# Creating Trojans

By embedding shellcodes inside PE exe files

## What are Trojans?

- A fake program that pretends to be something it is not
- Inside it, there is some hidden code that does something else, eg.
  - A listening port, or,
  - Connecting to a server, or, alternatively, reverse connection shell
  - Capturing keystrokes (keylogger)
  - Stealing usernames and passwords
  - Spying on screen
  - Download additional malicious tools
  - Spreading to other machines
  - Escalate privileges to become admin user
  - Encrypt files (ransomware)
- Remote Access Tool (RAT)

## Techniques to Create Trojans

- Insert malicious code inside Code Caves
- Create new Sections to put malicious code
- Extend existing Sections to put malicious code

#### Creating Trojan via Code Caves

## Objective for Code Cave Project

• To trojanize CrackMe1.exe to run mspaint.exe

	🧭 📮 🤊 🥐 =   Untitled - Paint —	□ ×
	File Home View	<u>^</u>
	📋 🗆 \land 🎑 🤫 🗮 🔳 🌍	
CrackMe1 by crackinglessons.com X	Clipboard Image Tools Brushes Shapes Size Colors Edit with Paint 3D	
Please enter the Serial Key:		^
Check About		
	4	~
	+ 100% 🕞 —	• •

#### How to choose a suitable exe to trojanize

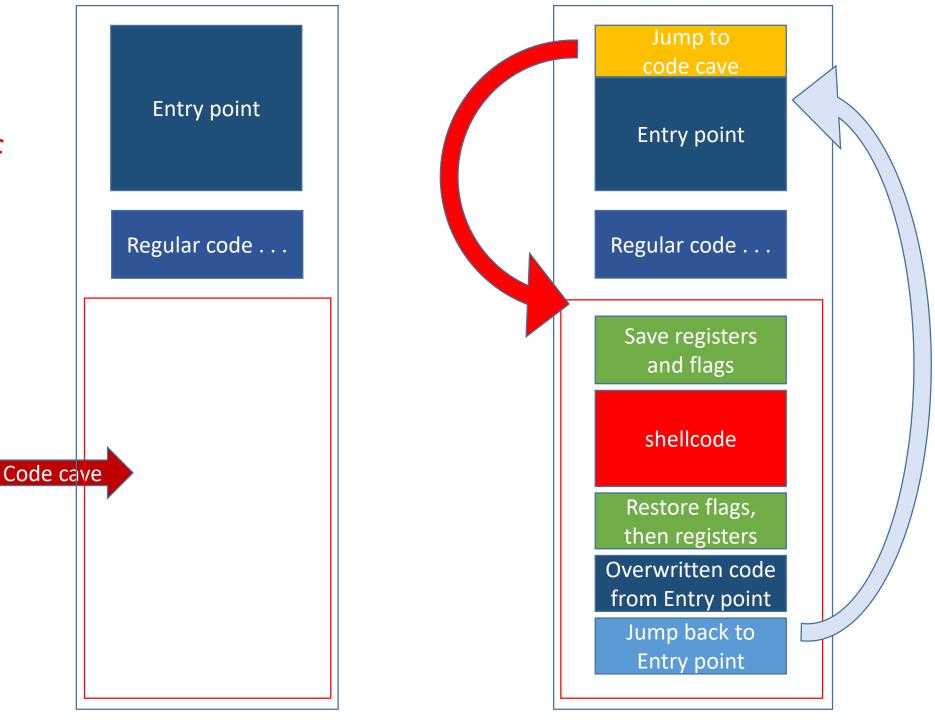
- For code cave, you need a large .TEXT section
- Raw size and Virtual size are different
- Raw size is file size, Virtual size is memory size
- When File is run, OS will map it to virtual memory and give it more memory space for optimization purposes.
- Check raw code cave size use PE-Bear and HxD hexeditor
- Our shellcode will be slightly < 200 bytes, so we need about 300 bytes of code cave
- If cannot find enough code cave, then need to use other methods

#### Checking for code cave size

PE-bear v0.5.0	-	D - [C:\Users\pc\Desktop\crackMe1.exe]	
File Settings Compare Info		ile Edit Search View Analysis Tools Window Help	
CrackMe1.exe	^ x 🔿 🗐 🖉 🄊 🏓 🐑 🎓	🕂 🔂 🗸 🔄 📓 🖉 🚽 📴 🕶 16 🕒 Windows (ANSI) 💽 hex 🗨	
😕 DOS Header	<sup>57</sup> 0 1 2 3 4 5 6 7 8 9 A B C D E F 0 1 2 3 4	5 6 7 8 9 A B C A rackMe1.exe	
DOS stub	590 00 E8 CB 2C 00 00 8B F0 E8 5E 05 00 00 6A 01 89 . è Ë , .	. a è ^	
MT Headers	5A0 06 E8 EF 02 00 00 83 C4 0C 5E 84 C0 74 73 DB E2 . È Ï	Fset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F Decoded text	
😕 Signature	5B0 E8 AD 07 00 00 68 8E 19 40 00 E8 8F 04 00 00 E8	0@.è 00B5F0 00 00 75 EB 66 0F 6E C2 83 C7 10 66 0F 3A 63 47uëf.nÂfÇ.f	.:cG
🦐 File Header	5C0 3A 05 00 00 50 E8 C6 25 00 00 59 59 85 C0 75 51	00B600 F0 40 8D 4C 39 F0 0F 42 C1 75 ED 5F C9 C3 B8 F0 80.198.BÁuí 1 00B610 FF FF FF 23 C7 66 0F EF C0 66 0F 74 00 B9 0F 00 999#cf.iÅf.t	£A,0
🦻 Optional Header	5D0 E8 33 05 00 00 E8 88 05 00 00 85 C0 74 0B 68 FB è 3	00B610 FF FF FF 23 C7 68 0F FF C0 68 0F 74 00 BS 0F 60 9YY#ÇI.AA.C	
Section Headers	5E0 16 40 00 E8 93 22 00 00 59 E8 4A 05 00 00 E8 45	00B630 FA 75 14 66 0F EF C0 66 0F 74 40 10 83 C0 10 66 úu.f.iÀf.t@.	
A Sections	5F0 05 00 00 E8 1F 05 00 00 E8 FE 04 00 00 50 E8 C6 <	• 00B640 0F D7 F8 85 FF 74 EC 0F BC D7 03 C2 EB BD 8B 7D .×ø…ÿtì.₩×.Â	
🔺 韂 .text	Disasm: .text General DOS Hdr Rich Hdr File Hdr Optional Hdr Sectio	00B650 08 33 C0 83 C9 FF F2 AE 83 C1 01 F7 D9 83 EF 01 .3ÀfÉÿð@fÁ.÷	
📫 EP = 7BF		- HAN OF BOUCH BAR 15 CO FD FZ AE 05 C/ OI 55 C/ H OF 55 CO ED SELYOBJÇ. C.C	.3Aë
📲 .rdata	+ 2	00B670 02 8B C7 FC 5F C9 C3 00 00 00 00 00 00 00 00 00 00B680 00 00 00 00 00 00 00 00 00 00 00 00 0	
📲 .data	Name Raw Addr. Raw size Virtual Addr. Virtual Size Characteristics		
.rsrc	▶ .text 400 B400 1000 B277 60000020	0 0 00 00 00 00 00 00 00 00 00 00 00 00	
	▷ .rdata B800 5A00 D000 59CA 40000040	0 00B6B0 00 00 00 00 00 00 00 00 00 00 00 00 0	
		0 0086C0 00 00 00 00 00 00 00 00 00 00 00 00 0	
	▶ .rsrc 11C00 17200 15000 17200 40000040	0 00B6D0 00 00 00 00 00 00 00 00 00 00 00 00 0	
		00B6F0 00 00 00 00 00 00 00 00 00 00 00 00 0	
		008700 00 00 00 00 00 00 00 00 00 00 00 00	
	( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( )	• 00B710 00 00 00 00 00 00 00 00 00 00 00 00 0	
	Raw & Virtual	00B720 00 00 00 00 00 00 00 00 00 00 00 00 0	
	-1999		
	[.text] [.text]		
		008760 00 00 00 00 00 00 00 00 00 00 00 00 0	
	B800 [.rdata] D000 [.rdata]	00 00 00 00 00 00 00 00 00 00 00	
	11200	00 00 00 00 00 00 00 00 00 00 00 00 00	
	[;4946] [;4946] [:15000 [;4946]]	008790 00 00 00 00 00 00 00 00 00 00 00 00 0	
	S S S S S S S S S S S S S S S S S S S	00B7A0 00 00 00 00 00 00 00 00 00 00 00 00 0	
	CrackMeL exe		
	ž	0087D0 00 00 00 00 00 00 00 00 00 00 00 00 0	
		00B7E0 00 00 00 00 00 00 00 00 00 00 00 00 0	

0xB7FF - 0xB680 = 0x17F = 383

Anatomy of code cave execution flow



# Step-by-step

- 1. Use Metasploit in Kali Linux to generate a 32-bit shellcode that can launch mspaint.exe
- 2. Test the shellcode using shellcoderunner
- 3. Open xdbg 32-bit version and identify address of code cave
- 4. Copy out first few lines of Entry Point
- 5. Insert jmp to code cave at start of EntryPoint (use fill with NOPs)
- Notice how many instructions overwritten (you need to insert them in code cave below\*)
- 7. Save registers using pushad
- 8. Save flags using pushfd
- 9. Insert shellcode

# Step-by-step (2)

- 10. Restore flags using popfd
- 11. Restore registers using popad
- 12. Insert overwritten instructions\*
- 13. Insert jmp back to beginning of Entry Point just after the jmp to codecave
- 14. Patch and save to file
- 15. Test and debug to see where shellcode cause program to exit
- 16. Assemble a jmp to bypass exit and go to codecave
- 17. Patch and save to final file

