Software Reverse Engineering
Software Cracking

Sometimes, the best way to advance *is in reverse*

Dr.Talal Alkharobi

What is Reverse Engineering?

- The process of extracting the knowledge or design blueprints from anything man-made.
- It is different than conventional scientific research (e.g, coming up with blueprints of the atom)
  - With reverse engineering, the artifact being investigated is man-made
  - With Scientific research, the artifact is a natural phenomenon

3

Software Reverse Engineering

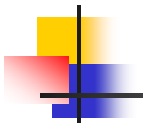
- Opening up a program's "box" and looking inside
- No screwdrivers needed, but integrates several arts of
  - Code breaking
  - Puzzle solving
  - Programming
  - Logical analysis

4

Why Reverse Engineering?

- Application development
  - Why reinvent the wheel
  - Undocumented API
  - Interoperability with proprietary software
  - Software quality and robustness
  - Taking measurements
  - Analysis to draw proper conclusions

5



Why Reverse Engineering?

- Security-Related
  - Malicious Software
    - Black-Hat Hackers: To find vulnerabilities and exploit them
    - White-Hat Hackers: To identify, dissect and analyze malware
  - Reversing ciphers
  - Defeating copy right protection
  - Must-do when it comes to propriety software
    - Really unknown what the code does and how secure

6

Is reversing legal?

- Jury is still out
- Bottom line: what is "*reverse engineering*" used for?

Approaches to reverse engineering

7

- White Box Analysis
  - Analyzing and understanding source code
  - Code coverage – *walking down dark allies*
- Black Box Analysis: Analyzing a running program by probing it with various inputs
- Gray Box Analysis

8

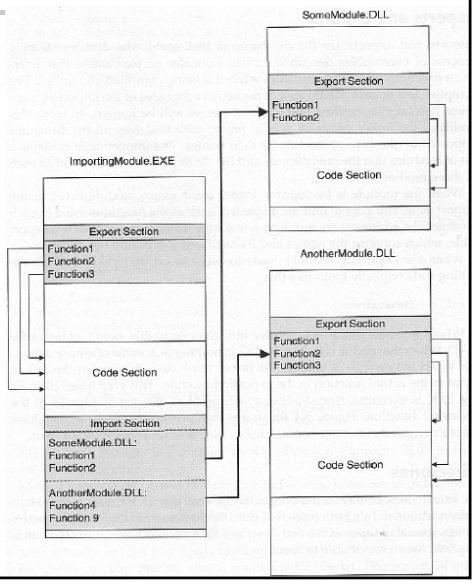
Program Structure

- Static libraries
  - Third party library external to program
  - Added to program while being built
  - Become integral part of the final executable
- Dynamic libraries
  - Idea is to utilize memory
  - Multiple processes can use the same DLL
  - Remains separate from the executable
  - Update to DLL requires no update to program

9

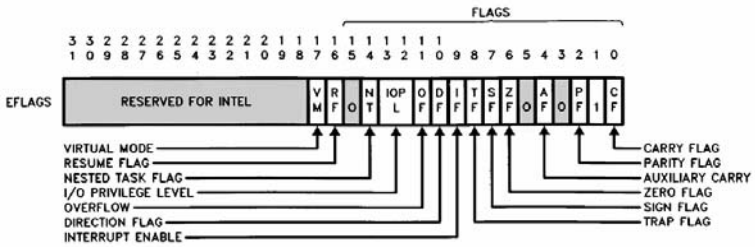
The .exe format and DLLs

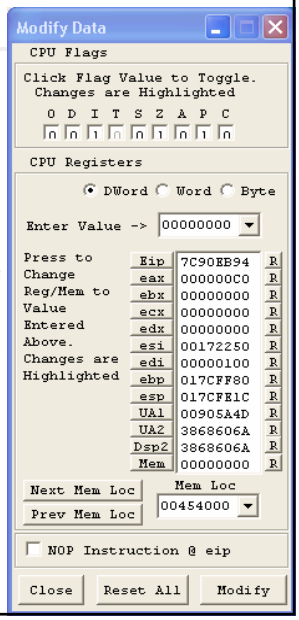
- Before loading DLLs, addresses of functions in DLL are pointing to dummy addresses in an import table
- When process is loaded, the OS loader loads every module listed in the imported table, and resolves the addresses of each of the functions listed in each modules. The addresses are found in the exported table of the module.



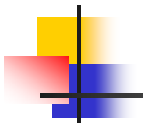
10

Important Flags





11



Reversing Tools

- Disassemblers and low-level debuggers
  - IDA Pro
  - ILDasm
  - OllyDbg
  - Win32Dasm
  - WinDbg
  - Numega SoftICE

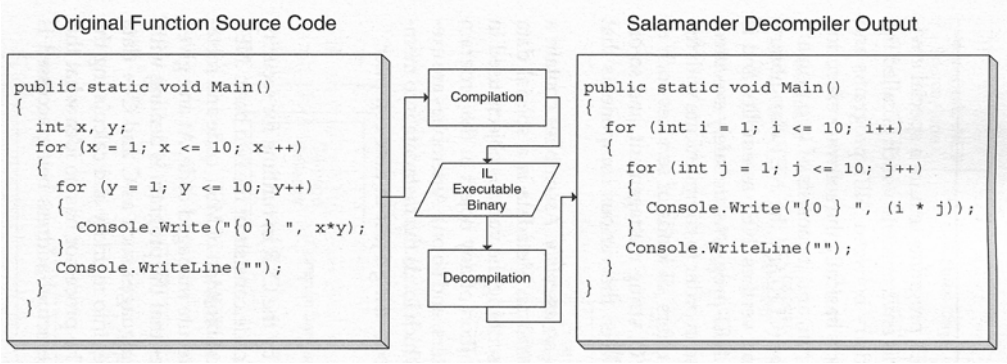
12

Reversing Tools

- Decompilers
  - *For Java and C#, the dream has come true*
- Hex Editors
  - Hex Workshop
  - Hiew
  - WinHex

13

Decompiling C# function



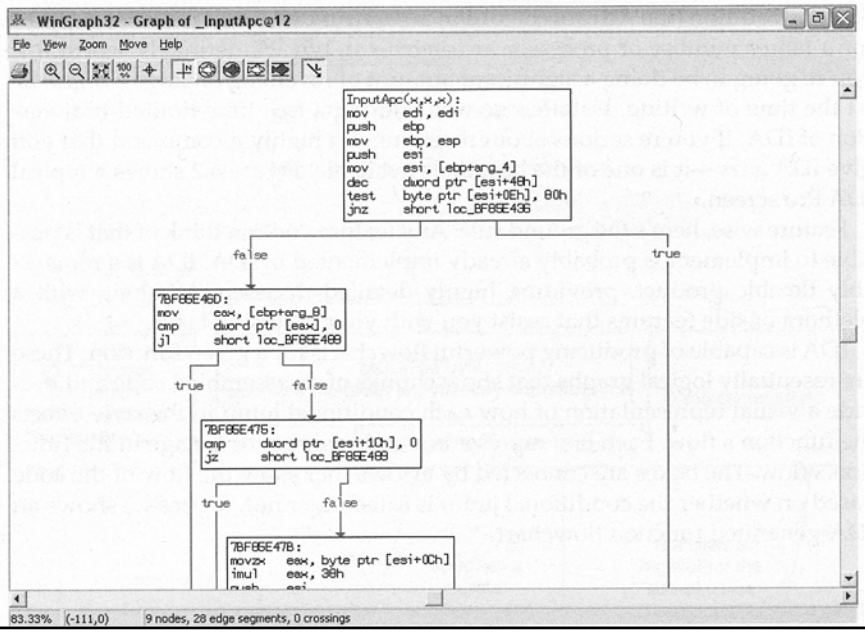
14

Reversing Tools

- System monitors:
  - FileMon,
  - TCPView,
  - TDIMon,
  - RegMon,
  - PortMon,
  - WinObj,
  - Process Exporer

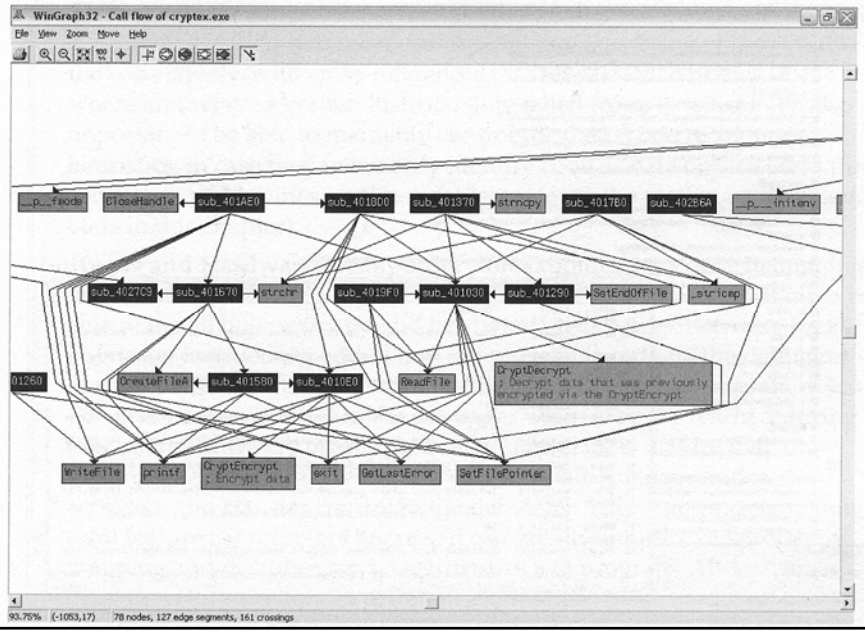
IDA-generated function flowchart

15

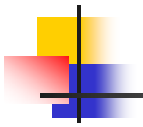


IDA-generated function interactions at high level

16



17



Copyright Protection

- Open Architecture
  - CPU can not check “authorized” or “legal” code
  - Only execution mode and process memory protection by OS
  - When program runs, its data and code has to be *exposed* or decrypted
  - Decryption key or decrypted data are impossible to hide – It is the CPU that does the decryption operation
- Make no mistake: It is virtually impossible to create a totally uncrackable copy protection scheme
- One crack is not the issue, “*class breaks*” is

18

Design Considerations

- Resistance to attack
- End-user transparency

19

Types of Protection

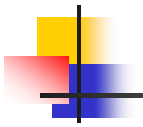
- Media-Based Protections: Deliberately writing bad sectors on floppy
- Watermarking: Check for watermark on media before running
- Serial Numbers: Program has “secret validation algorithm” to check for good serial numbers
- Challenge Response
- Online Activations
- Hardware-Based Protections: Program runs or keeps running if a dongle is attached
- Checksum: Check for valid checksum before or during running

20

checksum

- Most BIOSes and Embedded Systems calculate a "checksum" of the software before executing it (or at least, before executing anything past the "checksum" code).
- Then if the calculated checksum fails to match the "correct" checksum, the BIOS refuses to execute the "corrupted" software.
- The checksum assured the programmer that the program really was all there.

21



Types of Protection

- Software as a Service (SaaS)
  - Software application delivery model where a software vendor develops a web-native software application and hosts and operates (either independently or through a third-party) the application for use by its customers over the Internet.
  - Customers pay not for owning the software but for using it.
  - They use it through an API accessible over the Web and often written using Web Services or REST.
  - The term SaaS has become the industry preferred term, generally replacing the earlier terms Application Service Provider (ASP), On-Demand and "Utility computing".
  - Highly secure but …. Vendor is in control

22

Types of Protection

- Advanced Protection Concepts -- Crypto-Processors
  - Violates Open Architecture
  - Originated by Robert M. Best in his patent, “Executing Enciphered Programs.”
  - Decrypt code then Execute
  - Each Crypto-microprocessor has
    - Two pair of keys
    - A serial number

23

Crypto Processors execution

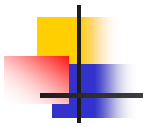
- Software is shipped by the vendor with binary executable encrypted
- Microprocessor decrypts with private key
- Machine code is placed in a special memory not software-accessable
  - *Similar to macrocode for each machine instruction*
  - *Only accessed by the process itself.*
  - *Any other access by OS or any software is a violation.*
- Code is executed from this (*theoretically*) inaccessible memory

24

Antireversing Techniques

- It is *never* possible to entirely prevent reversing
- What *is* possible is to hinder and obstruct reversers by wearing them out and making it so slow and painful.
- *Drown the reverser into a sea of irrelevant information….. they give up*
- *Anti-Reversing techniques are techniques for a programmer to foil reverse-engineering attempts on their software.*

25



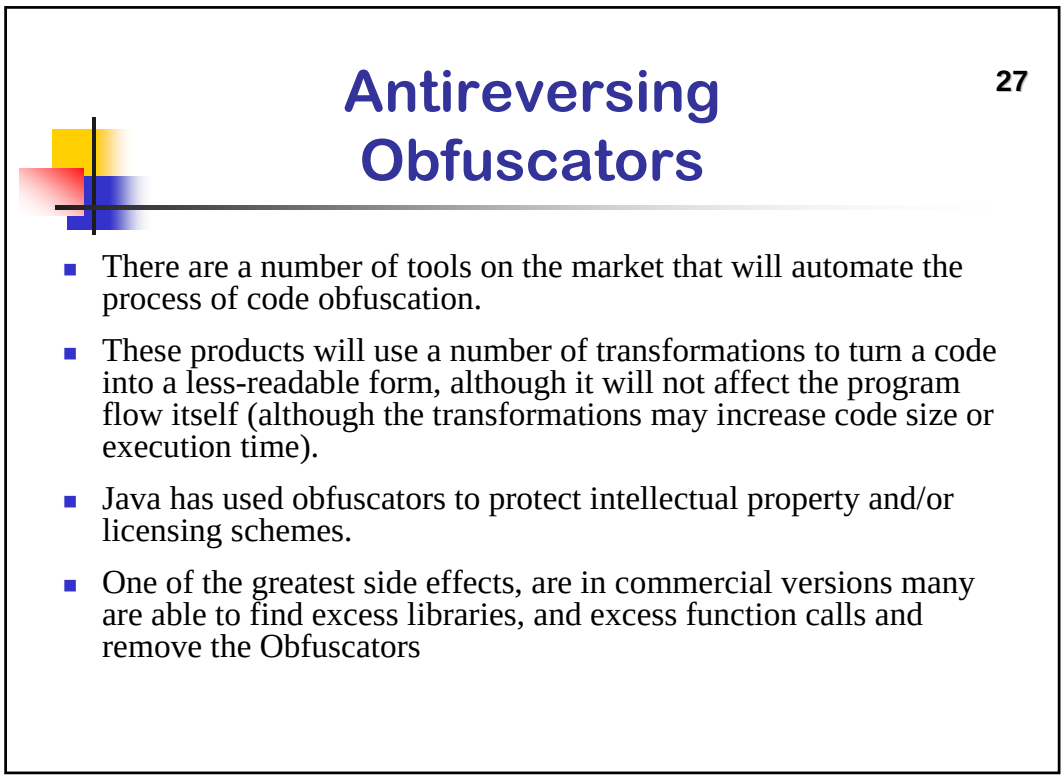
Anti-Reversing techniques

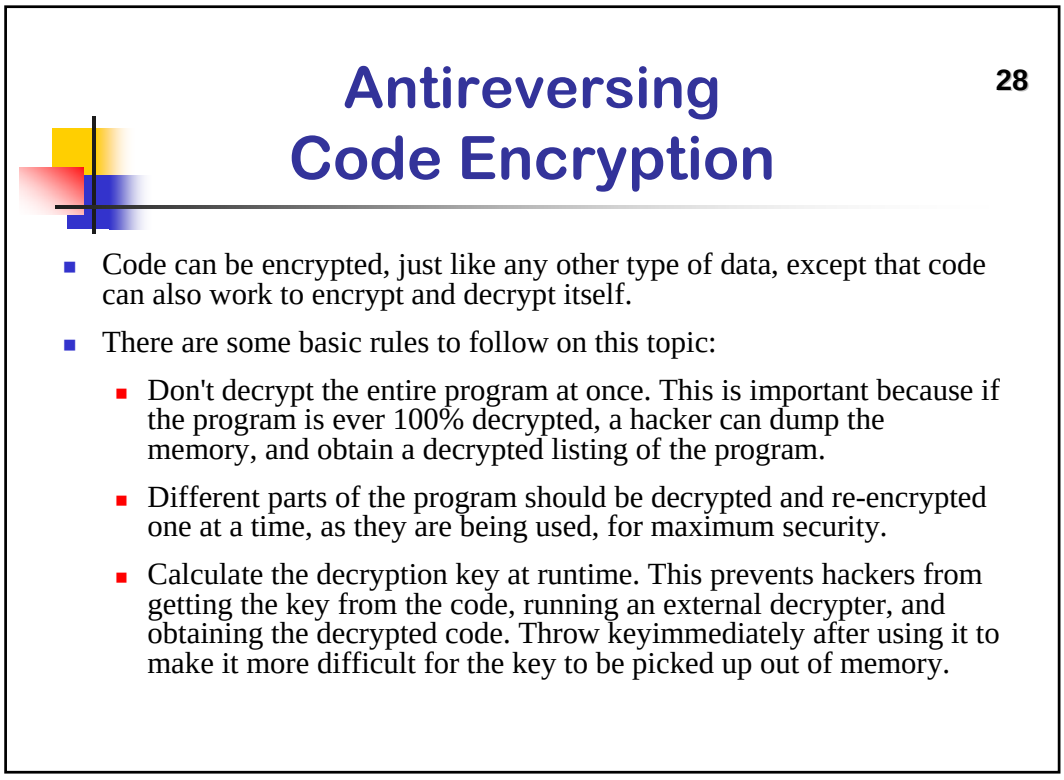
- Obfuscators
- Code Transformations
- Opaque Predicates
- Code Encryption-Decryption
- Code Hashing
- Detecting Debuggers
- Killing Debuggers

26

Antireversing Obfuscators

- Eliminate Symbolic
  - Non-byte-code: Imported, exported and internal Function names, string names.
  - Byte-code (Java and .NET)
    - Intermediate code of a compiler
    - Uses internal cross-references than addresses
    - More: class names, class member names, names of instantiated global objects
    - Use *inlining* & addresses for function names


Antireversing
Obfuscators
27
There are a number of tools on the market that will automate the process of code obfuscation.
These products will use a number of transformations to turn a code into a less-readable form, although it will not affect the program flow itself (although the transformations may increase code size or execution time).
Java has used obfuscators to protect intellectual property and/or licensing schemes.
One of the greatest side effects, are in commercial versions many are able to find excess libraries, and excess function calls and remove the Obfuscators


Antireversing
Code Encryption
28
Code can be encrypted, just like any other type of data, except that code can also work to encrypt and decrypt itself.
There are some basic rules to follow on this topic:
Don't decrypt the entire program at once. This is important because if the program is ever 100% decrypted, a hacker can dump the memory, and obtain a decrypted listing of the program.
Different parts of the program should be decrypted and re-encrypted one at a time, as they are being used, for maximum security.
Calculate the decryption key at runtime. This prevents hackers from getting the key from the code, running an external decrypter, and obtaining the decrypted code. Throw keyimmediately after using it to make it more difficult for the key to be picked up out of memory.

Antireversing Code Encryption

29

- Use Large Keys. 32bit keys can be easily decrypted using a moderately fast computer, a quick algorithm, and a brute-force approach. 64bit keys or longer should be used to ensure security.
- A skilled hacker will be able to get your decryption key anyway, by setting a hardware breakpoint at the spot where you finish calculating

Anti-reversing Code Transformations

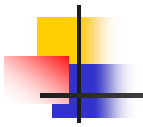
30

```
FA()
{  FuncAPart1();
    FuncAPart2();
    FuncAPart3();}
FB()
{ FuncBPart1();
   FuncBPart2();
   FuncBPart3();}
main()
{  FA();
  FB();}
```

```
main: jmp FAP1
FBP3: call FuncBPart3
      jmp end
FBP1: call FuncBPart1
      jmp FBP2
FAP2: call FuncAPart2
      jmp FAP3
FBP2: call FuncBPart2
      jmp FBP3
FAP1: call FuncAPart1
      jmp FAP2
FAP3: call FuncAPart3
      jmp FBP1
end:
```

31

Anti-reversing Code Transformations



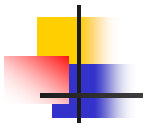
- Both previous codes does the same thing
- The code in the right is much harder to read, although it perfectly preserves the program flow of the original code
- This code is much harder for a human to read, although it isn't hard at all for an automated debugging tool (such as IDA Pro) to read.

32

Opaque Predicate

- A line (or lines) of code in a program that don't do anything, but that look like they do something.
- This is opposed to a transparent predicate that doesn't do anything and looks useless.
- A program filled with opaque predicates will take more time to decipher, because the reverser will take more time reading through useless, distraction code.

33



Code Hashing

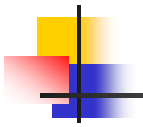
- If a reverse engineer experiments with modifying a few bytes of the software, then these checksums will fail, and the software will refuse to run.
- But many checksums are simple enough that it's easy to modify one or two non-critical bytes to force the sum to equal the "correct" checksum. Hashes, though, are a different story.
- Let's say we use a known hashing algorithm to find the given hash value of our program's code.
- Now, we can have the program recalculate the hash value at run-time, and compare this value to the known value.

34

Code Hashing

- If the two numbers don't match up, then we know that a hacker has patched the code.
- At this point, we can terminate the program with a warning: "Don't hack this program!!!".
- A good example of code hashing techniques is the .NET platform.
- .NET allows a programmer to sign a hash value or a "signature" to a compiled .NET module.
- If the code has been edited or patched in any way, the program will not run.

35



Detecting Debugger

- TF set in EFLAGS
- Check handlers in IDT
- Enable TF and see if exception is swallowed by debugger
- Code checksum
  - Slowdown runtime
  - Use only for sensitive functions
- Use a special thread to see if program was stopped for long time

36

Detecting Debuggers

- IsDebuggerPresent API:
  - Win32 API contains a function "IsDebuggerPresent", which will return a boolean true if the program is being debugged.
  - The following code shows a general usage of this function:
    - `if(IsDebuggerPresent())`
    - `{ TerminateProcess(GetCurrentProcess(), 1);}`
  - It is easy to spot uses of the IsDebuggerPresent() function in the disassembled code
  - A skilled reverser will simply patch the code to remove this line.

Detecting Debuggers Timeouts

37

- Debuggers can put break points in the code, and can therefore stop program execution.
- A program can detect this, by monitoring the system clock.
- If too much time has elapsed between instructions, it can be determined that the program is being stopped and analyzed (although this is not always the case).
- If a program is taking too much time, the program can terminate.

Detecting Debuggers Detecting SoftICE

38

- SoftICE is a local kernel debugger, and as such, it can't be detected as easily as a user-mode debugger can be.
- The IsDebuggerPresent API function will not detect the presence of SoftICE.
- To detect SoftICE, number of techniques can be used:
  - Search for the SoftICE install directory.
  - If SoftICE is installed, the user is probably a hacker/reverser.
  - Detect the presence of int 1.
  - SoftICE uses interrupt 1 to debug, so if interrupt 1 is installed, SoftICE is running.

39

Detecting Debuggers
Detecting OllyDbg

- OllyDbg is a popular 32-bit usermode debugger.
- Unfortunately, the last few releases, including the latest version (v1.10) contain a vulnerability in the handling of the Win32 API function OutputDebugString().
- A programmer trying to prevent his program from being debugged by OllyDbg could exploit this vulnerability in order to make the debugger crash.
- The author has never released a fix, however there are unofficial versions and plugins available to protect OllyDbg from being exploited using this vulnerability

40

Keygenning

- Definition: The process of creating programs that mimic the key-generation algorithm for everyone to use
- Software can be blacklisted if it always works with same pair of (serial number or username.)
- Key depends on pair of username plus computer-specific info (disk driver serial number)
- Ripping Keygen Algorithms: Locate/copy/paste/modify

41

Reversing 101

- Books
  - Elad Eilam, “Reversing: Secretes of Reverse Engineering,” Wiley, 2005
  - Greg Hoglund, “Exploiting Software: How to Break Code,” Addison-Wesley, 2004
  - Kris Kaspersky, “Hacker Disassebmling Uncovered,” A-List Publishing, 2003

42

Reversing 101

- Web sites
- Learn to crack
  - http://www.learn2crack.com/
  - http://woodmann.net/krobar/
- Crack Find
  - http://www.crackfind.com/
  - http://www.cracksearchengine.net/
  - http://astalavista.box.sk
  - http://ttdown.com/