SYMBOLS
$  in run script commands, 242
$$ command (comments), 226–227
$< . / $#< commands (script files), 240–241
* (asterisk) for creating comments, 227
?? commands (expressions), 190–191
@@ prefix (expressions), 191
[ ] (square brackets) to indicate memory access (x86), 5
{} (curly braces)
in block commands, 228
in conditional statements, 229
| command (debugging), 196
0x403F0 routine, 173–175
0x8 pattern, 71
( ) parentheses in conditional statements, 229
32-bit EFLAGS register, 3
32-bit general-purpose registers (GPRs), 2
/3GB switch, 89
64-bit GPRs, 3
64-bit registers, 36

A
ABI (Application Binary Interface) (ARM), 72
abstract interpretation, 290, 294–295
abstract semantics, 290–291
Acorn RISC Machine, 39
ADD instruction, 14
Address parameter (memory), 203–204
address translation, 26–27
ad-hoc execution. See asynchronous and ad-hoc execution
ADR instruction, 52
AL (always execute) condition, 70–71
aliases (DbgEng)
@call script file alias, 244–249
automatic, 219, 225–226
fixed-name, 225
user-named, 219–224
AM64 Architecture Programmer’s Manual, 28–34
AND operation, 75
APCs (asynchronous procedure calls)
APC LEVEL (1) IRQL, 105
basics, 131–135
implementing thread suspension with, 134
user-mode, 131–132
APIs (Application Program Interfaces) accessing DbgEng, 258–261
API hooking module (VxStripper), 309
DeviceIoControl, 151
KeInitializeApc, 132–133
MmGetPhysicalAddress, 181
PsCreateSystemThread, 128
Win32 API functions, 32
WinDbg extension (SDK), 261–262
Application Binary Interface (ABI), 15
application program status register (APSR), 61
approximation partial ordering, 290–291
arbitrary context (kernels), 109–110
arguments, passing to script files, 242–244
arithmetic operations
arithmetic substitution via identities (obfuscation), 275
ARM, 60–61
x86 instruction set, 11–13
ARM architecture
arithmetic operations, 60–61
ARM state, 41–42
basics, 40–42
branching and conditional execution, 61–66
data types and registers, 43–44
decompiling unknown function walk-through, 71–77
functions and function invocation, 57–60
incrementing values in memory, 4
instructions, 46–47, 70–71
JIT (just-in-time) code, 67
loading/storing data. See loading/storing data (ARM)
overview, 39–40
SMC (self-modifying code), 67
synchronization primitives, 67–68
system services and mechanisms, 68–70
system-level controls and settings, 45
testing ARM knowledge, 77–78
as command (aliases), 220–222
asynchronous and ad-hoc execution
asynchronous procedure calls (APCs), 131–134
completion routines, 143–144
deferred procedure calls (DPCs), 135–139
process and thread callbacks, 142
system threads, 128–129
timers, 140–141
work items, 129–131
asynchronous procedure calls (APCs), 131–134
AT&T syntax for x86 assembly code, 4–5
automatic aliases (DbgEng), 219, 225–226

B
B (Branch) instruction, 58
ba command (hardware breakpoints), 210
backtrace_binding method, 298
backtracking/slicing (Metasm), 297–302
barrel shifter feature (ARM), 42–43, 60
base frame pointer, 16
BeaEngine by BeatriX, 36
BeagleBoard, 77–78
BKPT instruction, 70
BL (Branch with Link) instruction, 58–59
Blink pointer, 113–114
blocks (scripting), 228–229
BLX (Branch with Link and Exchange) instruction, 59
bp command (software breakpoints), 210
branching and conditional execution (ARM), 61–66
.break command (scripting), 233
breakpoints (debugging), 208–211
bu command (unresolved breakpoints), 209
buffering methods (drivers), 151–152
BX (Branch and Exchange) instruction, 58
bytecode encoding/encryption, 316
bytes/word/dword/qword formats (registers), 202

C
C (Carry flag), 61–62
C++ evaluator syntax (expressions), 191
 CALL instruction, 15, 278, 281
@call script file alias, 244–249
callbacks, disassembler (Metasm), 296
calling conventions, 15–16
CALL-like save instruction pointer/branch, 279
calls window, 190
canonical addresses (x64), 37
.catch command token (scripting), 232
CBZ/CBNZ comparison instructions, 63
characters and strings (scripting), 227–228
circular doubly-linked lists, 112
CMN/TEQ comparison instructions, 64
CMOV instruction (Intel), 42
CMP instruction, 18, 62–63
code
code binding (Metasm), 302
code looping over arrays, 8
code obfuscation, 316–317
code virtualization, 285–286
code_binding method (Metasm), 316–317
code-flattening (deobfuscation), 316, 325
command/output window, 189–190
comments ($$ command) (scripting), 226–227
comparison instructions, 62–63
completeness/soundness of analysis algorithms, 290, 293–295
completion routines, 143–144
computational equivalence, 268
computational partial ordering, 290
concolic executions, 292, 331
conditional breakpoints, 210–211
conditional code (cc), 17–18, 62
conditional execution (ARM) basics, 42
branching and, 61–66
conditional statements (scripting), 229–231
Console Debugger (CDB), 188–189
constant folding obfuscation, 273
constant propagation algorithm, 291
constant unfolding compiler optimization, 271
constant unfolding compiler optimization, 273
constraint solvers (SMT/SAT), 292
CONTAINING _RECORD macro, 118–119
.continue command (scripting), 233
continuum of dynamic/static analysis, 291–293
control flow (x86 instruction set), 17–25
control indirection, 280–283
control-based obfuscations basics, 278–283
interplay with data-based obfuscations, 284–288
control-flow graphs (CFGs), 283, 285–286
coprocessors in ARM, 45
cPSR (current program status register), 41, 44
CreateToolhelp32Snapshot function, 32
current privilege level (CPL), 2

D
  d command (memory dump), 205
Data Execution Prevent (DEP), 182
data movement
  x64 architecture, 36–37
  x86 instruction set, 5–11
data types
  ARM architecture, 43–44
  x64 architecture, 36
  x86 architecture, 3
data-based obfuscations
  basics, 273–277
  interplay with control-based obfuscations, 284–288
data-encoding schemes (obfuscation), 273–274
DbgEng
  commands, 195
  DbgEng extension framework (SDK), 257
debugger interfaces, 258–261
  overview, 188
d-cache (ARM core), 67
dead code elimination (compiler optimization), 274–275
dead statement elimination, 271
Debug extension (ARM), 40
debugging tools (Windows)
  automating with SDK. See SDK for extending debugger
  breakpoints, 208–211
debugger interfaces (DbgEng), 258–261
  Debugger Markup Language (DML), 215
debugger windows, 189–190
evaluating expressions, 190–194
extensions/tools/resources, 264–265
inspecting processes and modules, 211–214
memory-related commands, 203–208
miscellaneous debugger commands, 214–216
  overview, 187–189
process and thread control, 194–198
registers management, 198–203
scripting with. See scripting with Debugging Tools
setting symbol paths, 189
symbols, 208
useful operators, 192–194
writing extensions, 262–264
decidable approximations of concrete semantics, 290–291
defered procedure calls (DPCs), 135–139
deobfuscation techniques
code-flattening, 325
continuum of dynamic/static analysis, 291–293
decidable approximations of concrete semantics, 290–291
  overview, 289–290
pattern-based, 312–315
program-analysis-based, 313–315
soundness/completeness concepts, 293–295
using Metasm, 317–325
using VxStripper, 325–328
VM implementations (code virtualization), 315–317
deobfuscation tools
IDA, 295–296
Metasm open source framework. See Metasm open source framework
Miasm reverse engineering framework, 302–304
summary of, 312
VxStripper binary rewriting tool. See
VxStripper binary rewriting tool
Desclaux, Fabrice, 302
descrambling routine (scripts),
255–256
device objects (drivers), 149–150
DeviceIoControl API, 151
Digital Rights Management (DRM),
268
Direct IO buffering method, 151
disassembler callbacks (Metasm), 296
disassembly window, 190
DISPATCH LEVEL (2) IRQL, 105
dispatcher switch statement, 285
display selector command (registers),
202–203
DIV/IDIV instructions, 13
DllMain routine walk-through (x86),
28–34
DMB instruction (ARM), 68
DosDevices string, 156
do-while loops (scripting), 235–236
DPC_WATCHDOG_VIOLATION (0x133),
139
drivers, kernel
analysis of real-life drivers, 184–185
basics, 146–147
driver and device objects, 149–150
DriverEntry function, 155
DriverUnload routine, 149
entry points, 147–149
IRP handling, 150
KeServiceDescriptorTable, 153–155
mechanisms for user-kernel
communication, 150–152
sections, 155
system control registers, 153
DriverUnload routine (x86 rootkit),
159–160
DSB instruction (ARM), 68
.dvfree command (memory), 238
Dynamic Binary Translator (DBT),
QUEMU, 305
dynamic IRPs, 145
dynamic slicing criteria, 291
dynamic/static analysis
(deobfuscation), 290–293
E
e command (memory editing),
206–207
Eagle, Chris, 295
EFLAGS register, common flags in, 17
EIP register, 2
else/.elsif command tokens, 229
encrypted programs upon encrypted
data, 274
Endianness bit (endianness), 44
EngExtCpp extensions (SDK), 257
ENOSE structures, 130
entry points (drivers), 147–149
errors, script (debugging), 231–232
ETHREAD kernel data structure,
107–108
ETHREAD objects, 132
events
monitoring (debugging), 197–198
signaled/non-sigaled, 110
exceptions
basics, 95
exception vectors, 68
exception/interrupt handling, 25,
27–28
monitoring (debugging), 197–198
Execute Never (XN), 182
execution context (Windows),
109–110
expressions, evaluating (debugging),
190–194
ExpWorkerThread, 131
extensions (debugging tools)
extension-provided foreach loops, 239–240
resources for, 264–265
writing (SDK), 262–264

guarded mutexes, 111
Guilfanov, Ilfak, 296
Guillot, Yoann, 296

F
fast mutexes, 111
fault exceptions, 28, 95
faults, page, 95
file monitor, writing (debug script), 253–255
files
file system minifilter drivers, 147
passing arguments to script files, 242–244
tokenizing from, 238
fixed-name aliases (DbgEng), 219, 225
Flink pointer, 113–114
floating-point formats (registers), 201
flushing the cache, 67
for loops (scripting), 233–234
forced-inline routines, 162
foreach loops (scripting), 236–240
forensics module/root-kit analysis (VxStripper), 309
frame pointer omission, 16
fully homomorphic mappings (obfuscation), 274
function invocation
ARM architecture, 57–60
x64 architecture, 37
x86 instruction set, 13–17
functions
function epilogue, 17
function pointers/offsets, 92–93
function prologue, 16
in/out-lining, 279
using scripts as, 244–249

G
general-purpose registers (GPRs), 2–3
goto constructs, 23
graph flattening, control-flow, 285–286

H
handlers semantics analysis
(obfuscation case study), 330–333
hardware breakpoints (DbgEng), 209
hardware interrupts, 28, 95
homomorphism property, 274

I
i-cache (ARM core), 67
IDA tool (deobfuscation), 295–296
idaocaml interpreter, 312
identities, arithmetic substitution via (obfuscation), 275
IDIV/DIV instructions, 13
IDT register, 31–32
.if command token, 229
if-else constructs, 18–20, 23
image base of specified module, getting (debug script), 249
Import Address Table (IAT), 310
IMUL instruction, 12
InitializeListHead function, 112–113
inline functions, 279
inspecting processes/modules (debugging), 211–214
instruction semantics (Metasm), 297–298
instructions (ARM)
basics, 46–47
conditional execution of, 70–71
for function invocations, 58
instructions (x86)
arithmetic operations, 11–13
control flow, 17–25
data movement, 5–11
function invocation, 13–17
overview, 3–4
stack operations, 13–17
syntax, 4–5
INT 3 instruction, 184
Intel 64 and IA-32 Architectures Software Developer’s Manual, 2
Intel Software Development Manual, 1
Intel syntax for x86 assembly code, 4–5
Intel/AMD reference manual, 31
intellectual property, protecting, 267
interfaces, debugger (SDK), 258–261
interrupts
basics (system calls), 95–98
exceptions and, 27–28
interrupt descriptor tables (IDTs), 95–96
interrupt request level (IRQL), 104–106
I/O request packets (IRPs), 144–146
I/O stack location structures, 175–176
I/O status block structure, 161
IoAllocateWorkItem function, 129
IoCompleteRequest, 143–144
IoCreateDevice (x86 rootkit), 156–157
IOCTL operations/codes, 151–152
IRP handlers, analyzing (drivers), 160–161
IRP handling (drivers), 150
ISB instruction (ARM), 68
IT (If-then bits) flag, 61–62
IT (if-then) instruction, 64

K
KAPC structure, 132
KDPC structure, 135–137
KdVersionBlock, 121, 123
KeInitializeApc API, 132–133
KeInsertQueueDpc kernel function, 115
kernel
Kernel Debugger (KD), 188–189
kernel drivers. See drivers, kernel
kernel memory space, 88
Kernel Patch Protection feature, 153
kernel synchronization primitives
(Windows), 110–111
kernel-mode APCs, 131–132
kernel-mode driver framework
(KMDF), 147
KeServiceDescriptorTable (drivers), 153–155
KeStackAttachProcess, 110
KLDR_DATA_TABLE_ENTRY structure, 121
KNODE structures, 130
KPRCB structure, 137
KSECDD driver, 183
KSPIN_LOCK structure, 111
KTHREAD kernel data structure, 107–108
KTHREAD structure, 134–135
KTIMER structure, 140–141

L
LARGE_INTEGER structure, 172
LDMFD pseudo-instruction, 55
LDM/STM instructions (ARM), 52–55
LDR pseudo-instructions (ARM), 51–52
LDREX instruction (ARM), 67–68
LDR/STR instructions (ARM), 47–51
LEA instruction, 5, 9
left/right shift instructions, 12
legacy filter drivers, 146
legacy software drivers, 146
link register (LR) (ARM), 43
lists (Windows kernel)
  implementation details, 112–119
  list manipulation functions in kernel
  mode walk-through, 119–123
  overview, 111–112
LLVM (Low Level Virtual Machine)
  compilation chain, 305, 307, 311, 325
lm command (kernel mode
  debugging), 211
loading/storing data (ARM)
  LDM/STM instructions, 52–55
  LDR pseudo-instructions, 51–52
  LDR/STR instructions, 47–51
  overview, 47
  PUSH/POP instructions, 56–57
LocalSystem privilege, 183
LODS instruction, 11
loops
  basics, 22–25
  unrolling, 316

M
Macro Assembler (MASM), 190–193
MajorFunction array, 158
malware, 267, 341
Manufacturing Cheap, Resilient, and
  Stealthy Opaque Constructs, 283
masks, register, 199–201
memory
  address and range notations, 203–204
  dumping contents of, 205–206
  editing contents of, 206–207
  memory breakpoints (DR0–DR3), 3
  memory descriptor lists (MDLs)
    (Windows), 106–107
  memory dump window, 190
  memory layout (Windows), 88–89
  memory-related commands
    (debugging), 203–208
  methods for specifying access, 5–6
  miscellaneous commands, 207–208
movement methods between
  registers and, 3–4
Metasm open source framework
  backtracking/slicing, 298–302
  code binding, 302
  Disassembler callbacks, 296
  instruction semantics, 297–298
  overview, 296
  using, 317–325, 331
Miasm reverse engineering
  framework, 302–304
MmGetPhysicalAddress API, 181
Mode bits (M), 44
model-specific registers (MSRs). See
  MSRs (model-specific registers)
  (x86)
modes, inspecting (debugging), 211–214
MOV/MOVSB/MOVSW/MOVSD instructions, 8–9
MOVW/MOVT instructions, 60
MRC (read)/MCR (write) instructions, 45
MSRs (model-specific registers) (x86), 3
MUL instruction, 12
MUL instruction (ARM), 61
mutexes, 111

N
N (Negative flag), 61–62
namomites feature (Armadillo), 283
Neither buffering method, 151–152
Never Execute (NX), 182
non-paged pool memory (Windows), 106
non-signaled events, 110
normalization module (VxStripper), 310–311
NOT operator, 275
NT Symbolic Debugger (NTSD), 188–189
NT_SUCCESS() macro, 157–158
obfuscation case study
analyzing handlers semantics, 330–333
first analysis, 328–330
overview, 328
solving the challenge, 334–336
symbolic execution, 333–334
obfuscation techniques
control-based, 278–283
data-based, 273–277
deobfuscation techniques. See deobfuscation techniques
eexample, 269–273
overview, 267–269
security by obscurity, 288–289
simultaneous control-flow/data-flow, 284–288
A Taxonomy of Obfuscating Transformations, 283
offset mode, 50–51
OllyDbg script, 312–313
opaque predicates, 283
opcodes, coprocessor, 45
operating system-control indirection, 282–283
operation-preserving mappings (algebraic structures), 274
operators (expressions), 192–194
Optimice plugin, 296
OR operator (|), 200
outline functions, 279
output of commands, tokenizing from, 237–238
over-approximation program analyses, 290

partial evaluation techniques (optimization), 291
PASSIVE LEVEL (0) IRQL, 105
PatchGuard, 153, 183
pattern-based deobfuscation, 312–313
pattern-based obfuscation, 271, 275–277
PC-relative addressing, 51
peephole compiler optimization, 271, 277
pool memory (Windows), 106
POP/PUSH operations (stacks), 13
post-indexed address mode, 50–51
predefined pseudo-registers, 216–218
.prefer_dll 1 command, 215
pre-indexed address mode, 50–51
.printf command (debugger), 214–216
.printf function, 23
privileges, modes defining (ARM), 40–41
process environment block (PEB), 109
!process extension command, 212
processes and threads basics, 107–109
callbacks, 142
control (debugging), 194–198
processors
processor control block structures (Windows XP), 120–121
processor control region (PCR), 89
processor initialization (Windows), 89–91
processor region control block (PRCB), 89–90
processor-based control indirection (obfuscation), 280–282
program counter (PC) (ARM), 43–44
program transformations (obfuscation), 268
program-analysis-based deobfuscation, 313–315
protected mode (x86), 1
PsCreateSystemThread API, 128
pseudo-instructions, 51, 55
pseudo-registers (scripting), 216–219
PUSH/POP
operations (stacks), 13
instructions (ARM), 56–57
PUSH-RET instruction, 279, 282

Q
QEMU (Quick EMUlator) Dynamic
Binary Translator (DBT), 305–307

R
r command (registers), 198–199
Radare framework, 312
Range parameter (memory), 204
RBP register, 36
RDMSR/WRMSR instructions, 3
RDTS C instruction, 3
real mode (x86), 1
references (Windows kernel), 179
registers
ARM architecture, 43–44
management (debugging), 198–203
window, 190
x64 architecture, 36
x86 architecture, 2–4
RegistryPath (drivers), 149
REP prefix, 9–10
repetition structures (scripting)
do-while loops, 235–236
foreach loops, 236–240
for loops, 233–234
overview, 232–233
while loops, 234–235
RESET exception handler, 68
resources, debugging, 264–265
RET instruction, 15, 278, 298
return addresses, 55
Reversing a simple virtual machine, 316
right/left shift instructions, 12
ring levels (x86), 1–2
RIP-relative addressing (x64), 36–37, 51
Rolles, Rolf, 312
rootkits
root-kit analysis (VxStripper), 309
x64 rootkit analysis walk-through, 172–178
x86 rootkit analysis walk-through, 156–171
Rough C, 25
round-based cryptography, 288–289

S
SCAS instructions, 10–11
scripting with Debugging Tools
aliases. See aliases (DbgEng)
blocks, 228–229
characters and strings, 227–228
comments ($$ command), 226–227
conditional statements, 229–231
debug script examples, 249–256
pseudo-registers, 216–219
repetition structures. See repetition
structures (scripting)
script errors, 231–232
script file commands, 240–244
using scripts like functions, 244–249
SCSI Commands Reference Manual, 185
SDK for extending debugger
debugger interfaces, 258–261
extension resources, 264–265
overview, 257–258
WinDbg extension APIs, 261–262
writing extensions, 262–264
sections (drivers), 155
security, achieving by obscurity, 288–289
SEH handler, 282
semantics
concrete, 290–291
instruction, 297–298
semantic equivalence, 268
sequenced singly-linked lists, 112
sequential locality, 279–280
service table descriptors, 92–93
SETNE instruction, 42
SHA1 hashes (malware), 341
SIDT instruction, 30
signaled events, 110
signed/unsigned division, 13
single-instruction multiple data (SIMD) instruction set, 45
singly-linked lists, 112
slicing/backtracking (Metasm), 297–302
SMC (self-modifying code), 67
SMULL instruction (ARM), 61
software breakpoints (DbgEng), 208–210
software interrupts (ARM), 69
soundness/completeness of analysis algorithms, 290, 293–295
spaghetti code, 279
Spasojevic, Branko, 296
spin locks, 111
stack operations (x86), 13–17
stack pointer (SP) (ARM), 43
stack-based obfuscation, 271
static IRPs, 145
static slicing criteria, 291
static/dynamic analysis techniques, 290, 291–293
STDCALL calling convention, 32
STMFpD pseudo-instruction, 55
STMIA/STMEA pseudo-instructions, 47
STOS instructions, 10–11
strength reduction (arithmetic operations), 12
STREX instruction (ARM), 67–68
strings (scripting)
and characters, 227–228
tokenizing from, 236–237
writing basic descrambler script, 255–256
strlen() function (C), 10
struct field names, 74
SUB instruction, 14
sub_10460 leaf routine, 158, 166
sub_10550 routine, 166
sub_11553/sub_115DA functions, 121
SVC instruction, 102
SVC mode, 70
SWI/SVC instruction, 69
switch-case blocks, 19–20
switch-case statements (ARM), 65–66
sxe/sxd commands (debugging), 198
symbolic executions, 292–294, 322,
333–334
symbols
commands for inspecting, 208
symbol paths, setting (debugging), 189
synchronization primitives
ARM architecture, 67–68
kernel, 110–111
syntax
ARM assembly, 46
derivation evaluation (debugger), 190
notations for x86 assembly code/
Intel/AT&T, 4–5
SYSENTER instruction, 28, 100–101
system calls (Windows)
basics, 92–94
faults/traps/interrupts overview, 94–95
interrupts, 95–98
traps, 98–104
system context (kernels), 109–110
system control coprocessor (CP15), 45
system control registers (drivers), 153
system services and mechanisms (ARM), 68–70
system threads, 128–129
system-level controls/settings (ARM), 45
system-level mechanisms (x85), 25–28

T
A Taxonomy of Obfuscating Transformations, 283
TBB/TBH instructions (ARM), 65–66
TCG (Tiny Code Generator), 305–307
TEB (Thread Environment Block) structure, 203
temporal locality, 279–280
testing
  ARM knowledge, 77–78
dynamic, 292
TEST instruction, 18
threads
  processes and. See processes and threads
thread and process callbacks, 142
thread context (kernels), 109–110
thread environment block (TEB), 109
thread suspension with APCs, 134
Thumb bit (T), 44
Thumb extension (ARM), 40
Thumb state, 41–42
Thumb-2-specific instructions (ARM), 64–65	
timers, 110–111, 140–141
tokenizing
  from files, 238
  from output of commands, 237–238
  from strings, 236–237
trace semantics, 290
transformation inversion. See deobfuscation techniques
translation blocks (TCG instructions), 305
trap exceptions, 28
traps (system calls), 95, 98–104

U
U-Boot bootloader, 69
UDPRs (user-defined pseudo-registers). See users
UMULL instruction (ARM), 61
under-approximation program analyses, 290
UNICODE_STRING structure, 156
unpacking module (VxStripper), 309–310
unresolved breakpoints (debugging), 209–210
unsigned/signed division (DIV/IDIV), 13
UPX unpacker, writing (debug script), 250–253
users
  user-defined pseudo-registers (UDPRs), 218–219, 242, 244–249
  user/kernel address ranges, 89
  user-kernel communication (drivers), 150–152
  user-mode APCs, 131–132
  user-mode driver framework (UMDF), 147
  user-named aliases (DbgEng), 219–224

V
v (Overflow flag), 61–62
Vellvm (Verified LLVM), 305
virtual address translation, 25–27
virtual black box property, 287–288
Virtual machines re-building, 316
virtual machines (VMs), 286
VM implementations (code virtualization), 315–317
VxStripper binary rewriting tool
  API hooking module of, 309
  architecture of, 308
  basics, 304–305
  forensics module/root-kit analysis, 309
  normalization module, 310–311
  QEMU DBT extension, 306–307
  unpacking module, 309–310
  using, 325–328

W
walking back control flows, 299–300
WdbgExts extension framework (SDK), 257–261
websites for downloading
  idaocaml interpreter, 312
  Metasm open source framework, 296
Miasm reverse engineering framework, 302
Optimice plugin, 296
Radare framework, 312
websites for further information
BeaEngine by BeatriX, 36
control-flow deobfuscations, 284
debugging, 264–265
x64 ABI on Windows, 37
x86 Opcode and Instruction Reference, 296
while loops (scripting), 234–235
white box attack context (WBAC), 268, 286–288
Win32 API functions, 32
WinDbg extension APIs (SDK), 261–262
WinDbg graphical interface, 188–189
windows, debugger, 189–190
Windows Driver Model (WDM), 147
Windows fundamentals
execution context, 109–110
interrupt request level (IRQL), 104–106
kernel synchronization primitives, 110–111
memory descriptor lists (MDLs), 106–107
memory layout, 88–89
pool memory, 106
processes and threads, 107–109
processor initialization, 89–91
system calls. See system calls (Windows)
Windows kernel
asynchronous and ad-hoc execution.
See asynchronous and ad-hoc execution
I/O request packets (IRPs), 144–146
kernel drivers. See drivers, kernel lists. See lists (Windows kernel) overview, 87–88
references/tips/exercises, 179–184
x64 rootkit walk-through, 172–178
x86 rootkit walk-through, 156–171
Windows object manager, 156
work items, 129–131

X
x64 architecture
canonical addresses, 37
data movement, 36–37
function invocation, 37
register sets and data types, 36
rootkit walk-through, 172–178

x86 architecture
DllMain routine walk-through, 28–34
instruction set. See instructions (x86)
Opcode and Instruction Reference, 296
overview, 1–2
register sets and data types, 2–3
rootkit walk-through, 156–171
system-level mechanisms, 25–28
variable-length instruction ranges, 4
XOR swap trick, 270

Z
z (Zero flag), 61–62
-z command-line switch, 188–189