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)

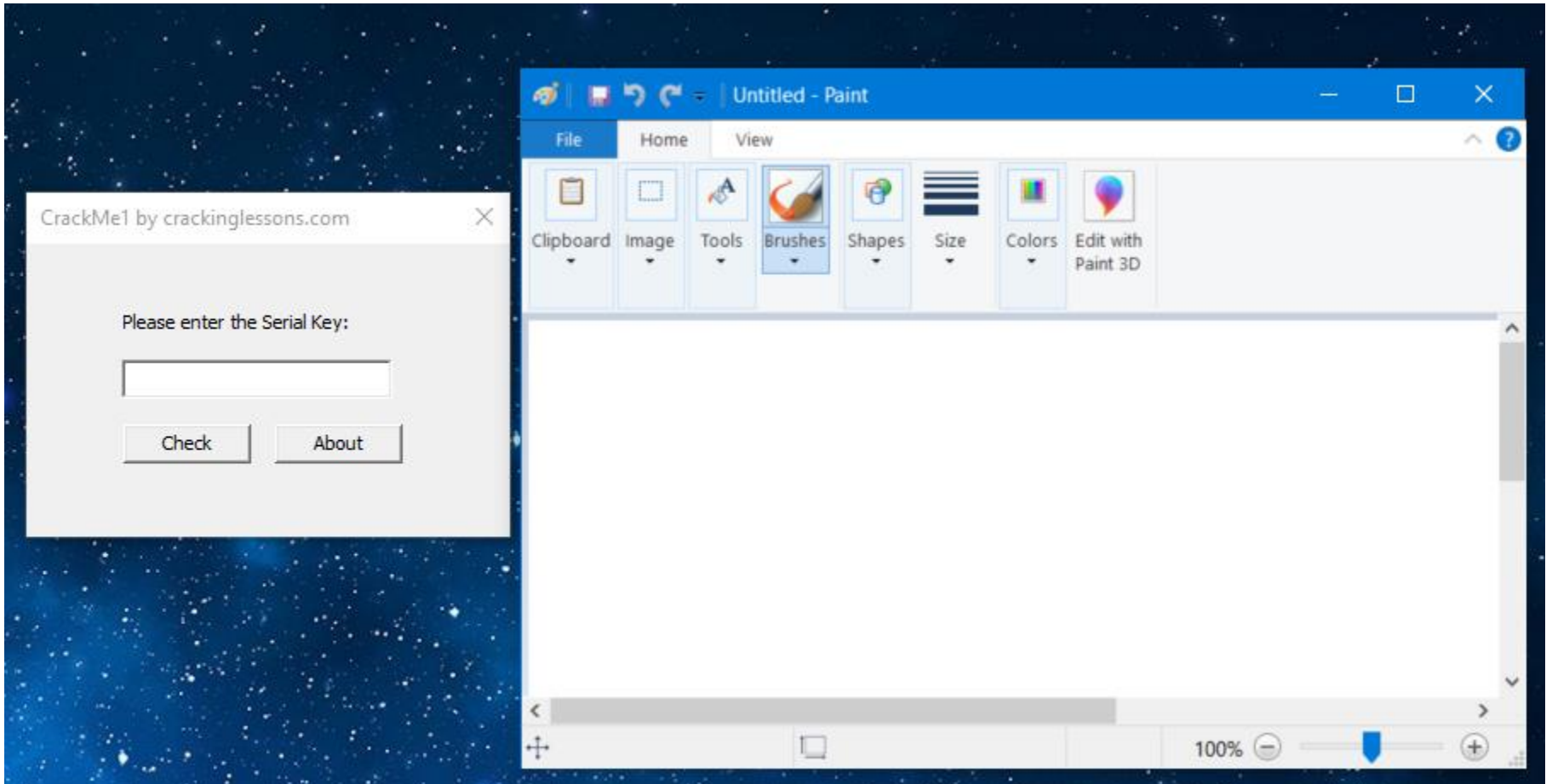
Techniques to Create Trojans

- 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

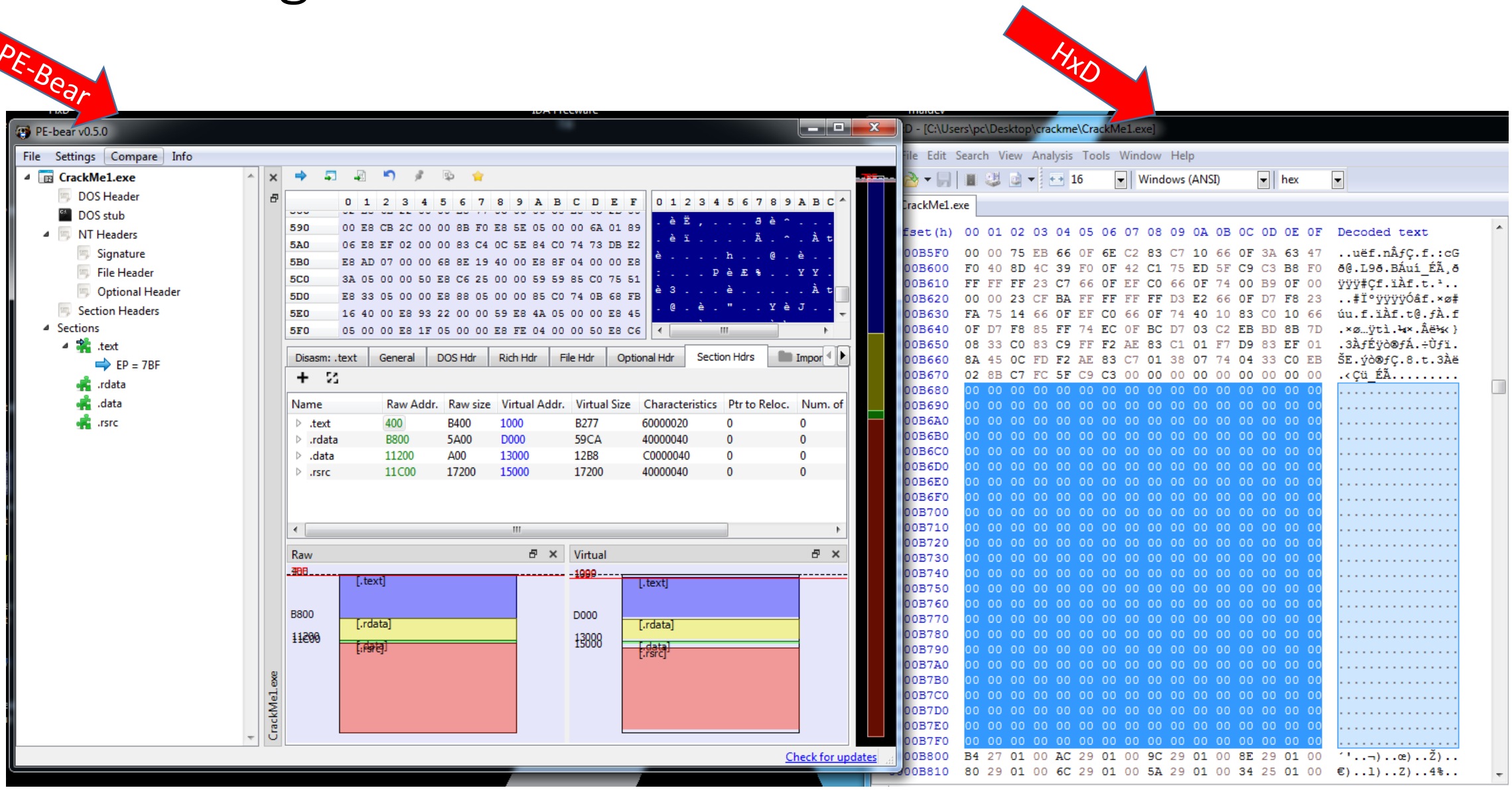
- To trojanize CrackMe1.exe to run mspaint.exe



How to choose a suitable exe to trojanize

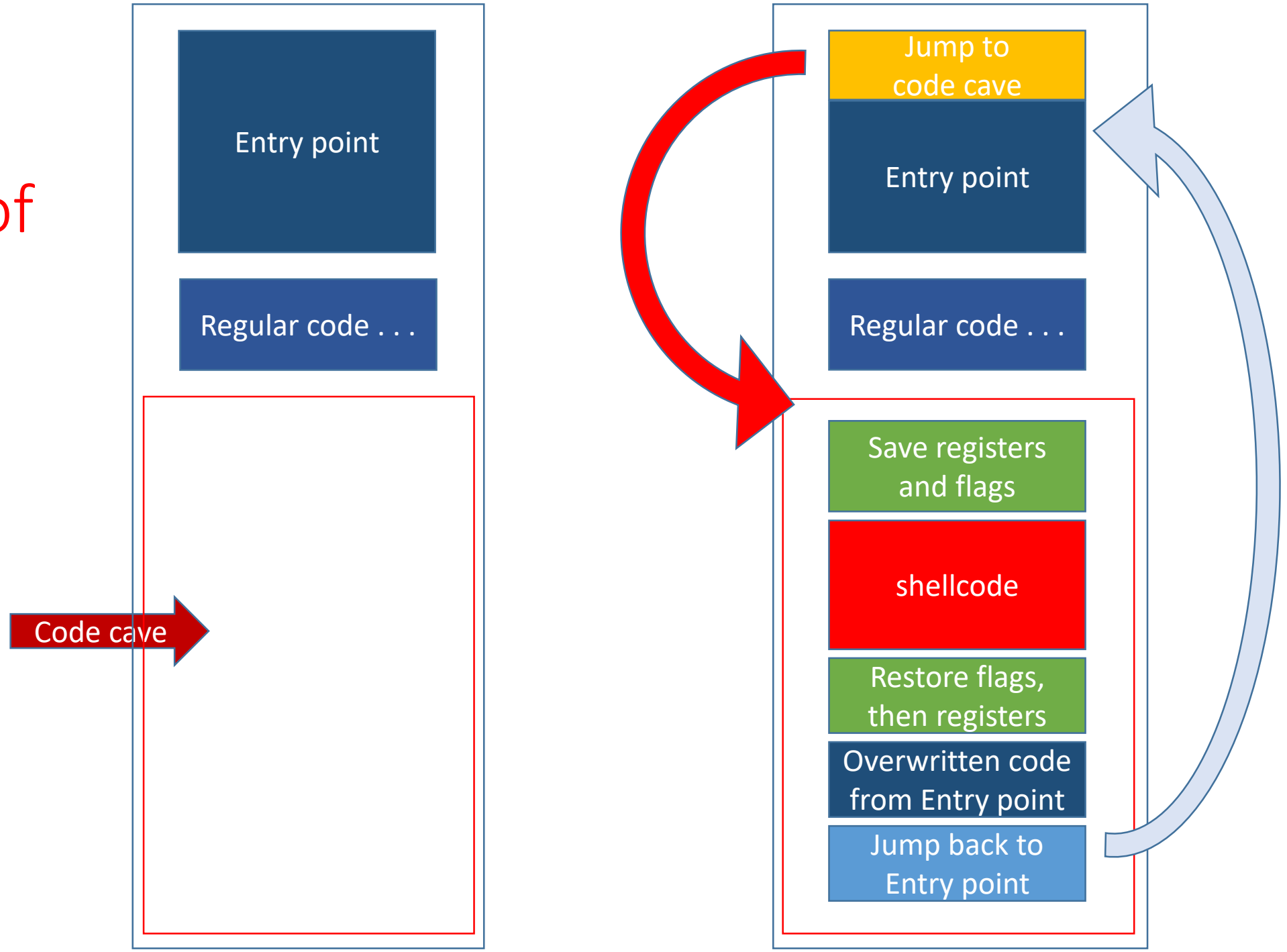
- For code cave, you need a large .TEXT section
- Raw size and Virtual size are different
- Raw size is file size, Virtual size is memory size
- When File is run, OS will map it to virtual memory and give it more memory space for optimization purposes.
- Check raw code cave size – use PE-Bear and HxD 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



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)
6. 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

Thank you