Malware Reverse Engineering

Tools:
IDA Pro, OllyDbg, PEiD etc.
What is reverse engineering?

- Reverse engineering is the process of analyzing a subject system to create representations of the system at a higher level of abstraction (Chikofsky, January 1990)
- Going backwards through the development cycle
  - Source code deduction
  - Architecture discovery
RE applications

• White hats:
  – Clean Room Design (avoid copyright infringement)
    - React OS, Wine HQ, IBM BIOS
  – Vulnerability discovery (trigger, pen testing)
  – Malware analysis

• Black hats:
  – Cracking, hacking
  – Vulnerability discovery (exploit)
  – Malware re-engineering (shell code reuse)
Reverse engineering approaches

• Behavioral analysis
  – Execute malware in isolated environment and record its activity footprint
    • Registry (RegShot)
    • Files (DiskMon, FileMon)
    • Network (Wireshark)
    • API and system services/calls (Process Explorer, EasyHook, custom filter drivers)

• Code analysis
  – Inspect Import/Exports (PEiD)
  – Identify packer and unpack (PEiD, various unpackers)
  – Disassemble, analyze call/data flow graph
  – Debug, decrypt and reveal actual values
  – Patch binary to traverse most of code branches
Basic skill set of RE

Tasks
- Unpacking
- Disassembling
- Binary Debugging/patching
- Dynamic (behavior) analysis

Tools
- PEiD, OllyDbg,
- OllyDbg, SoftICE, WinDbg
- IDA Pro, PE Explorer, OllyDbg
- PaiMei, Wireshark, Process Explorer
Malware analysis questions

• What type? (worm, virus, keylogger, dropper etc.)
• How does it propagate? (remote code injection, overwrite, network vector, etc.)
• What does it do (payload)? (registry abuse, dropper, credentials theft, etc.)
• What are the trigger conditions? (time bomb date, service presence, etc.)
Malware reverse engineering scheme

1. Create isolated experimental environment (host only VM)
2. Submit malware to existing sandboxes (Anubis, ThreatExpert)
   • Inspect its high level behavior
   • Watch file droppers and created processes
3. Examine its imports, exports and strings
   • Based on import API, guess malware type (e.g. SetWindowsHook)
4. Identify packer and unpack (manual or auto, PEiD)
5. Disassemble/decompile malware
   • Trace API usage (context, constant attributes)
   • Deduce variables, create structures, name them properly (symantics)
   • Deduce function semantics (what does it do, approx.)
   • Trace sensitive data path
6. Debug binary
   • Trigger conditions
   • Resolve implicit jumps (control flow)
   • Reveal actual values variables (dynamic API attributes)
7. Patch binary (optional)
   • Execute malicious payload (addresses limitations of step 2)
Assembly language
Disassembler: IDA Pro
Debugger: OllyDbg
Hands-On Training

```
.push    offset PathName  ; lpPathName
.call   SetCurrentDirectoryA
.push    offset phkResult ; phkResult
.push    1                  ; samDesired
.push    0                  ; ulOptions
.push    offset SubKey     ; "~+3:+|9|.:9|+|3::|72:+|9|9|7:+|9|"
.push    8000002h           ; hKey
.call   RegOpenKeyExA
.mov     cbData, 12Ch
.push    offset cbData     ; lpcbData
.push    offset Data       ; lpData
.push    offset Type       ; lpType
.push    0                 ; lpReserved
.push    offset ValueName  ; "\t\t|\t7|6|\"
.push    phkResult         ; hKey
.call   RegQueryValueExA
.push    phkResult
.call   _4E
.push    offset a_Li492_29 ; "\(\|\|4\|9|2|2<9|"
.push    offset Data       ; lpString1
.call   1strcatA
.push    0                 ; hTemplateFile
.push    80h               ; dwFlagsAndAttributes
.push    3                 ; dwCreationDisposition
.push    0                 ; lpSecurityAttributes
.push    1                 ; dwShareMode
.push    80000008h         ; dwDesiredAccess
.push    offset FileName  ; "\-7|&|1|.;||7"
.call   CreateFileName
.push    0                 ; nNumberOfBytesToLockHigh
```
## IDA Shortcuts

### DataRefence
Interactive Disassembler (IDA) Pro
Quick Reference Sheet
(http://www.datarefence.com)

### Navigation
- Jump to operand: Enter
- Jump in new window: Alt+Enter
- Jump to previous position: Alt+Esc
- Jump to next position: Ctrl+Enter
- Jump by name: Ctrl+L
- Jump to function: Ctrl+F
- Jump to segment: Ctrl+S
- Jump to segment register: Ctrl+G
- Jump to problem: Ctrl+Q
- Jump to cross reference: Ctrl+X
- Jump to sel to operand: X
- Jump to entry point: Ctrl+E
- Mark position: Alt+M
- Jump to marked position: Ctrl+M

### Search
- Next code: Alt+C
- Next data: Ctrl+D
- Next explored: Ctrl+A
- Next unexplored: Ctrl+U
- Immediate value: Alt+I
- Next immediate value: Ctrl+1
- Test: Alt+T
- Next test: Ctrl+T
- Sequence of bytes: Alt+B
- Next sequence of bytes: Ctrl+B
- Not function: Alt+U
- Next void: Ctrl+V
- Error operand: Ctrl+F

### Graphing
- Flow chart: F12
- Function calls: Ctrl+F12

### Open Subviews
- Names: Shift+F4
- Functions: Shift+F3
- Strings: Shift+F12
- Segments: Shift+F7
- Segment registers: Shift+F8
- Signatures: Shift+F5
- Type libraries: Shift+F11
- Structures: Shift+F9
- Enumerations: Shift+F10

### Data Format Options
- ASCII string style: Alt+A
- Setup data types: Alt+D

### File Operations
- Pause & loader file: Ctrl+P
- Create DSM file: Alt+F9
- Save database: Ctrl+W

### Debugger
- Start process: F9
- Terminate process: Ctrl+F2
- Step into: F7
- Step over: F6
- Run until return: Ctrl+F7
- Run to cursor: F4

### Breakpoints
- Breakpoint list: Ctrl+Alt+B

### Miscellaneous
- Calculator: Ctrl+Z
- Cycle through open views: Ctrl+Tab
- Select tab: Alt+[...N]
- Close current view: Ctrl+F4
- Exit: Alt+X
- IDC Command: Shift+F2

### Edit (Data Types – etc)
- Copy: Ctrl+Ins
- Begin selection: Alt+L
- Manual instruction: Alt+F2
- Code: C
- Data: D
- Struct variable: Alt+Q
- ASCII string: Alt+A
- Array: Ctrl+T
- Undeline: Ctrl+U
- Rename: Ctrl+N

### Operands
- Offset (data segment): Ctrl+O
- Offset (current segment): Ctrl+P
- Offset by (any segment): Alt+R
- Offset (user-defined): Ctrl+R
- Offset (struct): T
- Number (default): #
- Hexadecimal: Ctrl+Q
- Decimal: Ctrl+H
- Binary: Ctrl+B
- Character: Ctrl+R
- Segments: Ctrl+S
- Enum member: Ctrl+M
- Stack variable: Ctrl+K
- Change sign: Ctrl+R
- Bitwise negate: ~
- Manual: Ctrl+F1
- Comments: Enter
- Enter repeatable comment: 
- Insert anterior lines: Ins
- Insert posterior lines: Shift+Ins
- Insert predefined comment: Shift+F1
- Segments: Ctrl+Z
- Change segment register value: Alt+G
- Structs: Ctrl+T
- Force zero offset field: Ctrl+Z
- Select union member: Alt+Y

### Functions
- Create function: Ctrl+P
- Edit function: Alt+P
- Set function end: Ctrl+K
- Change stack pointer: Ctrl+K
- Rename register: Ctrl+V
- Set function type: Y