🞇 Windows 10 MalwareClass (Sna	apshot 1) [Running] - Oracle VM VirtualBox		2 						- a ×
Detect It Easy v3.01	evices Help							15	- 0 ×
File name									
C:/Users/Student/De	esktop/Practical Malware Analys	sis Labs/BinaryCollection/	Chapter_18L/Lab18-0	03.exe					
File type		En	try point			Base address			MIME
PE32 -		_	00405130	>	Disasm	004000	00	Memory map	Hash
PE			Export	Import	Resources	.NET	TLS	Overlay	Strings
Sections	TimeDateStamp) Size	OfImage			Re	sources		Entropy
0003	> 2011-04-	30 08:26:40	00006000				Manifest	Version	Hex
Scan			Er	ndianness	Mode	Architecture		Туре	
Detect It Easy(DiE)			•	LE	32	1386		GUI	
Detect It Easy(DiE) packer			PECompact(1.68-1.8	LE 34)[-]	32	1386		GUI S ?	
Detect It Easy(DiE) packer compiler	_	Mic	PECompact(1.68-1.8 crosoft Visual C/C+-	LE 34)[-] +(6.0)[-]	32	1386		GUI S ? S	
Detect It Easy(DiE) packer compiler linker		i Mia M	PECompact(1.68-1.8 crosoft Visual C/C+ icrosoft Linker(6.0)[LE 34)[-] +(6.0)[-] GUI32]	32	1386		GUI S? S S?	
Detect It Easy(DiE) packer compiler linker		Mic	PECompact(1.68-1.6 crosoft Visual C/C+- icrosoft Linker(6.0)[LE 34)[-] +(6.0)[-] GUI32]	32	1386		GUI S? S? S?	
Detect It Easy(DiE) packer compiler linker		Mid	▼ PECompact(1.68-1.6 crosoft Visual C/C+· icrosoft Linker(6.0)[/	LE 34)[-] +(6.0)[-] GUI32]	32	1396		GUI S? S S?	Options
Detect It Easy(DiE) packer compiler linker Signatures		Mi	PECompact(1.68-1.8 crosoft Visual C/C+- icrosoft Linker(6.0)[LE 34)[-] +(6.0)[-] GUI32]	32	1396	eep scan	GUI S ? S ?	Options About
Detect It Easy(DiE) packer compiler linker Signatures		Mid Mid 100%	PECompact(1.68-1.8 crosoft Visual C/C+ - icrosoft Linker(6.0)[(LE 34)[-] +(6.0)[-] GUI32]	32	1386 D4 D4	eep scan	GUI S ? S ? Scan	Options About Exit
Detect It Easy(DiE) packer compiler linker Signatures	1DA - Lab18-03.exe	Mic M 100% Detect It Easy v3.01	PECompact(1.68-1.8 crosoft Visual C/C+- icrosoft Linker(6.0)[LE 34)[-] +(6.0)[-] GUI32]	32	1386	eep scan	GUI S? S? Scan	Options About Exit 7:25 PM 4/9 7:27 PM

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

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:

Windows 10 MalwareClass (File Machine View Input	Snapshot 1) [Running] - Oracle VM Devices Help	VirtualBox	-		28					- (3 X
Detect It Easy v3.01	Di Entropy							- 🗆	×	- 0	×
- File name C:/Users/Student	Type PE32 💌	Total 7.21808	90%	Status	Off	set 00000000	Size 00003000	Reload			
File type PE32	Entropy Bytes Regions									MI Ha	ME ash
PE	Section(0)Preset1	Name	,		Offset	Size	Entropy	Status		Stri	ngs
Sections 0003	Section(0)['pec2']				00002400	00002000	6.67710	packed		Entr	ropy
Scan Detect It Easy(Di	8 7 6									H	ex
packer compiler linker	5 4 3 2										
Signatures		2,000	4,000	6,000	8,000	10,00	0 12,00		000	Opt Ab	ions out xit
4 💽 🛤	10A - Lab18-03.exe .	. Detect It Easy	v v3.01					5		() 4)	7:31 PM /27/2022

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:

Windows 10	MalwareClass (Snapshot 1) [Running] - Oracle VM VirtualBox View Input Devices Help									o ×
Detect	De PE								- 🗆 ×	×
File nar C:/Use	Reload	_						Disasm	Readonly	
File typ	Hex Disasm Strings	Hash 64 000	4)00002b711c4	Has f7	h 32 8a9895fc					٩E
P	Memory map Entropy		iginalFirstTh	u imeDates	Stam orwarderCha	ai Name	FirstThunk	Hash		sh
Section	Heuristic scan IMAGE_DOS_HEADER ▼ IMAGE_NT_HEADERS	1	0000a03c	00000000	00000000	0000a058	0000a03c	347ca5d8	WS2_32.dll	ngs opy
Scan Detec	IMAGE_FILE_HEADER VIMAGE_OPTIONAL_HEADER IMAGE_DIRECTORY_ENTRIES Sections									<u>*x</u>
packei	Import									
compi			Thun	k 🛛	Ordinal	Hint			Na	
miker		0	0000a065			0000	LoadLibraryA			
		1	0000a074			0000	GetProcAddress			
		2	0000a085			0000	VirtualAlloc			ons
Signa		3	0000a094			0000	VirtualFree			but
		4	0000a0a2			0000	ExitProcess			
		5	0000a0b0			0000	GetModuleHandle	eA		it
	💽 📄 🐁 IDA - Lab 18-03.exe 🗈 Detect It Er	asy v3.01							영 1월 1월 40 이번 문 20 1월 1월 10	7:33 PM 4/27/2022

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:

Windows 10 MalwareClass (Snapshot 1) [Running] - Oracle VM VirtualBox		σ	×
ter machine veri span Ubrics regu IDA - Labte3 exe CUlserStudent/Deskton/Practical Malware Analysis Labs/BinaryCollection/Chapter 181/Lab18-03.exe		٥	×
ile Edit Jump Search View Debugger Options Windows Help ie Edit Jump Search View Debugger Options Windows Help ie Edit Jump Search View Debugger Options Windows Help			
H H			*
Library function Regular function Data Unexplored External symbol			
Tunctions window			
function name			
xor eax, eax			
A Graph overview			
0 Output window			<i>6</i> ×
he initial autoanalysis has been finished.			< >
U: Idle Down Disk: 56GB	∎ 💭 �\) 7:40 4/27/	PM /2022

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:

Hie Edit Jump Search Vi is in the transformed search vi is in t	ew Debugger G		windows	reip rei X	Local Wind	dows deb	ugger	• 10 🛃 🖪	* *							: [•
Library function 📘 Regula	r function 📕 Instr	uction	🛛 Data 📕 Ui	nexplored	External symbol												
Functions window	□ & ×		IDA View-A		Hex View-1	0	A	Structures	8	E 6	Enums 🛛 🖸		Imports	P	Exports		
stat	,									sub_485 mov add xchg mov sub xchg mov add add add add add add add add add ad	141 proc near eax, 45 eax, 44 eax, 6b eax, 44 eax, 6b eax, 51,000 eax, 6b eax, 6b	k_40903F 4090E6[ebp], 9 [ebp], 9 [ebp], 0 2[ebp], 0 2[ebp], 0 4090E6[el k_4090E6[el k_4090E6[el k_4090E6[el c] c]ebp], 0 c]ebp], 0	рр] рак 9990- рак ак ак ак ур] рр] ак				
A Graph overview	_ 6 ×	64.00	% (-1081,-)	81) (929	,543) 00002541 (2000000	000405	141: sub_4	05141 (S	rep mov mov retn sub_405 ynchron	sd edi, ebx 141 endp ized with B	lex Vie	w-1)				
Output window																	e ×
The initial autoanaly	sis has been	finis	hed.													 	\$
The initial autoanaly IDC	sis has been	†1n1s	hed.											 		 	

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:

M2			
Ta Windows 10 MalwareClass (Snapshot 1) [Running] - Oracle VM VirtualBox			×
rie Machine view input Devices Heip	1	-	\vee
Initianity beouger - Lab is-to-sexe - [Lef or - main thread, includie Lab is-to-s]	a	۰. 	~
The view Debug Plugins immute Options Window Help Jobs	_	-	6' X
with the second se			
00405130 EB 06 JMP SHORT Lab18-03,00405138 * Registers (FPU)		<	<
00405132 66 //150000 POST 13// EAX 0019FFC2 AFF67 AFF67 AFF67	-	n .	
00405138 9C PUSHFD ECX 00405130 OFFSET Lab18-03. (Module	Entry	POI	nt (
00405139 60 PUSHAD EBX 00203000	Entry	y=01	nt/
0040513A E8 02000000 CALL Lab18-03.00405141 ESP 0019FF74			
0040513F 33C0 XOR EAX, EAX EBP 0019FF80			
00405141 88C4 MOV EAX, SP ESI 00405130 OFFSEI Lab18-03. (Module	Entry	Poi	nt?
00405145 050 04 ADD EAX,4 EDI 00405130 OFFSET Lab18-03. (Module	Entry	γPoi	nt >
00405147 8BE3 MOV ESP.EBX EIP 00405130 Lab18-03. <moduleentrypo< td=""><td>int></td><td></td><td></td></moduleentrypo<>	int>		
00405149 8858 FC MOV EBX, DWORD PTR DS:[EBX-4] C Ø ES 002B 32bit Ø(FFFFFFF)			
0040514C 81EB 3F904000 SUB_EBX_Lab18-03.0040903F P 1 CS 0023 32bit 0(FFFFFFFF)			
00405152 870D XCHG ENP, EBX			
00405154 0055 E0504000 MOV EAX, WORD FTR SSILEDF4070E 2 1 DS 002B 32bit 0(FFFFFFF)			
00405160 66.C785 3090400(MOV WORD PTP SS-[EBP+409030] 40 5 0 FS 0053 326 (200000(FFF)			
Address Hex dump ASCII ACUPTOR AUDIT AUDIT ACUPTOR AUDIT AUDIT AUDIT ACUPTOR AUDIT AUDIT AUDIT AUDIT AUDIT ACUPTOR AUDIT ACUPTOR AUDIT AUDIT ACUPTOR AUDIT AUD	ERNEL	.32.	76 ^
00405000 00 AA 00 C3 F8 27 40 84			
00405008 2A 83 63 6D 64 80 46 06 *acmdCF+ 0019FF7C 7649FA10 + IV KERNEL32.Bas	seThr	ead	Ιr
00405010 16 00 54 42 05 12 1B 47 - TB++G 0019+F80 C019+F0C ■ +.	-11.1	7741	_
	all.	//A	27
02425020 45 80 01 1D 52 08 05 0F EURANDAR 00 0019FF8C FC7309D3			
00405038 05 01 C0 0B 1C 1D C8 19 * L + L 0019FF94 00000000			
00405040 04 E4 96 3C 8D 89 08 1C +Σú <i≅l0019ff98 .0="" .<="" 00203000="" td=""><td></td><td></td><td></td></i≅l0019ff98>			
00405048 8F 87 90 8F F2 90 1F 43 äcfà>faC 9019FF9C 000000000			~
			-
[19:49:53] Program entry point	Pau	sed	
📲 💽 📻 🛃 Immunity Debugge 😵 🖼	「「」 (*)	7:50 4/27/2	РМ 2022
n de la companya de l		3 🔇 💽	Right Ctr

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:



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:

Immunity Debuç	ger - Lab18	3- <mark>03.exe</mark> - [(CPU - ma	in thread, r	nodule La	ab18-03]												٥	×
C Eile View D	ebug Plur	gins Imm	Lib Op	tions Wir	ndow H	elp Job	DS I											-	5 X
► 🗷 🖩 🌾 🥌	× F II	6 4 2		Backup			>	k b z r s ?	Immunity: Co	onsulting Serv	vices Manager								
00405130	EB (96		Сору			>	18-03.0040513	8	^ Reg	isters	(FF	PU)					<	<
00405132	68 7	77150	20	Binary			>			EAX	0019F	FCC							
0040513/	C3			Breakpoint	E T		>	Memory, on access		ECX	00405	130	OFFSET	Lab1	8-03.	< <u>Modu</u>	leEntr	yPoi	nt
00405139	60			Search for			>	Memory, on write		EBX	00405	130	OFFSEI	Labi	8-03.	<modu< td=""><td>leEntr</td><td>yPoi</td><td>nt/</td></modu<>	leEntr	yPoi	nt/
0040513A	E8 0	02000	20	Follow DW	/ORD in D	ump		Hardware on access	>	Byte	19F	F50							
0040513F	3306	2		Go to			>	Hardware, on write	>	Word	19F	F80						-	
00405143	8300	a 04	~	Hex			>	Hardware, on execution		Dword	405	130	OFFSEI	Labl	8-03.	< Modu	leEntr	yPoi	nt
00405146	93			Text				That and the second sec		ETD	00405	130		00.00	40510	A NOGU	recitor	YPOI	nt /
00405147	8BE3	3		Short			2		47	EIP	00405	13A	Labio-	03.00	40513	A			
00405149	8836	S FC	24	Long				3-03 0040903F	41	C Ø	ES Ø	02B	32bit	Ø(FFF	FFFFF)			
00405152	87DI		Ĩ	Float			,	5-03.0010000		L Q	55 0	1023 1028	32bit	Ø(FFF	FFFFF)			
00405154	8B85	5 E690	24	Disassemt	de			PTR_SS: LEBP+	4090E	Ž 1	DS Ø	Ø2B	32bit	Ø(FFF	FFFFF	j j			
0040515A	0185	785	24	Special				<pre>SS:LEBP+4090 cc+FERD+40903</pre>	331,E	, S Ø	FS Ø	053	32bit	20600	Ø(FFF)			
Address	Hey	dump		-						^ 00 ¹	19FF50	00	405130	NOP.	OFFSE	T Lat	18-03	. KMor	
0019FF50	30 51	1 40 (ac	Appearanc	e		·	20		001	19FF54	00	405130	ØQ@.	OFFSE	ET Lat	18-03	. <mo< td=""><td>du</td></mo<>	du
0019FF58	80 FF	19 (00 7	ØFF	19 0	ØÇ	↓.p	<u>↓</u> .		001	19FF58	00	19FF80	ÇĮ.					
0019FF60	00 30	20	20 3	0 <u>51</u>	40 0	0.0	.00	J@.		00.	19FF50	00	196670	P +.					
0019FF68	30 31	2 00 0	90 C 20 2	9 FA	19 0	6 56	@.⊫	. † .		001	19FF64	õõ	405130	øõe.	OFFSE	ET Lat	518-03	. <mor< td=""><td>du</td></mor<>	du
0019FF78	00 30	ā 20 0	ð0 1	Ø FÅ	49 7	6.0		·Īv		001	19FF68	00	405130	ØQ@.	OFFSE	ET Lat	18-03	. <mo< td=""><td>du</td></mo<>	du
0019FF80	DC FF	F 19 (00 9	E 7A	A5 7	7 -	↓.Ptz	zÑw		001	19FF6C	00	19FFCC	₽ , + .					
00196688	00 36	20	00 D:	3 09	73 E	C.0		.5∞		001	19FF74	76	49FA29) · Iv	RETUR	RN to	KERNE	32.1	76
0019FF98	00 30	2 20	20 0	8 88	00 0	ø ie				× 001	19FF78	ÓŎ	203000	.0.	NET OF			-02.	~
																			-
Step over (F8	>										_		_	_			Pa	used	
• 💽		🤱 İmmuni	ity Debug	gge												S.	e 🖵 م) 9:11 F 4/27/2	PM 2022

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 (Snap	pshot 1) [Running] - Oracle VM Virtu	alBox					o ×
File Machine View Input Der	vices Help						
🔩 Immunity Debugger - Lab	18-03.exe - [CPU - main thread	l, module Lab18-03]				— (
<u>C</u> File <u>View</u> <u>Debug</u>	lugins ImmLib Options	<u>V</u> indow <u>H</u> elp <u>J</u> obs					- 8 ×
🔁 🐝 💷 🔣 🖬 🗶 🔁	∥₩₩\$ ₩ ₩₩	emtwhcPkbzrs	Code auditor and s	oftware assessment specialist neede	ėd		
00407548 ^0F8	35 9DFDFFFF	JNZ Lab18-03.004072EB	^ <u>F</u>	Registers (FPU)			< <
0040754E 9D		POPED		LAX UU19FFCC		M 1 - 1	Detet
00407550 50		PUSH FAX		ECX 00403130 0FF	-SEI LADIO-03.(ModuleEntr)	POINT
00407551 68	77154000	PUSH Lab18-03.0040157	7	EBX 00203000	-SET Lab16-05.1	rodurechtry	FOILT
00407556 C2	0400	RETN 4		ESP 0019FF70			
00407559 8BE	35 5B974000	MOV_ESI, DWORD PTR SS:	EBP+40975	EBP 0019FF80			
0040755F 0BF	-6	OR ESI,ESI		ESI 00405130 OFF	FSET Lab18-03.<	ModuleEntry	/Point>
00407561 74	18	JE SHORT Lab18-03.0040	0/5/B	EDI 00405130 OFF	FSET Lab18-03.<	ModuleEntry	/Point>
00407563 002	75 E6704000	ADD EST EDY	EBP+4090E	EIP 0040754F Lab	b18-03.0040754F		
00407568 F8		CALL Lab 18-03 0040757		0 EC 0000 201	+ the OUEEEEEEEE		
00407570 72	ØB	JB SHORT Lab18-03,0040	757D	- 0 = 5 = 0025 = 325	O(CECECECE)		
00407572 830	26 14	ADD ESI.14		A A SS AA28 324	hit Ø(FFFFFFFF)		
00407575 837	7E ØC ØØ	CMP DWORD PTR DS: [ESI+	-C],0	7 1 DS 0028 321	bit Ø(FFFFFFFF)		
00407579 ^75	FØ	JNZ SHORT Lab18-03.004	10756B	5 0 FS 0053 32L	bit 206000(FFF)		
00407578 E8	-701 0000001/		~	I 0 GS 002B 321	hit 0(FFFFFFFF)		
Address Hex	dump	ASCII	^	0019FF70 00000	246 F0		~~~^^
0019FF50 30 5	51 40 00 30 51	40 00 <mark>00@.</mark> 00@.		0019FF/4 /649F	A29) IV RETURN	N to KERNEL	32.76
0019FF58 80 F	F 19 00 70 FF	19 00 Ç↓.p↓.		00195570 76495		22 Dana Tha	
0019FF60 00 3	30 20 00 30 51	40 00 .0 .0Q@.		001955770 70495	EDC	JZ. baseinr	eadir
00195568 30 3	51 40 00 CC FF			0019FF84 77457	A9F BZNW RETURN	to ptdll	77457
00195570 46 0	02 00 00 29 FA			0019FF88 00203	3000 .0 .	to maii.	///////
0019FF80 DC F	F 19 00 9F 7A	AS 77 - B - Nu		0019FF8C EC730	903 L.s.		
0019FF88 00 3	20 20 00 03 09			0019FF90 00000	0000		
0019FF90 00 0	30 00 00 00 00	00 00		0019FF94 00000			
0019FF98 00 3	30 20 00 00 00	<u> ÃÃ ÃÃ Â</u>	~	0019FF98 00203	3000.0.		~
						100	•
[21:15:27] Hardware	breakpoint 1 at Lab	18-03.0040754F - EIP points to ne	ext instruction			Pau	sed
4 💽 🖬	4 Immunity Debugge	🞻 Untitled.png - Paint				😵 🔚 🎞 do)	9:15 PM 4/27/2022
						2	🛚 🚺 🛃 Right Ctrl

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 MalwareClas	ss (Snapshot 1) [Running] - Ori	acle VM VirtualBox		- 0 ×
File Machine View Inpu	it Devices Help			
4 Immunity Debugger	- Lab18-03.exe - [CPU - I	nain thread, module Lab18-03]		– 0 ×
C File View Debu	g <u>P</u> lugins <u>I</u> mmLib	Dp <u>t</u> ions <u>W</u> indow <u>H</u> elp <u>J</u> obs		- 5 ×
X 🕨 💹 🗐 👯 🥥	III 4 4 2 11	🚽 📲 lemtwhcPkbz	r s ? Code auditor and software assessment specialist needed	
00401577	55	DB 55	^ Registers (FPU)	((
00401578	8B	DB 8B	FAX 0019FFCC	, ,
00401579	EÇ	DB EC	ECX 00405130 OFFSET Lab18-03. <module< td=""><td>eEntryPoint></td></module<>	eEntryPoint>
0040157A	<u>64</u>	DB 6A	EDX 00405130 OFFSET Lab18-03. <modul< td=""><td>eEntryPoint></td></modul<>	eEntryPoint>
0040157B	FF		EBX 00203000	
00401570	60		ESP 0019FF74	
0040157F	40	DB 40	EBP UUI9FF80	-Estav Datat
0040157F	40	DB 40	EDI 00405130 OFFSET Lab18-03 (Modul)	eEntryPoint/
00401580	ØØ	DB 00		echtry Fornez
00401581	68	DB 68	EIP 004015// Lab18-03.004015//	
00401582	30	DB 3C	C 0 ES 002B 32bit 0(FFFFFFFF)	
00401583	2040 00	AND BYTE PIR DS:L	P 1 CS 0023 32bit 0(FFFFFFFF)	
00401587	Δ1		A 0 SS 002B 32bit 0(FFFFFFF)	
00401588	6ō	ĎB ÖØ	S 0 ES 0053 32bit 0(FFFFFFF)	
00401589	āā	กิลิ ดีดี	\sim I 0 GS 002B 32bit 0(FFFFFFF)	
Address H	ex dump	ASCII	^ 0019FF74 7649FA29) · Iv RETURN to k	(ERNEL32.76^
0019FF50 3	0 51 40 00	30 51 40 00 <mark>00@.</mark> 00@.	0019FF78 00203000 .0 .	
0019FF58 8	0 FF 19 00	70 FF 19 00 Ç ↓.p ↓.	0019FF/C /649FA10 > IV KERNEL32.Ba	selhreadlr
0019FF60 0	0 30 20 00	30 51 40 00 .0 .0Q@.	0019FF80 F0019FF0C ■ +. 0019FE84 77A57A9E B×Nuu PETUDN to r	+ 11 77457
0019550 3	0 51 40 00	// 15 40 00 000.wse. 29 FA 49 74 L).T.	0019FF88 00203000 0	ituri.//AO/
0019FF78 0	0 20 20 00	10 FA 49 76 0 . TV	0019FF8C EC7309D3 4.50	
0019FF80 D	C FF 19 00	9E 7A A5 77 $= 1.87$	0019FF90 00000000	
0019FF88 0	0 30 20 00	D3 09 73 EC .0 .⊾.s∞	0019FF94 00000000	
0019FF90 0	0 00 00 00	00 00 00 00	0019FF98 00203000 .0 .	
0019FF98 0	0 30 20 00	00 00 00 00 .0	✓ 0019FF9C 00000000	~
				•
				Paused
. 🔍 🖡	Immunity Del	bugge 🧭 Untitled.png - Paint	\$	□ □ □ □ □ □ □
				🖉 🛄 🧱 🎬 🔰 🚺 🚺 Right Ctr

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).

🔩 Immunity Debugger - Lab18-03.exe - [CPU - main thread, module Lab18-03]			- 0 ×
C Eile View Debug Plugins ImmLib Options Window Help Jobs			- 8 X
🔁 🐝 🖩 🔣 📢 🗙 🕨 🖬 🐫 📲 📲 📲 🕯 lemtwhcPkbz		sment specialist needed	
004401577 55 DB 55 004401578 8B DB 8B 004401579 EC DB EC 004401574 EA DB EC 004401574 EA DB EA 004401576 EF DB FF 004401577 CØ DB EA 004401570 CØ DB EØ 004401577 4Ø DB 4Ø 004401577 4Ø DB 4Ø 004401577 4Ø DB 4Ø 004401577 AØ DB 4Ø 004401580 ØØ DB 8Ø 004401581 68 DB 68 004401582 3C DB 3C 004401583 2040 AND BYTE PTR DS: I 004401586 64 DB 64 04 04 DB 64 004401587 A1 DB A1 04 <th>Backup Copy Binary Modify byte Assemble Sp Label Comment Add Header Modify Variable Breakpoint Hit trace Run trace Go to</th> <th><pre> ers (FPU) 19FFCC 405130 0FFSET Lab18-03. <modulee 0(fffffff)="" 0(ffffffff)="" 0ffset="" 19ff74="" 19ff80="" 203000="" 2060(fff)="" 32bit="" 405130="" 405132="" 50028="" <="" <modulee="" lab18-03.="" pre=""></modulee></pre></th> <th><pre>< < IntryPoint? IntryPoint? IntryPoint? IntryPoint?</pre></th>	Backup Copy Binary Modify byte Assemble Sp Label Comment Add Header Modify Variable Breakpoint Hit trace Run trace Go to	<pre> ers (FPU) 19FFCC 405130 0FFSET Lab18-03. <modulee 0(fffffff)="" 0(ffffffff)="" 0ffset="" 19ff74="" 19ff80="" 203000="" 2060(fff)="" 32bit="" 405130="" 405132="" 50028="" <="" <modulee="" lab18-03.="" pre=""></modulee></pre>	<pre>< < IntryPoint? IntryPoint? IntryPoint? IntryPoint?</pre>
	Thread	S 002B 32bit 0(FFFFFFF)	DNEL 22 74
Address nex dump ASCII 0019FF50 30 51 40 00 30 51 40 00 00@.00@. 0019FF58 80 FF 19 00 70 FF 19 00 C ↓.p ↓. 0019FF60 00 30 20 00 30 51 40 00 .0.00@. 0019FF60 C FF 19 00 29 FA 49 76 ↓ .).Iv 0019FF78 00 30 20 00 10 FA 49 76 .0Iv 0019FF78 00 30 20 00 10 FA 49 76 .0Iv	Follow in Dump Search for Find references to View Copy to executable	705 7649FA10 → IV KERNEL32.Bas 80 70019FFDC ↓. 84 77A5739E NZÑW RETURN to nt 88 00203000 .0. 884 00203000 .0. 886 EC7309D3 ╙.s∞	eThreadIr dll.77A57
0019FF80 DC FF 19 00 9E /A A5 // ■ +.hzNw 0019EE88 00 20 20 00 D2 09 72 EC 0 L cm	Analysis	> Analyse code Ctrl+A	
0019FF90 00 00 00 00 00 00 00 00 00 00 00 00 0	Bookmark Dump process (OllyDumpEx)	Remove analysis from module Scan object files Ctrl+0 Remove object scan from module	•
👯 💽 📄 🛃 Immunity Debugge 🛷 Untitled.png - Paint	Арреанние	Remove analysis from selection BkSpc During next analysis, treat selection as > 😪 🖘	Paused 9:43 PM 및 (가) 4/27/2022

This will make things a bit more readable:

🔩 Immunity Debugger - Lab18-03.exe - [CPU - main thre	ad, module Lab18-03]		– o ×
<u><u>C</u> <u>File View Debug Plugins ImmLib Options</u></u>	<u>W</u> indow <u>H</u> elp <u>J</u> obs		_ 8 ×
>>> >> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >>> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = >> = > = >> = >> =	lemtwhcPkbzrs? Code	uditor and software assessment specialist needed	
00401577 r. 55	PUSH EBP	^ Registers (FPU)	< <
00401578 . 8BEC_	MOV EBP, ESP	EAX 0019FFCC	
004015/A . 6A FF		ECX 00405130 OFFSET Lab18-03. (Mo	duleEntryPoint
00401570 . 68 00404000	PUSH Lab18-03.00404000	EDX 00405130 OFFSET Lab18-03. (Mo	duleEntryPoint>
00401586 64:A1 000000	MOV FAX, DWORD PTR FS: [0]		
0040158C . 50	PUSH EAX	EBP 0019FF80	
00401580 . 64:8925 00000	MOV DWORD PTR FS: [0], ESP	ESI 00405130 OFFSET Lab18-03. <mod< td=""><td>duleEntryPoint></td></mod<>	duleEntryPoint>
00401594 . 83EC 10	SUB ESP, 10	EDI 00405130 OFFSET Lab18-03. <mod< td=""><td>duleEntryPoint></td></mod<>	duleEntryPoint>
00401597 . 55		EIP 00401577 Lab18-03.00401577	
00401599 57	PUSH EDI	C = 0 ES $002B = 32bit = 0(EEEEEEE)$	
0040159A . 8965 E8	MOV DWORD_PTR_SS:[EBP-18],ESF	P = 1 CS 0023 32bit 0(FFFFFFFF)	
0040159D . FF15 30404000	CALL DWORD PTR DS: [404030]	A Ø SS ØØ2B 32bit Ø(FFFFFFF)	
004015A3 . 33D2	XOR EDX, EDX	Z 1 DS 002B 32bit 0(FFFFFFFF)	
00401545 . 0404 00401547 . 8915 D4524000	MOV DUCED PTP DS. 14052041 EDV	✓ S 0 FS 0053 32bit 206000(FFF) T 0 CS 002P 32bit 0(EEEEEE)	
Address Hex dump	ASCII	∧ 0019FF74 7649FA29) · I ∨ RETURN t	o KERNEL32.7€∧
0019FF50 30 51 40 00 30 5	1 40 00 000.000.	0019FF78 00203000 .0 .	
0019FF58 80 FF 19 00 70 F	F 19 00 Ç ↓.p ↓.	0019FF7C 7649FA10 ►·IV KERNEL32	.BaseThreadIr
0019FF60 00 30 20 00 30 5	1 40 00 .0 .0Q@.	0019FF80 F0019FFDC ■ ↓. 0019FF80 F745749E ■ ↓.	a atal11 7745
0019668 30 51 40 00 // 1	5 40 00 000.ws0.	10019FF88 1 00203000 0	o ntarr.//AJ/
0019FF70 CC FF 19 00 29 F	$A 49 76 F + . J \cdot I \vee$	0019FF8C EC7309D3 4.50	
0019FF80 DC FF 19 00 9E 7	$A A5 77 = \downarrow PrzNw$	0019FF90 00000000	
0019FF88 00 30 20 00 D3 0	9 73 EC .0 .∟.s∞	0019FF94 00000000	
0019FF90 00 00 00 00 00 00	0 00 00	0019FF98 00203000 .0 .	
<u> </u>	00 00 .0	▼ 0013FF3C 00000000	*
		. A second barrow	•
Hnalysing Labia-03: 52 heuristical pro	cedures, by calls to known, 5 calls to guess	ea functions	raused
듬 💽 📑 🛃 Immunity Debugge	🧭 Untitled.png - Paint		◎ 🖙 🖵 🕼 9:44 PM 4/27/2022

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 Virts	alBox		- a ×
File Machine View Input Devices Help			
4 Immunity Debugger - Lab18-03.exe - ICPU - main threa	. module Lab18-03]		- 0 ×
C File View Debug Plugins ImmLib Options	Nindow Help Jobs		- 8 X
	emtwhcPkbzrs? Codeaua	litor and software assessment specialist needed	
00401594 8965 F8	MOV DUODD DTD SSIFERD 181 ESD		A Desistant
00401590 FF15 30404000	CALL DWORD PTR DS: [4040301	KERNEL 32. Get Version	
004015A3 . 33D2	XOR EDX.EDX		FCX 0019
004015A5 . 8AD4	MOV DL, ÁH		EDX 00405
004015A7 . 8915 D4524000	MOV DWORD PTR DS: [4052D4], EDX		EBX 00203
004015AD . 88C8	AND ECY OFF		ESP 0019F
004015AF . 81E1 FF0000000 004015B5 8900 D0524000	MOV DWORD PTR DS: [4052D01 FCX		EBP 0019F
004015BB . C1E1 08	SHL ECX.8		ESI 00403
004015BE . 03CA	ADD ECX, EDX		
004015C0 . 890D CC524000	MOV DWORD PTR DS: [4052CC], ECX		EIP 00401
004015C6 . C1E8 10	SHR EAX, 10		C Ø ES Ø
004015C9 . A3 C8524000	DUCH O		P 1 CS Q
00401500 . E8 33090000	CALL Lab18-03.00401F08		
004015D5 . 59	POP ECX		
004015D6 . 85C0	TEST_EAX,EAX		TÕ ĠŠŎ
004015D8 . 75 08	JNZ SHORT Lab18-03.004015E2		D 0
			0 0 Last
004015F1 59	POP FCX		EFL 00000
004015E2 > 8365 FC 00	AND DWORD PTR SS:[EBP-4].0		STØ empty
004015E6 . E8 72070000	CALL Lab18-03.00401D5D		ST1 empty
004015EB . FF15 2C404000	CALL_DWORD_PTR_DS:[40402C]	FGetCommandLineA	ST2 empty
004015F1 . A3 D85/4000	MOV DWORD PTR DS: L405/D8J, EAX		 ST3 empty
Address Hex dump	ASCII	^ 0019FF74 7649FA29) · I v RETURN to	S KERNEL32.76
0019FF50 30 51 40 00 30 5:	40 00 000.000.	0019FF78 00203000 .0 .	· · ·
Analysing Lab18-03: 52 heuristical proc	edures, 68 calls to known, 5 calls to guessed	functions	Paused
	🖉 Linthia dagan - Balat		🗢 🚛 📇 🥢 9:47 PM
Timmunity Debugge	Montiled.png - Paint		🍇 🎦 🖓 4/27/2022
			🗿 🚍 🔗 🥅 📰 🕮 🕅 🔇 💽 Right Ctrl

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, mod	le Lab 18-03] — 🗇	\times
C File View Debug Plugins ImmLib Options Window	Help Jobs	- 8 ×
□ 3 = 5 + + × → 1 Bookmarks	> w h c P k b z r s ? Code auditor and software assessment specialist needed	
00401575 . 2 Embedded Command Line	> Registers (FPU) <	<
00401576 3 OllyDumpEx	Dump process EAX ØØ19FFCC	
00401577 C. OPEC MOL	ECX 00405130 OFFSEI Lab18-03. (ModuleEntryF	oint
00401578 64 FF PUS	EDX 00405130 OFFSET Lab18-03. (ModuleEntryF	oint.
0040157C . 68 C0404000 PU	H About 2 ESP 0019EF74	
00401581 . 68 3C204000 PUS	H_Lab18-03.0040203C	
00401586 . 64:A1 0000000 MO	LEAX, DWORD PIR FS: [0] ESI 00405130 0FFSEI Lab18-03. (ModuleEntryF	oint
00401580 . 64:8925 00000 MOV	DWORD PTR FS: [0].FSP EDI 00405130 OFFSET Lab18-03. (ModuleEntryF	oint.
00401594 . 83EC 10 SUE	ESP, 10 EIP 00401577 Lab18-03.00401577	
00401597 . 53 PUS	EBX C 0 ES 002B 32bit 0(FFFFFFF)	
00401598 . 56 PUS	ESI P 1 CS 0023 32bit 0(FFFFFFFF)	
	A 0 SS 00/2B 32bit 0(FFFFFFF)	
Address Hex dump	ASCII ACCAS ASCIN CONTRACTOR CONTRACTOR ASCIN	2.78^
0019FF58 80 FF 19 00 70 FF 19	0019FF7C 7649FA10 ► IV KERNEL32.BaseThree	adIr
0019FF60 00 30 20 00 30 51 40	00.0.000	
0019FF68 30 51 40 00 77 15 40	00 00@.ws@. 0019FF84 77A57A9E AzNw RETURN to ntdll.7	7A57
0019FF70 CC FF 19 00 29 FA 49		
0019FF/8 00 30 20 00 10 FA 45		
0019FF88 00 30 20 00 D3 09 73	fc 0 + L≤150 0019FF94 0000000	
0019FF90 00 00 00 00 00 00 00 00	0019FF98 00203000 .0 .	
0019FF98 00 30 20 00 00 00 00		
0019FFA0 00 00 00 00 00 00 00 00		
0019FFA8 00 00 00 00 00 00 00 00	a 19FFA8 a a a a a a a a a a a a a a a a a a a	~
		•
Execute till return (Ctrl+F9)	Pause	4
🕂 💽 📜 ⊀ Immunity Debugge 🧭	ntitled.png - Paint 😵 🖙 🖵 🗇 🧍	51 PM

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.



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

Windows 10 MalwareClass (Snapshot 1) [Running] - Oracle VM Virtual	ox				- 8 ×
File Machine View Input Devices Help	OllyDumpEx v1.72 - Lab18-03.exe		×		- 0 × _ # x
Odd Odd <th>Base: C Module C:Users/Stude C Memory 004000001000 C Address 00400000 List Section: C Base Drly C All M Dung Mode: C Rebuild C Binag Image Source: Memory C Disk Search Area: C Select C All M Search Mode: C Strict C Fuzzy Search Mode: C Strict C Fuzzy</th> <th>A Save Dump to File Save Dump to File Corganize New folder Chapter_1 Ch</th> <th>Dump</th> <th></th> <th>X ntryPoint ntryPoint 1 1 1 1 1 1 1 1 1 1 1 1 1</th>	Base: C Module C:Users/Stude C Memory 004000001000 C Address 00400000 List Section: C Base Drly C All M Dung Mode: C Rebuild C Binag Image Source: Memory C Disk Search Area: C Select C All M Search Mode: C Strict C Fuzzy Search Mode: C Strict C Fuzzy	A Save Dump to File Save Dump to File Corganize New folder Chapter_1 Ch	Dump		X ntryPoint ntryPoint 1 1 1 1 1 1 1 1 1 1 1 1 1
00401594 . 83EC 10 53 00401597 . 53 . 00401598 . 56 . 00401599 . 57 . Address Hex dump . 0019FF50 30 51 40 00 30 51 0019FF58 80 FF 19 00 30 51 0019FF58 80 FF 19 00 30 51 0019FF58 80 FF 19 00 30 51 0019FF50 30 51 40 00 30 51 0019FF50 30 51 40 00 30 51 0019FF60 30 51 40 00 77 15 0019FF70 CC FF 19 00 29 FA	Search Hesuit Image Image Base: 00000000 Frix Image Size: 00000000 Entry Point: 00001577 Ge Section Select All Select All	Chapter_1 ILab18-01_dump.exe Chapter_1 ILab18-01_dump_exe Chapter_1 ILab18-01_dump_exe Chapter_1 ILab18-02_exe Chapter_1 ILab18-03_dump_exe Chapter_1 ILab18-03_dump_exe Chapter_1 ILab18-03_dump_exe		2/27/2011 9:22 AM 4/25/2022 6:25 PM 4/25/2022 6:30 PM 3/26/2011 7:54 AM 2/5/2012 5:53 AM 4/27/2022 6:24 PM 4/27/2022 6:27 PM 10/18/2011 1:46 PM	Applicat Applicat Applicat Applicat Applicat Applicat Applicat Applicat
0019FF78 00 30 20 00 10 FA 0019FF80 DC FF 19 00 9E 7A 0019FF88 00 30 20 00 D3 09 0019FF88 00 30 20 00 00 00 0019FF98 00 30 20 00 00 00 0019FFA8 00 30 20 00 00 00 0019FFA8 00 00 00 00 00 00 0019FFA8 00 00 00 00 00 00 0019FFA8 00 00 00 00 00 00	Address Size Owner ✓ 00401000 00004000 Lab18-0 ✓ 00405000 00005000 Lab18-0 ✓ 00404000 00001000 Lab18-0	File name: Lab18:03.dump.exe Save as type: Executable file (*.exe)		Save Can	Cel at a cel
Execute till return (Ctrl+F9)					Paused
🔲 💽 ᇘ ⊀ Immunity Debugge	🗊 Untitled.png - Paint				Sim to a state of the state
					🚈 🛃 🥟 🛄 🧱 🚝 🕼 🚱 💽 Right Ctrl

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:

Windows 10 MalwareClass (Snapshot 1) [Running] - Oracle VM Virtu File Machine View Input Devices Help	alBox		- o ×
Recycle Bin Untitled.png			
	Attach to an Active Process		
	c:\users\student\desktop\practical malware analysis labs\binarycollection\chapter_18l\lat	Pick DLL	
	Imported Functions Found		
README.txt Practical Malware An		Show Invalid	
		Show Suspect	
- 💐 🛛 🔬			
		Auto Trace	
Debugger		Clear Imports	
	Log		
This PC ImpREC.ini	Module loaded: c:\windows\system32\apphelp.dll	Clear Log	
	Module loaded: c:\windows\system32\rpcrt4.dll Gatting associated modules.does		
-	Image Base:00400000 Size:0000B000		
10	IAT Infos needed New Import Infos (IID+ASCII+L0ADER)	Options	
PS_Transcri	0EP 00005130 IAT AutoSearch RVA 00000000 Size 00000000		
	RVA 00000000 Size 00001000	About	
	Load Tree Save Tree Get Imports Fix Dump	Exit	
7			
FLARE			
🕂 💽 📄 🥂 Immunity Debugge	🐼 Untitled.png - Paint 🛛 🦂 Import REConstruct		Signed Content of the second secon
			🔽 💿 📜 🚍 🖉 🥅 🌉 🚰 💟 🚫 💽 Right Ctrl

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":

Windows 10 MalwareClass (Snapshot 1) [Running] - Oracle VM VirtualBox			- 0 ×
File Machine View Input Devices Help			
Recycle Bin Untitled.png	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\lat 💌	Pick DLL	
	Imported Functions Found		
README.txt Practical		Show Invalid	
Malware An		Show Suspect	
			
***	Found something!	Auto Trace	
Immunity ImportREC			
Debugger	Found address which may be in the Original IAT. Try 'Get Import'. (If it is not correct, try RVA: 00001000 Size:00004000)	Clear Imports	
	Module loaded: c OK		
This PC ImpREC.ini	Module loaded: c	Clear Log	
	Getting associated modules done.		
	Image base.00400000 Size.0000b000		
	IAT Infos needed New Import Infos (IID+ASCII+LOADER)	Uptions	
PS_Transcri	0EP 00001577 <u>IAT AutoSearch</u> RVA 00000000 Size 00000000	About	
	RVA 00003FFC Size 000000C0 V Add new section		
	Load Tree Save Tree Get Imports Fix Dump	Exit	
FLARE			
🕂 💽 📄 🥂 Immunity Debugge 🧭 Un	titled.png - Paint 🥠 Import REConstruct	S	■ に di) 12:16 AM 4/28/2022
		🗵 💿 💯 📑 /	🖉 🔚 🔚 🔡 🚺 😵 Right Ctrl

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:

🔀 Windows 10	MalwareClass (Snapshot 1) [Running] - Oracle VM Virtual	3ox		- 0 ×
File Machine	View Input Devices Help			
3				
Recycle Bin	Untitled.png	HIMPORT REConstructor v1.7e FINAL (C) 2001-2010 MackT/uCF		
		Attach to an Active Process		
	12n	c:\users\student\desktop\practical malware analysis labs\binarycollection\chapter_18l\lat_	Pick DLL	
		Imported Functions Found		
README.txt	Practical Malware An	E- kernel32 dll FT hunk: 00004000 NbFunc: 26 (decimal: 38) valid/YES	Show Invalid	
	<u></u>	 rva:00004004 mod:kernel32.dll ord:00E6 name:CreateProcessA rva:00004008 mod:kernel32.dll ord:0583 name:Sleep 	Show Suspect	
2		 rva:0000400C mod:kernel32.dll ord:0279 name:GetModuleFileNameA rva:00004010 mod:kernel32.dll ord:020A name:GetStringTypeA 	Auto Trace	
Immunity	ImportREC	 rva:00004014 mod:kernel32.dll ord:0387 name:LCMapStringW rva:00004018 mod:kernel32.dll ord:0385 name:LCMapStringA 		
Debugger		- rva:0000401C mod:kernel32.dll ord:03F5 name:MultiByteToW/ideChar 🗸	Clear Imports	
		Log		
	1 C	rva:00004028 forwarded from mod.ntdll.dll ord:0284 name:RtiReAllocateHeap		
This PC	ImpREC.ini		Clear Log	
		2 (decimal:2) valid module(s) (added: +2 (decimal:+2)) 20 (decimal:4) imported function(s), (added: +20 (decimal:+45))		
		IAT Infra nanded New Import Infra IIID (ASCIL) DADED	Options	
DS Transcri		OEP 00001577 IAT AutoSearch RVA 0000000 Size 0000030		
r5_nansen			About	
		Add new section	Exit	
		Load Tree Get Imports Fix Dump		
PLADE				
PLAKE				
	n n n n n n n n n n n n n n n n n n n	🖬 Hatilad pag - Paint 🦂 Import REConstruct		12:18 AM
		mport deconsuite		4/28/2022

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:

Windows 10 MalwareClass (Snapshot 1) (Running) - Oracle VM VirtualBox		o x
Recycle Bin Untitled.png	1.7 ← → → ↑ 📙 « BinaryCol » Chapter_18L 🗸 & 🖏 🔎 Search Chapter_18L	
C:\users\student\desktop	Organize - New folder	
README.txt Practical Malware An Importantial and the second s	Object Chapter_6 ^ Name Date modified Type Object Chapter_7 II Lab18_05.exe 9/30/2011 2:49 PM Applicat Kek Chapter_9 II Lab18-01_dump.exe 2/27/2011 9:22 AM Applicat Kek Chapter_1 II Lab18-01_dump.exe 4/25/2022 6:30 PM Applicat Kek Chapter_1 II Lab18-01_dump.exe 4/25/2022 6:30 PM Applicat Kek Chapter_1 II Lab18-01_dump.exe 3/26/2011 7:54 AM Applicat Kek Chapter_1 II Lab18-03_dump.exe 4/25/2022 6:30 PM Applicat Kek Chapter_1 II Lab18-03_dump.exe 4/26/2021 7:53 AM Applicat II Lab18-03_dump.exe 10/18/2011 1:46 PM Applicat II Lab18-04.exe 10/18/2011 1:46 PM Applicat II Lab18-04.exe 10/18/2011 1:46 PM Applicat II Lab18-04.exe 10/18/2011 1:46 PM Applicat	
PS_transcrit.	der T A File name: Lab18-03_dump.exe V PE files (*.exe, *.dll) V a (Get Imports Fix Dump	
FLARE	السومات المراجع الم المراجع المراجع ال المراجع المراجع الم المراجع المراجع المراجمع المر المراجع المراجع المراجع المراجع المراجع المراجع المر	12:50 AM 4/28/2022
		🚫 🛃 Right Ctrl

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.