

# REVERSE ENGINEERING CHEAT SHEET

By Lenny Zeltzer

This cheat sheet of shortcuts and tips for reverse-engineering malware. It covers the general malware analysis process, as well as useful tips for OllyDbg, IDA Pro, and other tools. Feel free to customize it to your own needs. My [reverse-engineering malware course](#) explores these, and other useful techniques.

## General Approach

1. Set up a controlled, isolated laboratory in which to examine the malware specimen.
2. Perform behavioral analysis to examine the specimen's interactions with its environment.
3. Perform static code analysis to further understand the specimen's inner-workings.
4. Perform dynamic code analysis to understand the more difficult aspects of the code.
5. If necessary, unpack the specimen.
6. Repeat steps 2, 3, and 4 (order may vary) until sufficient analysis objectives are met.
7. Document findings and clean-up the laboratory for future analysis.

## Behavioral Analysis

Be ready to revert to good state via [dd](#), [VMware](#) snapshots, [CoreRestore](#), [Ghost](#), [SteadyState](#), etc.

Monitor local ([Process Monitor](#), [Process Explorer](#)) and network ([Wireshark](#), [tcpdump](#)) interactions.

Detect major local changes ([RegShot](#), [Autoruns](#)).

Redirect network traffic (hosts file, DNS, [Honeyd](#)).

Activate services (IRC, HTTP, SMTP, etc.) as needed to evoke new behavior from the specimen.

## IDA Pro for Static Code Analysis

Text search	Alt+T
-----	-----
Show strings window	Shift+F12
-----	-----
Show operand as hex value	Q
-----	-----
Insert comment	:
-----	-----
Follow jump or call in view	Enter
-----	-----
Return to previous view	Esc
-----	-----
Go to next view	Ctrl+Enter
-----	-----
Show names window	Shift+F4
-----	-----
Display function's flow chart	F12
-----	-----
Display graph of function calls	Ctrl+F12
-----	-----
Go to program's entry point	Ctrl+E
-----	-----

# REVERSE ENGINEERING CHEAT SHEET

By Lenny Zeltzer

Go to specific address	G
Show listing of names	Ctrl+L
Display listing of segments	Ctrl+S
Show cross-references to selected function	Select function name » Ctrl+X
Show stack of current function	Ctrl+K

## OllyDbg for Dynamic Code Analysis

Step into instruction	F7
Step over instruction	F8
Execute till next breakpoint	F9
Execute till next return	Ctrl+F9
Show previous executed instruction	-
Show next executed instruction	+
Return to previous view	*
Show memory map	Alt+M
Follow expression in view	Ctrl+G
Insert comment	;
Follow jump or call in view	Enter
Show listing of names	Ctrl+N
New binary search	Ctrl+G
Next binary search result	Ctrl+L
Show listing of software breakpoints	Alt+B
Assemble instruction in place of selected one	Select instruction » Spacebar
Edit data in memory or instruction opcode	Select data or instruction » Ctrl+E
Show SEH chain	View » SEH chain
Show patches	Ctrl+P

## **Bypassing Malware Defenses**

To try unpacking quickly, infect the system and dump from memory via LordPE or OllyDump.

For more surgical unpacking, locate the Original Entry Point (OEP) after the unpacker executes.

If cannot unpack cleanly, examine the packed specimen via dynamic code analysis while it runs.

When unpacking in OllyDbg, try SFX (byte-wise) and OllyDump's "Find OEP by Section Hop".

# REVERSE ENGINEERING CHEAT SHEET

By Lenny Zeltzer

Conceal OllyDbg via HideOD and OllyAdvanced.

A JMP or CALL to EAX may indicate the OEP, possibly preceded by POPA or POPAD.

Look out for tricky jumps via SEH, RET, CALL, etc.

If the packer uses SEH, anticipate OEP by tracking stack areas used to store the packers' handlers.

Decode protected data by examining results of the decoding function via dynamic code analysis.

Correct PE header problems with XPELister, LordPE, ImpREC, PEiD, etc.

To get closer to OEP, try breaking on unpacker's calls to LoadLibraryA or GetProcAddress.

## Common x86 Registers and Uses

EAX	Addition, multiplication, function results
ECX	Counter
EBP	Base for referencing function arguments (EBP+value) and local variables (EBP-value)
ESP	Points to the current "top" of the stack; changes via PUSH, POP, and others
EIP	Points to the next instruction
EFLAGS	Contains flags that store outcomes of computations (e.g., Zero and Carry flags)