INDEX

A

ABI (application binary interface), 353
abstract syntax tree (AST), 335–337
address alignment, 400
address-taken functions, 134
anti-debugging, 156
anti–dynamic analysis methods, 125–126
application binary interface (ABI), 353
array assignment, assembly representation, 137
AsPack, 252
assembly language, 373
  comments, 374–375
  common code constructs, 383
  conditional branch implementation, 388
  directives, 374–375
  function calls, 384
  function frames, 384
  instruction format, 376
  instructions, 374–375
  labels, 374–375
  loop implementation, 389–390
  mnemonics, 376
  operands, 376
  program layout, 374
  the stack, 383
assembly phase, of compilation, 16
AST (abstract syntax tree), 335–337
AT&T syntax, 6, 376
automatic unpacking, 252, 258

B

back edge, in CFG, 147
backward slicing, 151, 337
`base64` utility, 91
Base64 encoding, 90–91
basic blocks, 132–133
big-endian, 34
binary analysis, overview, 2–3
binary executable, 2, 11
  formats
    ELF, *see* ELF format
    PE, *see* PE format
  loading, 27, 50
  loading (static), 67–85
Binary File Descriptor library, *see* `libbfd`
binary instrumentation, 2, 224
  dynamic, 225, 233–236
    architecture, 233
    implementation, 233
  instrumentation code, 224
  instrumentation point, 224
  state saving, 227, 230–231, 235
  static, 225, 226–233
    `int 3` approach, 227
    trampoline approach, 228–233
  trade-offs of static and dynamic, 225
binary rewriting, *see* binary instrumentation, static
branch constraint, 310–311
branch edges, 132–133
breadth-first search, 318
breakpoints, 112
  implementation with `int 3`, 227–228
buffer overflow, 293, 356
  detection using taint analysis, 295
buffer overread, 268–269
buffer overwrite, *see* buffer overflow

C

`c++filt` (demangling utility), 101
C++ function name mangling, 99
call graph, 133
calling convention, for function calls, 353, 386
Capstone disassembly framework, 196
  API data types
    `cs_arch`, 200, 203
    `cs_detail`, 202, 203–209, 210
    `cs_err`, 200, 202, 203
    `csh`, 200

Capstone disassembly framework, *continued*
  API data types, *continued*
    cs_insn, 201, 203, 208
    cs_mode, 200, 203
    cs_x86, 203–204, 210–211
    cs_x86_op, 210–211
  API functions
    cs_close, 202
    cs_disasm, 200–201
    cs_disasm_iter, 209
    cs_errno, 202
    cs_free, 202
    cs_malloc, 208
    cs_open, 200
    cs_option, 204
    cs_strerror, 202
  capstone.h, 203
  detailed disassembly, 204, 210–211
  header files, 203–204
  instruction groups, 210
  iterative disassembly, 209
  linear disassembly, 198
  operand inspection, 210
  Python API, 197
  recursive disassembly, 204
  x86.h, 203
Capture the Flag (CTF), 89
CFG (control-flow graph), 132
code coverage, 125, 309–310
  fuzzing, 127
  symbolic execution, 127
    implementation with Triton, 346
  using test suite, 126
code coverage problem, 122, 125, 316
code injection, 171–175, 399–400
  constructor hijacking, 179
  destructor hijacking, 179
  entry point modification, 176
  GOT hijacking, 182
  hijacking direct calls, 186
  hijacking indirect calls, 186
  injecting a code section, 169
  overwriting padding bytes, 156
  overwriting unused code, 156
  PLT hijacking, 185
code modification, with hex editor, 160–162
code vs. data problem, 3, 117
command line options, passing in gdb, 112
compilation phase, of compilation, 14
compiler
  assembly phase, 16
  compilation phase, 14
  compilation process, 12
  linking phase, 17
  optimization, 14
  preprocessing phase, 12
concolic execution, 316, 334
constant propagation, 150
constraint solver, 128, 313, 321
  bitvector, 321
  satisfiability, 322
  SAT solver, 321
  SMT solver, 321
  validity, 325
  Z3, *see* Z3 (constraint solver)
context-sensitivity, 144
control dependency, 275
control-flow analysis, 146
control-flow graph (CFG), 132
control hijacking attack, 293
  detection using taint analysis, 295
CTF (Capture the Flag), 89
cycle detection, 146, 147–148
cycle (loop), 146

D

data-flow analysis, 146, 148
data flow tracking (DFT), *see* dynamic taint analysis
data structure detection, through library calls, 136
DBI (dynamic binary instrumentation), 223, 225, 233–236
dd utility, 95
dead code elimination, 143
debugger detection, 125
decompilation, 138, 150
def-use chain, 150
demangling C++ symbols, 99
depth-first search, 318
DFT (data flow tracking), *see* dynamic taint analysis
disassembler desynchronization, 117

disassembly, 116
  dynamic, 116, 122
  linear, 117, 198
    implementation in Capstone, 198
    for stripped binaries, 25
    with symbols, 23
  object file, 21
  recursive, 118, 204
    entry points, 209
    implementation in Capstone, 204
    termination conditions, 209, 211
  static, 116
displaying an instruction, in gdb, 112
displaying register contents, in gdb, 112
dlsym function, 167
dominance trees, 146–147
DSE (dynamic symbolic execution), 316, 334
DTA, *see* dynamic taint analysis
dumping memory contents, in gdb, 113
dynamic analysis, 2
dynamic binary instrumentation (DBI), 223, 225, 233–236
dynamic disassembly, 116, 122
dynamic library, 17, 48
dynamic linker, 18, 47
dynamic loader, 27, 50
dynamic symbolic execution (DSE), 316, 334
dynamic taint analysis (DTA), 266
  accuracy of, 272
  control dependency, 275, 296
  detecting Heartbleed, 269
  in fuzzing, 270
  implicit flows, 275, 296
  overtainting, 274–275
  performance of, 272
  shadow memory, 276
  taint colors, 272, 297
  taint granularity, 271
  taint policy, 267, 273
  taint propagation, 267, 270, 273
  taint sink, 267
  taint source, 266
  undertainting, 274–275
DynamoRIO, 233
Dyninst, 226, 233

E

effectively propositional formulas, 326
.eh_frame section, function detection with, 131
ELF format, 31
  alignment requirements for loadable segments, 400
  entry point, 36
  executable header, 33, 97
    struct Elf32_Ehdr, 411
    struct Elf64_Ehdr, 33, 411
  injecting a code section, 169
  loading, 50
  overwriting PT_NOTE segment, 170, 391–411
  program header, 52
    struct Elf32_Phdr, 410
    struct Elf64_Phdr, 52, 410
  program header table, 36, 52
  relocations
    R_X86_64_GLOB_DAT, 49
    R_X86_64_JUMP_SLO, 49
  section, 38, 41
    .bss, 44
    .data, 44
    .dynamic, 50
    .dynstr, 52
    .dynsym, 52
    .fini, 43
    .fini_array, 51
    .got, 45
    .got.plt, 45
    .init, 43
    .init_array, 51
    .note.ABI-tag, 170
    .note.gnu.build-id, 170
    .plt, 45
    .plt.got, 46
    .rel, 48–49
    .rela, 48–49
    .rodata, 21, 44
    .shstrtab, 52
    .strtab, 52
    .symtab, 52
    .text, 43
  section header, 38
    struct Elf32_Shdr, 405
    struct Elf64_Shdr, 38–39, 405

ELF format, *continued*
section header table, 36, 38
segment, 38, 52
PT_NOTE, 170
string table, 37
elfinject, 169, 171, 391
endianness, 34
entry point, of a binary, 36
environment variable, setting in gdb, 112
executable packer, *see* packer
execution tracing, 116, 122

F

file utility, 90
flow-sensitivity, 143–144
forward slicing, 151
function detection, 26, 130
function epilogue, 131
function frame, 384
function inlining, 152
function prologue, 131
function signature, 130
fuzzing, 127, 270, 355

G

gdb (GNU debugger), 111–113
GElf, *see* libelf
Global Offset Table (GOT), 45

H

header node, of a loop, 147
head utility, 90
heap overflow, 163–165
Heartbleed vulnerability, 268
hexadecimal system, 94
hex dump, 94–95
hex editor, 94–95, 155–156, 160–162
Hex-Rays decompiler, 138

I

ICFG (interprocedural control-flow graph), 133
indirect control flow, 118, 119–122
inline data, 117
inlining, 152
instruction reference, 156
instruction set architecture (ISA), 6, 373
instruction tracing, 116, 122
instrumentation code, 224
instrumentation point, 224
int 3 (binary instrumentation), 227–228
Intel Pin, *see* Pin
Intel syntax, 6, 376
intermediate language, 140
intermediate representation (IR), 140
interpreter, 28, 36, 106
interprocedural analysis, 142
scalability of, 142
interprocedural control-flow graph (ICFG), 133
intraprocedural analysis, 142
ISA (instruction set architecture), 6, 373

J

jump table, 121
just in time (JIT) compilation, 12, 235, 236

L

lazy binding, 28, 45
ldd utility, 93
LD_LIBRARY_PATH, 101
ld-linux.so, 28, 36
LD_PRELOAD, 163
libbfd, 68, 72
API data types
asection, 82
asymbol, 78
bfd, 74
bfd_arch_info_type, 77
bfd_error_type, 74
bfd_flavour, 75, 77
bfd_format, 74
bfd_section, 82
bfd_symbol, 78
bfd_target, 77
bfd_vma, 77
API functions
bfd_asymbol_value, 79
bfd_canonicalize_dynamic_symtab, 81

libbfd, *continued*
  API functions, *continued*
    bfd_canonicalize_symtab, 79
    bfd_check_format, 74
    bfd_close, 77–78
    bfd_errmsg, 74
    bfd_get_arch_info, 77
    bfd_get_dynamic_symtab_upper_bound, 81
    bfd_get_error, 74
    bfd_get_flavour, 75
    bfd_get_section_contents, 83
    bfd_get_section_flags, 82
    bfd_get_start_address, 77
    bfd_get_symtab_upper_bound, 79
    bfd_init, 74
    bfd_openr, 74
    bfd_section_name, 82–83
    bfd_section_size, 82–83
    bfd_section_vma, 82–83
    bfd_set_error, 74–75
  bfd.h, 72
libdft, 279
  API data structures
    ins_desc, 286
    syscall_desc, 282, 285, 298
  API data types
    syscall_ctx_t, 289, 301, 302, 304
  API functions
    ins_set_post, 282, 286
    ins_set_pre, 282, 286
    libdft_die, 285
    libdft_init, 285
    likely, 285
    syscall_set_post, 282, 285, 298
    syscall_set_pre, 282, 285, 298
    tagmap_clrn, 302
    tagmap_getb, 282, 287, 304
    tagmap_getl, 287
    tagmap_getw, 287
    tagmap_setb, 282, 290
    tagmap_setl, 290
    tagmap_setn, 289, 302
    tagmap_setw, 290
    unlikely, 285
  branch_pred.h, 285
  header files, 285
  internals, 280
  libdft_api.h, 285
  segment translation table (STAB), 280
  shadow memory, 280
  syscall_desc.h, 285
  tagmap, 280, 285
  tagmap.h, 285
  taint policy, 282
  virtual CPU, 281–282
libdwarf, 20
libelf, 169, 392
  API data types
    Elf, 395
    GElf_Ehdr, 397, 411
    GElf_Phdr, 399, 410
    GElf_Shdr, 403, 405
  API functions
    elf32_getehdr, 397, 411
    elf32_getphdr, 410
    elf32_getshdr, 405
    elf64_getehdr, 397, 411
    elf64_getphdr, 410
    elf64_getshdr, 405
    elf_begin, 394
    elf_end, 397
    elf_errmsg, 398
    elf_errno, 398
    elf_getphdrnum, 399
    elf_getshdrstrndx, 402
    elf_kind, 394
    elf_ndxscn, 403
    elf_nextscn, 403
    elf_version, 394
    gelf_getclass, 394
    gelf_getehdr, 397
    gelf_getphdr, 399
    gelf_getshdr, 403
    gelf_update_ehdr, 411
    gelf_update_phdr, 410
    gelf_update_shdr, 405
  gelf.h, 392
  header files, 392
  libelf.h, 392
library preloading, 163
linear disassembly, 117, 198
linker, 17
linking phase, of compilation, 17
link-time optimization (LTO), 17, 152
little-endian, 34
LLVM bitcode, 140

LLVM IR, 140
loader (static), implementation, 72–83
loading, 27, 50
loading (static), 67–85
loop detection, 146
loop unrolling, 152
LTO (link-time optimization), 17, 152
`ltrace` utility, 104, 107

M

machine code, 11, 376
malware analysis, 2–3
mangled function names (C++), 99
McSema, 140
module, 16

N

natural loop, 146
`nm` utility, 99
nonreturning function, 121
now binding, 46
ntdll.dll, 28

O

obfuscation
  disassembling overlapping basic blocks, 204, 211
  instruction/basic block overlapping, 192
  opaque predicate, 195, 329
`objdump` utility, 21, 109
  disassembling raw binary, 263
object file, 16
object-oriented code, 135
object-oriented reverse engineering, 135
OEP (original entry point), 252, 258
off-by-one bug, 156–159
opaque predicate, 195, 329
opcode reference, 156
optimization, effect on disassembly, 152
overlapping basic block, 130, 192
overlapping instruction, 192
overtainting, 274–275

P

packer, 251
  AsPack, 252
  original entry point (OEP), 252, 258
  UPX, 252, 259
path constraint, 128, 310, 321
path explosion problem, 318
PEBIL, 226
PE format, 57
  base address, 62
  data directory, 62, 64
  export directory, 64
  Import Address Table (IAT), 64
  import directory, 64
  MS-DOS header, 58
    `struct IMAGE_DOS_HEADER`, 58
  MS-DOS stub, 58
  MZ header, 58
  PE file header, 58, 61
    `struct IMAGE_FILE_HEADER`, 61
  PE optional header, 58, 62
    `struct IMAGE_OPTIONAL_HEADER64`, 62
  PE signature, 58, 61
  Relative Virtual Address (RVA), 62
  section, 63
    `.bss`, 63
    `.data`, 63
    `.edata`, 64
    `.idata`, 64
    `.rdata`, 63
    `.reloc`, 63
    `.text`, 63
  section header, 62
    `struct IMAGE_SECTION_HEADER`, 62
  section header table, 62
  `struct IMAGE_NT_HEADERS64`, 58
  thunk, 64
pentesting, 2–3
PIC (position-independent code), 152
PIE (position-independent executable), 152
Pin, 235
  analysis routine, 236, 246
  API, 236
  API data types
    `BBL`, 242
    `CONTEXT`, 247
    `IMG`, 239, 240, 242

Pin, *continued*
API data types, *continued*
INS, 239, 244, 255–256
KNOB, 238, 254
RTN, 241
SEC, 240–241
SYSCALL_STANDARD, 247
TRACE, 239, 242
API functions
BBL_InsertCall, 242
BBL_Next, 242
BBL_NumIns, 242
BBL_Valid, 242
IMG_AddInstrumentFunction, 239
IMG_FindByAddress, 242
IMG_IsMainExecutable, 242
IMG_SecHead, 240–241
IMG_Valid, 240–241, 242
INS_AddInstrumentFunction, 239, 254–255
INS_HasFallthrough, 245, 256
INS_hasKnownMemorySize, 255–256
INS_InsertCall, 244, 256
INS_InsertPredicatedCall, 244, 256
INS_IsBranchOrCall, 244, 256
INS_IsCall, 246
INS_IsIndirectBranchOrCall, 256
INS_IsMemoryWrite, 255–256
INS_OperandCount, 256
PIN_AddFiniFunction, 239, 254–255
PIN_AddSyscallEntryFunction, 239
PIN_Detach, 250
PIN_GetSyscallNumber, 247
PIN_Init, 239, 254–255
PIN_InitSymbols, 238
PIN_SafeCopy, 257
PIN_StartProgram, 240, 254–255
RTN_Address, 240–241
RTN_Name, 240–241
RTN_Next, 240–241
RTN_Valid, 240–241
SEC_Next, 240–241
SEC_RtnHead, 240–241
SEC_Valid, 240–241
TRACE_AddInstrumentFunction, 239
TRACE_BblHead, 242
architecture, 233, 236
attaching to a process, 249
detaching from a process, 250
documentation, 235
example tools, 235
insertion points, 242–243, 244, 255–256
instrumentation arguments, 242–243, 244, 255–256
instrumentation routine, 236, 241
introduction to, 235
pin.H, 238
Pintool, 236, 237, 251
for profiling, 237
running your, 247, 260–261
for unpacking, 251
reading application memory, 257
running Pin, 247, 260–261
PLT (Procedure Linkage Table), 45
position-independent code (PIC), 152
position-independent executable (PIE), 152
preloading, library, 163
preprocessing phase, of compilation, 12
procedural language, 135
Procedure Linkage Table (PLT), 45
process, 28, 106
program slicing, 150
backward, 151, 337
forward, 151
PT_NOTE overwriting, 170, 391–411

R

reaching definitions analysis, 148
readelf utility, 96
recursive disassembly, *see* disassembly, recursive
REIL (Reverse Engineering Intermediate Language), 140
relocation, 22, 28, 40, 45–49
relocation symbol, 17, 22
RELRO (relocations read-only), 45
return-oriented programming (ROP), 213–214
reverse engineering, 2
Reverse Engineering Intermediate Language (REIL), 140
ROP gadget scanner, 215, 221
ROP (return-oriented programming), 213–214
RTLD_NEXT, 168
running a binary, in gdb, 112

S

satisfiability, 322
SAT solver, 321
SBI, *see* binary instrumentation, static
shadow memory, 276–277, 280
shared library, 17, 48
`SIGTRAP`, 228
slicing, *see* program slicing
slicing criterion, 150
SMT solver, 321
SSA (static single assignment) form, 323–324
SSE (static symbolic execution), 314–315, 334
stack frame, 384
stack memory, 383
static analysis, 2
static binary instrumentation, *see* binary instrumentation, static
static library, 17
static single assignment (SSA) form, 323–324
`strace` utility, 104
`strings` utility, 102, 261
string table, 37
  modifying, 406–408
`strip` utility, 20
stripped binary, 3, 19, 20
struct assignment, assembly representation, 137
switch detection, 121
`switch` statement, assembly representation, 121
symbex, *see* symbolic execution
symbolic emulation, 314–315, 334
symbolic execution, 127, 309–310
  address concretization, 317
  branch constraint, 310–311
  code coverage, 309–310, 318
  concolic execution, 316, 334
  concrete state, 317
  constraint solver, 128, 313, *see also* constraint solver
  copy on write, 317
  dynamic, 316, 334
  fully symbolic memory, 317
  model, 313, 324
  offline, 316
  online, 316
  optimization of, 319
  path constraint, 128, 310, 321
  path coverage, 318
  path explosion, 318
  path selection heuristics, 318
  scalability of, 129, 319
  static, 314–315, 334
    environment interactions, 315
  symbolic expression store, 310
  symbolic memory access, 317
  symbolic pointer, 317
  symbolic state, 310, 317
  symbolic value, 128, 310
  symbolic variable, 128, 310
symbolic information, 3, 18, 78
  DWARF format, 19
  parsing, 20
  parsing with `libbfd`, 78
  parsing with `libdwarf`, 20
  PDB format, 19
  symbol file, 19
symbolic reference, 17
symbolic value, 128, 310
symbolic variable, 128, 310
symbols, *see* symbolic information
syscall number, 284
System V ABI, 34, 353, 386–387

T

tail call, 130
`tail` utility, 90
taint analysis, *see* dynamic taint analysis
trampoline (binary instrumentation), 228–233
Triton (symbolic execution engine), 334
  `ALIGNED_MEMORY` optimization, 340
  API data types
    `triton::arch::architectures_e`, 344
    `triton::arch::Instruction`, 342
    `triton::arch::MemoryAccess`, 348
    `triton::arch::Register`, 341
    `triton::arch::registers_e`, 341
    `triton::ast::AbstractNode`, 350
    `triton::ast::AstContext`, 350
    `triton::engines::solver::SolverModel`, 351
    `triton::engines::symbolic::PathConstraint`, 350

Triton (symbolic execution engine), *continued*
API data types, *continued*
triton::engines::symbolic::SymbolicExpression, 342, 345
triton::modes::mode_e, 340, 348
API functions
triton::API::convertMemoryToSymbolicVariable, 348
triton::API::convertRegisterToSymbolicVariable, 348
triton::API::enableMode, 340
triton::API::getAstContext, 350
triton::API::getConcreteRegisterValue, 342, 343
triton::API::getModel, 351
triton::API::getPathConstraints, 350
triton::API::getRegister, 341, 343
triton::API::getSymbolicRegisters, 345
triton::API::getSymbolicVariableFromId, 351–352
triton::API::processing, 342, 343
triton::API::setArchitecture, 344
triton::API::setConcreteMemoryValue, 342
triton::API::setConcreteRegisterValue, 341
triton::API::sliceExpressions, 345
triton::API::unrollAst, 337
triton::arch::Instruction::setOpcode, 342
triton::arch::MemoryAccess::MemoryAccess, 348
triton::arch::Register::getName, 348
triton::ast::AstContext::bvtrue, 350
triton::ast::AstContext::equal, 350
triton::ast::AstContext::land, 351
triton::engines::solver::SolverModel::getValue, 351–352
triton::engines::symbolic::PathConstraint::getBranchConstraints, 350
triton::engines::symbolic::PathConstraint::isMultipleBranches, 350
triton::engines::symbolic::SymbolicExpression::getComment, 345
triton::engines::symbolic::SymbolicExpression::setComment, 342
triton::engines::symbolic::SymbolicVariable::setComment, 348
AST reference nodes, 337
AST representation, 337
automatic exploitation with, 355
backward slicing with, 337
code coverage with, 346
Python API data types
triton.AstContext, 366–367
triton.Instruction, 366
triton.MemoryAccess, 365
triton.Register, 366
triton.SymbolicVariable, 365
triton.TritonContext, 363
Python API functions
AstContext.bv, 366–367
AstContext.bvuge, 367
AstContext.bvule, 367
AstContext.equal, 366–367
AstContext.land, 367
Instruction.getAddress, 366
Instruction.getOperands, 366
Instruction.isControlFlow, 366
pintool.getCurrentMemoryValue, 365
pintool.getCurrentRegisterValue, 365
pintool.getTritonContext, 363
pintool.insertCall, 363
pintool.startAnalysisFromAddress, 363
pintool.startAnalysisFromSymbol, 363
Register.getType, 366
SymbolicVariable.getComment, 367
SymbolicVariable.setComment, 365

Triton (symbolic execution engine), *continued*
  Python API functions, *continued*
    `TritonContext.convertMemoryToSymbolicVariable`, 365
    `TritonContext.enableMode`, 363
    `TritonContext.getAstContext`, 366
    `TritonContext.getAstFromId`, 366-367
    `TritonContext.getModel`, 367
    `TritonContext.getSymbolicRegisterId`, 366
    `TritonContext.getSymbolicVariableFromId`, 367
    `TritonContext.getSymbolicVariables`, 367
    `TritonContext.setArchitecture`, 363
    `TritonContext.setConcreteMemoryValue`, 365
    `TritonContext.unrollAst`, 366
  Python API modules
    `pintool`, 362
    `triton`, 362
  `triton` (wrapper script), 362
two's complement, 380
type information, 3

U

undertainting, 274–275
unpacking, 251 , 252, 258
UPX, 252, 259
use-def chain, 149–150

V

validity, of a formula, 325
value set analysis (VSA), 144
VEX IR, 140
  data types
    `Ity_I64`, 141
  IMark (Instruction Mark), 141
  instructions
    `Add64`, 141
    `GET`, 141
    `PUT`, 141
  IR Super Block (IRSB), 141
  jump kinds
    `Ijk_Boring`, 141
    `Ijk_Call`, 141
    `Ijk_Ret`, 141
virtual CPU, 281–282
virtual memory, 28
virtual memory address (VMA), 28
vtable, 135
vulnerability detection, 2–3

W

WinNT.h, 58

X

x86 encoder/decoder library (XED), 286
x86 instruction set (ISA), 373
  base/index/scale addressing, 379
  conditional branch, 382, 388
  conditional jump, 382, 388
  control register, 379
  debug register, 379
  endianness, 380
  function call, 384
  function frame, 384
  general purpose register, 378
  immediate operand, 380
  instruction format, 376–377
    addressing mode, 377
    immediate operand, 380
    memory operand, 379
    MOD-R/M byte, 377
    opcode, 377
    operand, 377
    prefix, 377
    register operand, 377
    SIB byte, 377
  instruction overview, 380
  loop, 389–390
  memory operand, 379
  model-specific register (MSR), 379
  properties of, 6, 376–377
  red zone, 387

x86 instruction set (ISA), *continued*
  register, 378–379
    control, 379
    debug, 379
    general purpose, 378
    model-specific, 379
    overview, 378
    `rflags`, 379, 382
    `rip`, 379
    segment, 379
  `rip`-relative addressing, 380
  segment register, 379
  signed integers, 380
  stack, 383
  status flags, 382
  syntax, 6, 376
  system calls, 382
x86 opcode reference, 156
x86/x64 instruction set, *see* x86 instruction set (ISA)
XED (x86 encoder/decoder library), 286
`xxd` utility, 94–96

Z

Z3 (constraint solver), 321
  arithmetic operators, 324
  assertion, 322
  bitvectors, 323, 327–330
  commands
    `assert`, 324
    `check-sat`, 324
    `declare-const`, 323
    `define-fun`, 324
    `get-model`, 324
    `simplify`, 327
  data types
    `Array`, 323
    `Bool`, 323, 326
    `Int`, 323
    `Real`, 323
  logical operators, 326
  proving opaque predicates, 329
  satisfiability, 322
  validity, 325