SJSU SAN JOSÉ STATE UNIVERSITY

Lesson 11 – Software Reverse Engineering

Yan Chen

CS166 Fall 2024

Malware Examples

- Malware: applications that designed to do bad things
  - Virus: passive propagation (relies on someone or something)
  - Worm: active propagation (propagates by itself)
  - Trojan horse: unexpected functionality (disguised)
- Introduced real-world examples based on time roughly...
  - 1980s: spread slowly so easy to stop, but increased the awareness of security (e.g., Brain Virus, Morris Worm)
  - 2000s: spread faster but didn't do anything too harmful (more like "showing off") (e.g., Code Red Worm, SQL Slammer)
  - 2010s: attack for profits (e.g., Purelocker Ransomware, Zeus)

Malware Detection

	Signature Detection	Change Detection	Anomaly Detection
If detect...	The “signature” of malware	The changes in legit files	The abnormal usage of the system (changes in usage)
It means...	File with the signature may be malware	The system may be infected (Can't detect the cause though)	
Advantage	Effective & Easy to implement & maintain	Zero false negative	Can detect infection from previously unknown malware
Disadvantage	Cannot detect unknown/advanced malware	High false alarms	Can’t prevent a patient attacker
		Need to combine with signature detection	
Other notes	False alarm possible: normal code may also contain “signature”	Compared by comparing hash value	Need to define “normal” first and it may change

- Advanced malware try to evade signature detection
- Avoid common signatures to evade signature detection

	Encryption	Polymorphic	Metamorphic
Encrypt?	Yes (don't need to use strong cipher though)		NO
Mutates?	"NO"	Decryptor mutates	Yes! (functionalities same)
How to detect	Find signature of decryptor	Emulate until the code is decrypted, then find signature of code	Difficult research problem...

- Or, just spread so fast that no time to react
  - Flash worm: infect entire Internet almost instantly by embedding all vulnerable IP address in the worm

- Software Reverse Engineering (SRE)
  - Also known as Reverse Code Engineering, or “reversing”
  - “Good” usage: understand malware/legacy code
  - “Bad” usage: remove restrictions, find & exploit flaws, etc.
- We assume…
  - Reverse engineer is an attacker
  - Attacker only has exe (no source code, no bytecode)
- Attacker might want to
  - Understand the software
  - Modify (“patch”) the software

- Disassembler: converts exe to assembly (as best it can)
  - Cannot always disassemble 100% correctly
  - In general, can't re-assemble into working executable
  - Gives static results –good overview of program logic
  - User must "mentally execute" program
  - Difficult to jump to specific place in the code
- Debugger: dynamically check assembly code
  - Must step thru code to completely understand it
  - Can set break points
  - Can treat complex code as "black box"

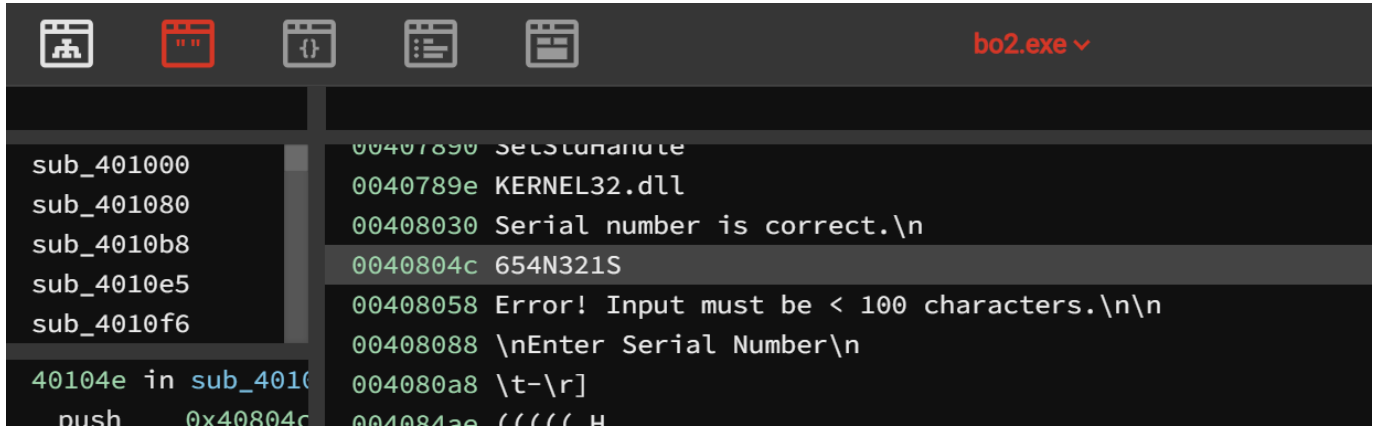
- Disassembler & Debugger usually bundled together
  - Any serious SRE task requires both!
  - E.g., IDA Pro, or online disassembler for easy SRE task, such as Binary Ninja Cloud (register required)
- Hex editor: to view/modify bits of exe ("patch")
  - "Patch" the software by saving the new exe
  - E.g., UltraEdit, HIEW (windows only, with disassembler), or online hex editor for easy SRE task, such as onlinehexeditor
  - Fun fact: hex editor can patch itself (e.g., 010 Editor)
- (Optional) Process monitor: check file system activities

- Working knowledge of target assembly code
  - CS47, CS147
- Experience with the tools
  - IDA Pro – sophisticated and complex
  - OllyDbg – easier to use (not updating anymore)
  - But for assignments/exams, online tools are good enough
- Boundless patience and optimism
- SRE is a tedious, labor-intensive process!

Exploit Buffer Overflow

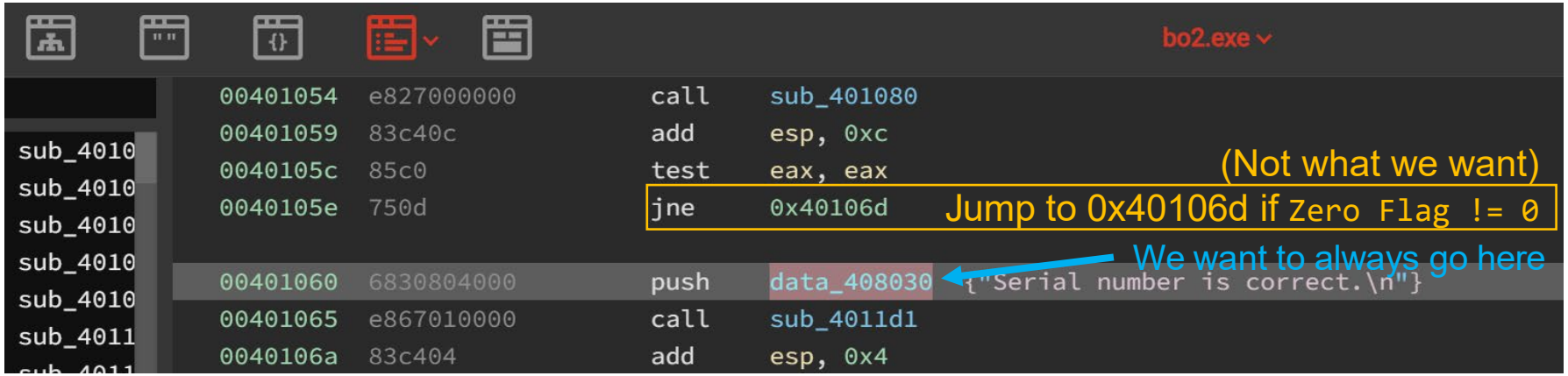
- Can exploit buffer overflow to get serial number...
  - Recall: can "redirect" the return address by overriding it
- Need some trials & errors...
  - To find the length of input that can override return address
- Then disassemble .exe to find the return address of the correct result
  - "Translate" the hex address to characters by ASCII table
  - Input enough character + hex address to override ret.
  - Note that windows X86 processors are "little-endian"
- "Demo" in class

- May be easier to find number directly...
- Disassemble .exe, check if can find the serial number
- In-class demo: use Binary Ninja Cloud
  - Example: bo2.exe (originally from Prof. Stamp's website)
  - Go to Strings, look for a string that looks like a serial number (should be near "Serial Number is correct")



  - As we tried, "654N321S" is the correct serial number

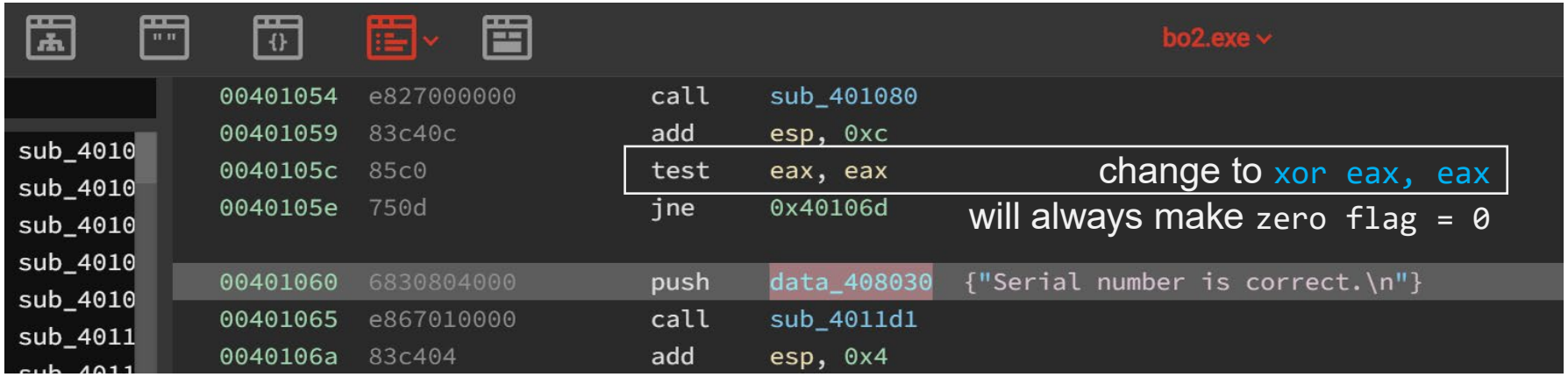
- Or, patch .exe so it will accept all numbers...
- First, find how you can modify the code
  - Use the previous example...
  - We found "Serial number is correct" is at data_408030
  - Check the assembly code...



  - To always jump to data_408030, we can set

    `zero flag` always = 0 after `test eax, eax`

- First, find how you can modify the code (continued)



  - Can change "`test eax, eax`" to "`xor eax, eax`" !
  - Can use online tools to convert assembly to hex...
  - "`xor eax, eax`" is "31 c0" or "33 c0"
- Finally, use hex editor to make the change ("patch")
  - Example: using onlinehexeditor
  - Search for "85 c0" and change it to "31 c0" or "33 c0"

- Impossible to prevent SRE on open system
  - Can only make such attacks more difficult...
- Will cover 4 ways to mitigate SRE
  - Anti-disassembly: to confuse static view of code
  - Anti-debugging: to confuse dynamic view of code
  - Tamper-resistance: code checks itself to detect tampering
  - Code obfuscation: make code more difficult to understand

- Some anti-disassembly methods
  - The idea similar to virus trying to evade signature detection...
  - Encrypted or "packed" object code

    (but need decryptor...same problems we saw before!)
  - False disassembly (put some junk assembly instructions)
  - Self-modifying code
- Some anti-debugging methods
  - Check IsDebuggerPresent()
  - Monitor use of debug registers or/and inserted breakpoints
  - Multithreading – interacting threads may confuse debugger

- Tamper-resistance: check if the code is being changed
  - Goal is to make patching more difficult
  - Code can hash parts of itself
  - If tampering occurs, hash check fails
  - Research has shown, can get good coverage of code with small performance penalty
  - This approach sometimes called "guards"
- But don't want all checks to look similar
  - Then will have "signature"...
  - And easy for attacker to remove checks!

- Code obfuscation: make code hard to understand
  - Opposite of good software engineering
  - Spaghetti code is a good example
  - E.g., opaque predicate: if ((x – y)*(x – y) > (x*x – 2*x*y + y*y)) always false... Attacker wastes time analyzing dead code!
- Code obfuscation for a powerful security technique?
  - Diffie and Hellman's original idea for public key crypto was based on code obfuscation (but didn't work out that way)
  - It has been shown that obfuscation probably cannot provide strong, crypto-like security...

- Other Attacks on Software
  - Salami
  - Linearization
  - Time bomb

Concepts

- Software Reverse Engineering
  - Tools: disassembler, debugger, hex editor, process monitor
  - “Patch”
- SRE to get serial number
  - Exploit buffer overflow
  - Find the correct number directly
  - Patch the .exe to accept all inputs
- SRE mitigation
  - Anti-disassembly
  - Anti-debugging
  - Tamper-resistance
  - Code obfuscation

- For the SRE example, we patched the code by changing `test` instruction to `xor`.
  - Give at least two other ways that Trudy could patch the code so that any serial number will work.
  - Can we change the `jnz` to `jz` to accept all numbers? Why or why not?
- Write some programs, try to reverse engineer it

References

- Stamp, Mark, “Information Security, Principles and Practice, 2nd ed.,” Wiley, New Jersey, USA, 2011