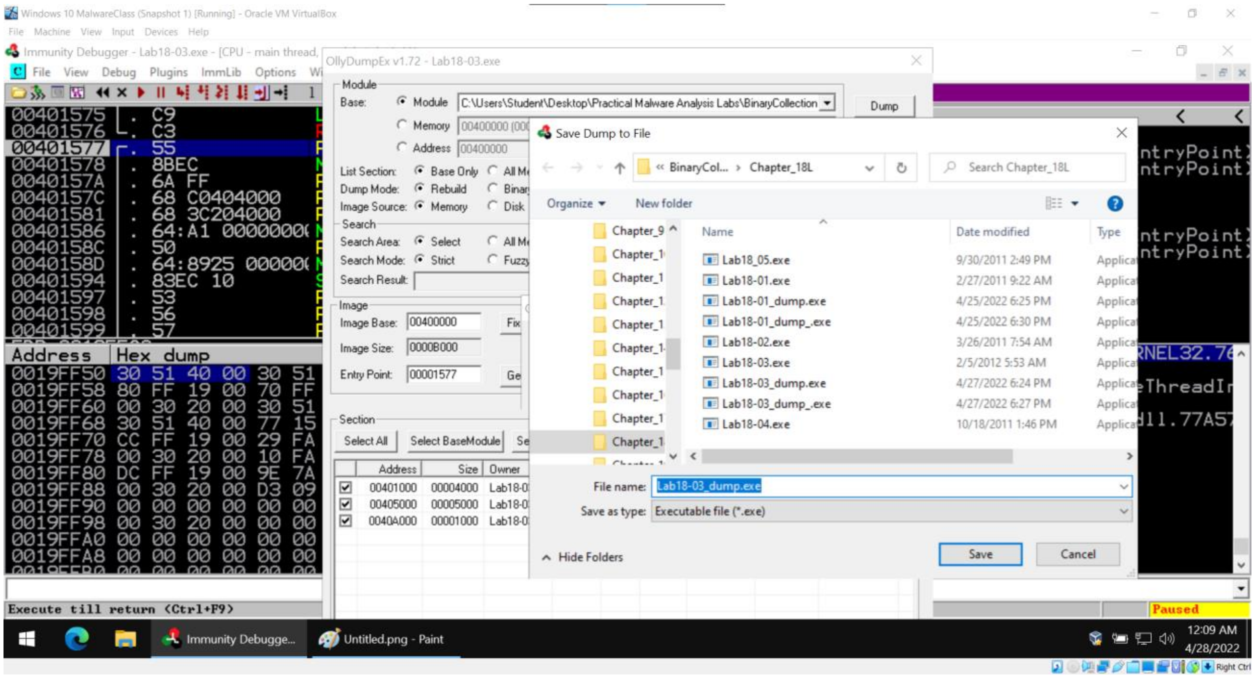
☑ Prefer Original Characteristics (Need Rescan)
☑ Fix Corrupted Image Header Structure
☐ Disable Relocation
☐ Auto Adjust Image Base Address
☐ Rebuild DataDirectory (Follow ImageBase Change)
☐ Search All Occurrences

Section

	Address	Size	Owner	Section	Type	Access	VirtualOffset	VirtualSize	Characteristics
☑	00401000	00004000	Lab18-03	pec1	Imag	R	00001000	00004000	E0000020
☑	00405000	00005000	Lab18-03	pec2	Imag	R	00005000	00005000	C0000040
☑	0040A000	00001000	Lab18-03	.rsrc	Imag	R	0000A000	00001000	C0000040

```
Execute till return (Ctrl+F9)
```

Now we just press "Dump" and save our dumped executable somewhere.



Immunity Debugger - Lab18-03.exe - [CPU - main thread,

```
00401575  L.  C9
00401576  L.  C3
00401577  ┌.  55
00401578  .   8BEC
0040157A  .   6A FF
0040157C  .   68 C0404000
00401581  .   68 3C204000
00401586  .   64:A1 0000000(
0040158C  .   50
0040158D  .   64:8925 00000(
00401594  .   83EC 10
00401597  .   53
00401598  .   56
00401599  .   57
```

```
Address   Hex dump
0019FF50  30 51 40 00 30 51
0019FF58  80 FF 19 00 70 FF
0019FF60  00 30 20 00 30 51
0019FF68  30 51 40 00 77 15
0019FF70  CC FF 19 00 29 FA
0019FF78  00 30 20 00 10 FA
0019FF80  DC FF 19 00 9E 7A
0019FF88  00 30 20 00 D3 09
0019FF90  00 00 00 00 00 00
0019FF98  00 30 20 00 00 00
0019FFA0  00 00 00 00 00 00
0019FFA8  00 00 00 00 00 00
```

Execute till return (Ctrl+F9)

OllyDumpEx v1.72 - Lab18-03.exe

Module
Base: ● Module C:\Users\Student\Desktop\Practical Malware Analysis Labs\BinaryCollection
○ Memory 00400000 (00
○ Address 00400000

List Section: ● Base Only ○ All M
Dump Mode: ● Rebuild ○ Binar
Image Source: ● Memory ○ Disk

Search
Search Area: ● Select ○ All M
Search Mode: ● Strict ○ Fuzzy
Search Result:

Image
Image Base: 00400000
Image Size: 0000B000
Entry Point: 00001577

Section

	Address	Size	Owner
☑	00401000	00004000	Lab18-0
☑	00405000	00005000	Lab18-0
☑	0040A000	00001000	Lab18-0

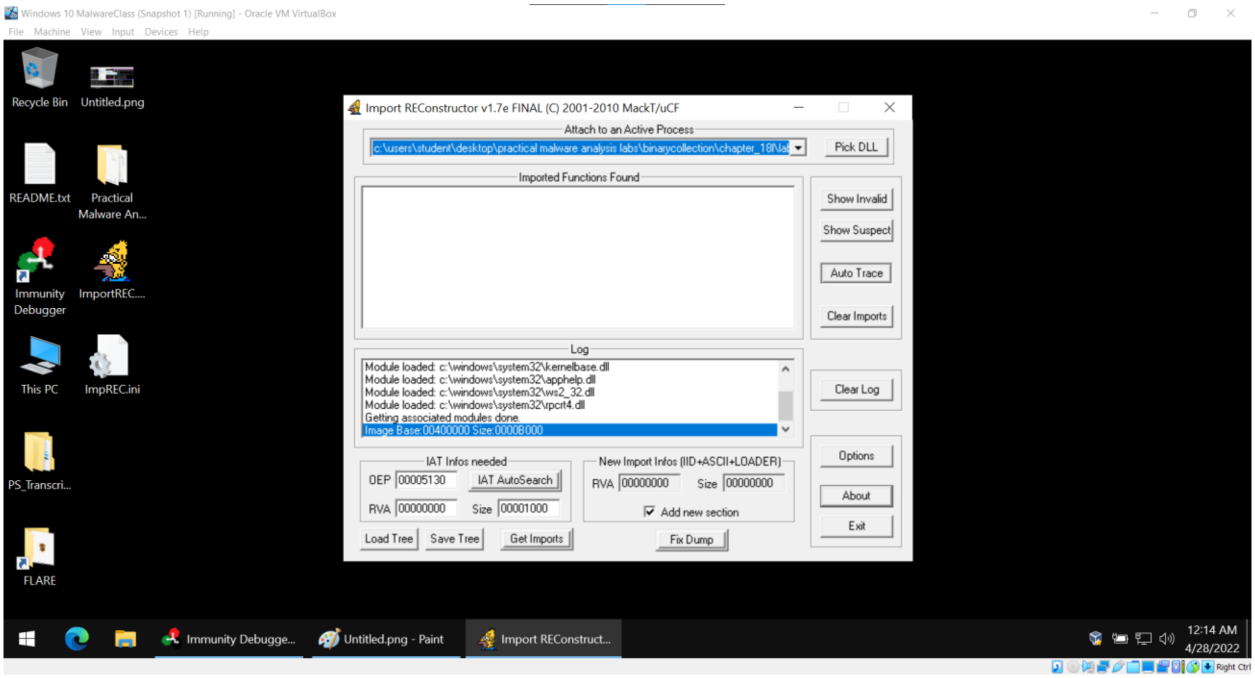
Save Dump to File

« BinaryCol... › Chapter_18L

Name	Date modified	Type
Lab18_05.exe	9/30/2011 2:49 PM	Applicat
Lab18-01.exe	2/27/2011 9:22 AM	Applicat
Lab18-01_dump.exe	4/25/2022 6:25 PM	Applicat
Lab18-01_dump_.exe	4/25/2022 6:30 PM	Applicat
Lab18-02.exe	3/26/2011 7:54 AM	Applicat
Lab18-03.exe	2/5/2012 5:53 AM	Applicat
Lab18-03_dump.exe	4/27/2022 6:24 PM	Applicat
Lab18-03_dump_.exe	4/27/2022 6:27 PM	Applicat
Lab18-04.exe	10/18/2011 1:46 PM	Applicat

File name: Lab18-03_dump.exe
Save as type: Executable file (*.exe)

Currently, the dumped executable will start running from the OEP. While this is what we want, we need to make sure that all the libraries and functions that the unpacked malware needs are imported. The unpacking stub actually dynamically loaded and linked all the relevant libraries, but the code that does this won't be executed in the dumped executable anymore (since we start at a point of the code that normally executes after the unpacking stub finishes). We need to set up an Import Address Table (IAT) that loads and links the relevant libraries and functions when the program is loaded. ImpRec lets us do just that. First, we need to select the actively running malware process that we're debugging in the "Attach to an Active Process" drop down:



Import REConstructor v1.7e FINAL (C) 2001-2010 MackT/uCF

Attach to an Active Process: c:\users\student\desktop\practical malware analysis labs\binarycollection\chapter_18l\la

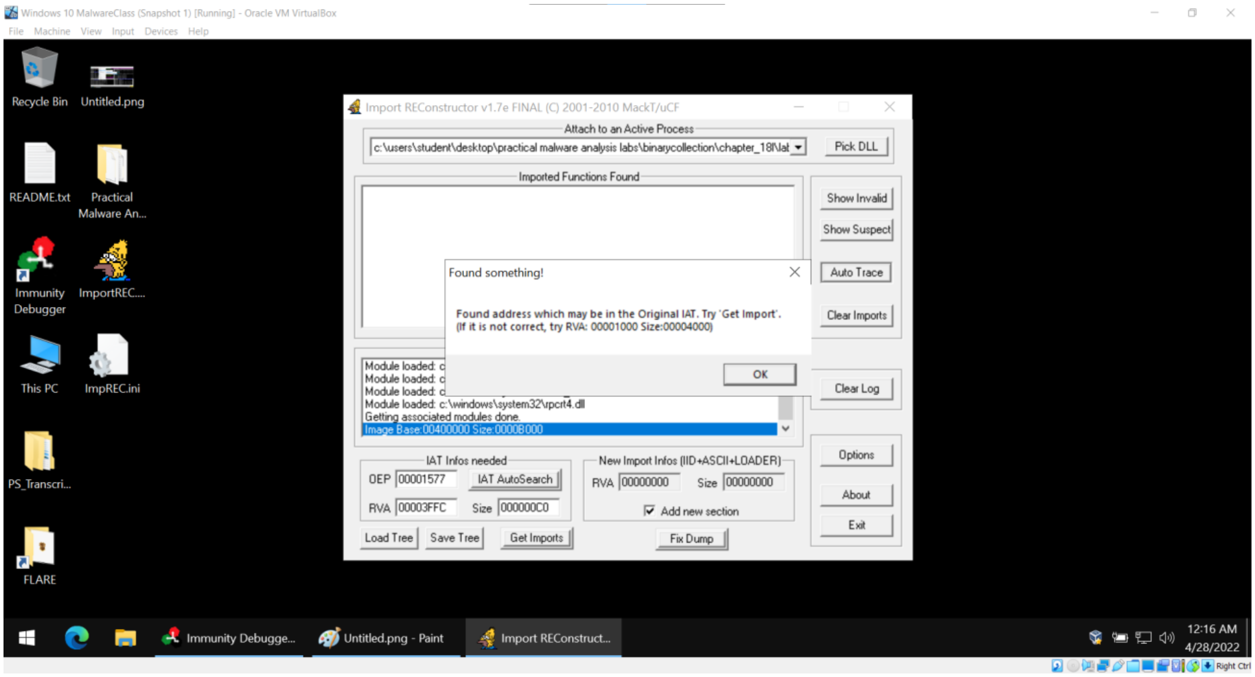
Log

```
Module loaded: c:\windows\system32\kernelbase.dll
Module loaded: c:\windows\system32\apphelp.dll
Module loaded: c:\windows\system32\ws2_32.dll
Module loaded: c:\windows\system32\rpcrt4.dll
Getting associated modules done.
Image Base:00400000 Size:0000B000
```

IAT Infos needed
OEP 00005130
RVA 00000000 Size 00001000

New Import Infos (IID+ASCII+LOADER)
RVA 00000000 Size 00000000
☑ Add new section

The process should appear as long as it's open and paused in the debugger. ImpRec will only work if the malware is paused at the OEP, so it's very important to make sure we are able to identify and get to the OEP with Immunity. ImpRec also needs the RVA of the OEP entered into the OEP field. For this malware, the RVA of the OEP was found to be 1577, so we put that in and press "IAT Autosearch":



c:\users\student\desktop\practical malware analysis labs\binarycollection\chapter_18l\lat

Found something!

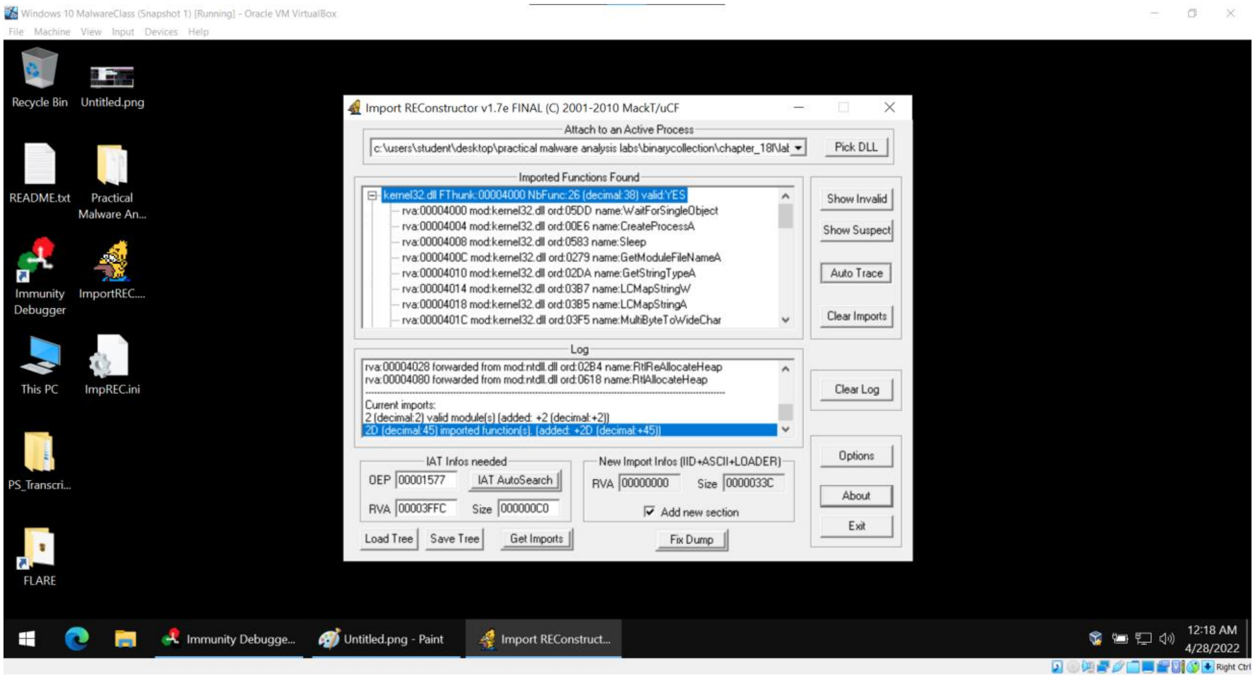
Found address which may be in the Original IAT. Try 'Get Import'.
(If it is not correct, try RVA: 00001000 Size:00004000)

```
Module loaded: c
Module loaded: c
Module loaded: c
Module loaded: c:\windows\system32\rpcrt4.dll
Getting associated modules done.
Image Base:00400000 Size:0000B000
```

OEP 00001577
RVA 00003FFC Size 000000C0

New Import Infos: RVA 00000000 Size 00000000
☑ Add new section

ImpRec uses the OEP to try and locate the malware's IAT. By pressing "Get Imports", it adds the libraries and functions that were dynamically loaded by the unpacking stub to the IAT so they are loaded automatically when the dumped program is executed:



c:\users\student\desktop\practical malware analysis labs\binarycollection\chapter_18l\lat

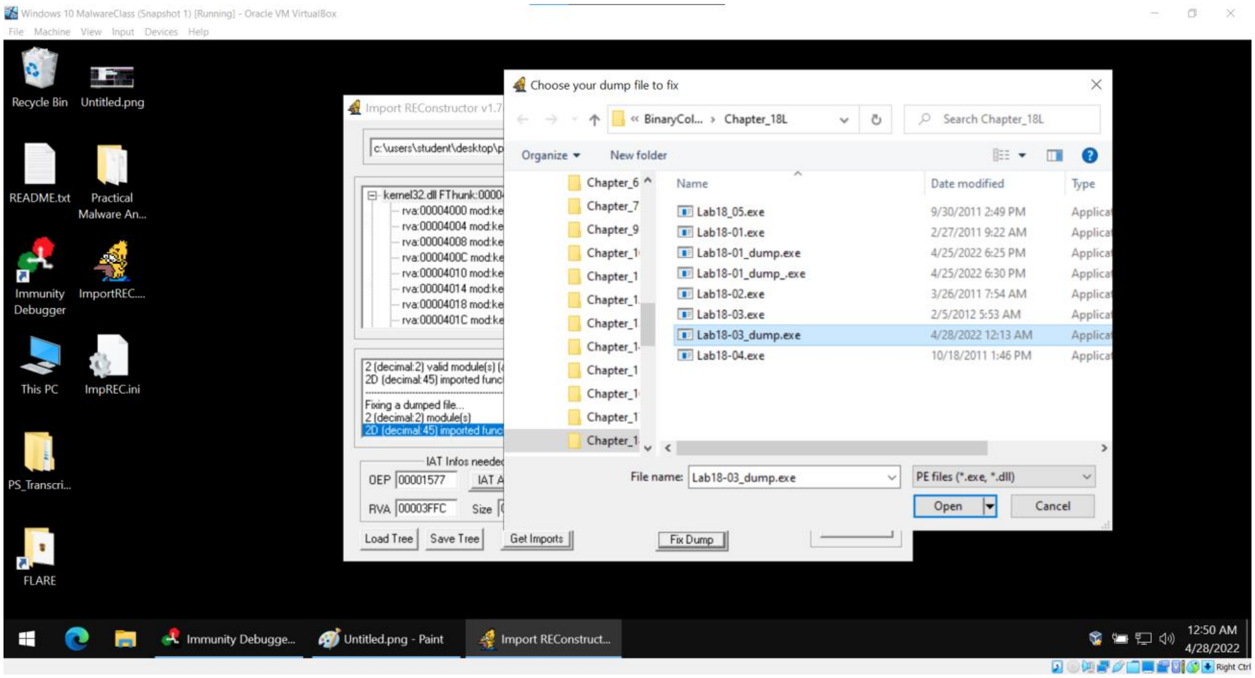
```
kernel32.dll FThunk:00004000 NbFunc:26 (decimal:38) valid:YES
    rva:00004000 mod:kernel32.dll ord:05DD name:WaitForSingleObject
    rva:00004004 mod:kernel32.dll ord:00E6 name:CreateProcessA
    rva:00004008 mod:kernel32.dll ord:0583 name:Sleep
    rva:0000400C mod:kernel32.dll ord:0279 name:GetModuleFileNameA
    rva:00004010 mod:kernel32.dll ord:02DA name:GetStringTypeA
    rva:00004014 mod:kernel32.dll ord:03B7 name:LCMapStringW
    rva:00004018 mod:kernel32.dll ord:03B5 name:LCMapStringA
    rva:0000401C mod:kernel32.dll ord:03F5 name:MultiByteToWideChar
```

```
rva:00004028 forwarded from mod:ntdll.dll ord:02B4 name:RtlReAllocateHeap
rva:00004080 forwarded from mod:ntdll.dll ord:0618 name:RtlAllocateHeap
------------------------------------------------------------------------
Current imports:
2 (decimal:2) valid module(s) (added: +2 (decimal:+2))
2D (decimal:45) imported function(s). (added: +2D (decimal:+45))
```

OEP 00001577
RVA 00003FFC Size 000000C0

New Import Infos: RVA 00000000 Size 0000033C
☑ Add new section

Note that kernel32.dll imports many more functions now than it did in the packed executable. We can save this patched executable to a file by pressing “Fix Dump” and selecting the dumped executable:



c:\users\student\desktop\p

```
⊟ kernel32.dll FThunk:0000
   rva:00004000 mod:ke
   rva:00004004 mod:ke
   rva:00004008 mod:ke
   rva:0000400C mod:ke
   rva:00004010 mod:ke
   rva:00004014 mod:ke
   rva:00004018 mod:ke
   rva:0000401C mod:ke
```

```
2 (decimal:2) valid module(s) (
2D (decimal:45) imported func
------------------------------
Fixing a dumped file...
2 (decimal:2) module(s)
2D (decimal:45) imported func
```

IAT Infos needed

OEP 00001577 IAT A

RVA 00003FFC Size

Name	Date modified	Type
Lab18_05.exe	9/30/2011 2:49 PM	Applicat
Lab18-01.exe	2/27/2011 9:22 AM	Applicat
Lab18-01_dump.exe	4/25/2022 6:25 PM	Applicat
Lab18-01_dump_.exe	4/25/2022 6:30 PM	Applicat
Lab18-02.exe	3/26/2011 7:54 AM	Applicat
Lab18-03.exe	2/5/2012 5:53 AM	Applicat
Lab18-03_dump.exe	4/28/2022 12:13 AM	Applicat
Lab18-04.exe	10/18/2011 1:46 PM	Applicat

File name: Lab18-03_dump.exe

PE files (*.exe, *.dll)

The file will have the same name as the unpacked executable, but with an underscore at the end. We can use this file to perform normal static analysis (such as opening it in DiE, IDA, or Ghidra) on the actual unpacked code of the malware.