INDEX

A

abstract syntax trees (ASTs), 74-76 fuzzing, 280-281 vs. Semgrep approach, 86 Semgrep dump-ast feature, 98–99 access control list (ACL), 64 accumulated complexity, 254 Acer QuickAccess, 64 Active Server Pages (ASP) files, 152 AddressSanitizer (ASan), 215-216, 248 Adobe Systems Incorporated, 293-294 AFL++ (American Fuzzy Lop plus plus) afl-cmin tool, 243-244 afl-cov tool, 248-250 afl-plot tool, 276-277 CMPLOG mode, 259 coverage-guided fuzzing with, 234-238 dictionaries, 275-278 Frida mode, 259-262 Grammar Mutator project, 279-280 patching validation checks, 242 QEMU mode, 258-259 writing harnesses, 246-247 "afl-fuzz" blog post (Zaleweski), 233-234 AFP (Apple Filing Protocol), 7, 295 Agent Extensibility Protocol (AgentX), 52 - 55Amegma Galaxy Attack, reverse engineering, 122-126 Analyze function, ILSpy, 131 Andriesse, Dennis, xxv Android Package (APK) format, 69-70 "Angelboy" security researcher, 7 angr framework, 190-191 performing symbolic execution, 191-193 solving constraints, 193-195 static analyses supported by, 192 writing SimProcedures, 195–199 angr-management frontend, 190 antivirus software, 296

Apache Log4j vulnerability, 6, 9, 268 Apache OpenOffice, 6, 68, 287 Apache Software Foundation (ASF), 9-10 API (application programming interface) calls hijacking in emulation, 185–187 identifying and classifying, 42-43 fuzzers, 206 APK (Android Package) format, 69-70 Apple Filing Protocol (AFP), 7, 295 application layer, TCP/IP, 50 applications, reverse engineering. See scripts under reverse engineering Apport, exploiting hardcoded path in, 57 - 59archinfo classes, 190 archive viewer utility, PyInstaller, 122-124 arguments, tainted, 22-24 ARM architecture, 146, 179 Art of War, The (Sun Tzu), 257 ASan (AddressSanitizer), 215-216, 248 asar tool, 112 ASF (Apache Software Foundation), 9-10 ASP (Active Server Pages) files, 152 assembly code, 137-138 ast module, Python, 74-77 ASTs. See abstract syntax trees asynchronous functions, 118 Atom Shell. See Electron framework Atom Shell Archive (ASAR) files, 110 - 112attacker-controlled sources, identifying, 26 - 29Attacking Network Protocols (Forshaw), xxv attack surfaces, mapping code to, xxiii, 39-40, 71 file formats, 66–67 custom fields, 70-71 directory-based, 69-70 TLV pattern, 67-68

attack surfaces (continued) internet. 40 web client vulnerabilities, 40-43 web server vulnerabilities, 43–50 local attack surfaces, 55-56 files in IPC, 56-61 named pipes, 63-65 other IPC methods, 65-66 sockets. 61-63 network protocols, 50–52 data structures, 52-53 procedures, 53-55 Auto-Fuzz feature, Fuzz Introspector, 255 autofuzz mode, Jazzer, 263-264, 266 automated code analysis, xxiii, 19-20, 73, 103-104 abstract syntax trees, 74-76 static code analysis tools, 77 CodeQL, 77-84 Semgrep, 84-87 variant analysis, 87-88 multi-repository, 101-103 single-repository, 87-100 @Autowired annotation, Spring framework, 46 availability in target selection, 8-9 Azure, SONiC build pipelines on, 30

B

backtrace command, GDB, 17-18, 237 - 238banned functions, 20 Baudrillard, Jean, 231 beautifiers, using on minified code, 119 - 120binaries, xxiii, 107-108, 143-144. See also hybrid analysis in reverse engineering; source and sink discovery in reverse engineering closed source, fuzzing, 258-262 compiled, applying code coverage for, 172-175 emulating with Qiling, 178-180 intermediate representations, 126 - 127Common Language Runtime assemblies, 127-131 Java bytecode, 131-137

machine code, 137-139 dynamically linked binaries, 140-141 packed binaries, 142-143 statically linked binaries, 139-140 stripped binaries, 141-142 managed memory, fuzzing, 262-263 with Go, 268–273 with Jazzer, 263-268 performing symbolic execution, 191-193 scripts, 109 Node.js Electron applications, 109 - 122Python applications, 122–126 targets in sink-to-source analysis, 30 types of, 108-109 Binary Template format, 010 Editor, 224-229 Binwalk, 146-147 black-box fuzzing, 5, 205. See also boofuzz with radamsa, 219-223 blocks in boofuzz, 209 body in file formats, 66 Böhme, Marcel, 207 boofuzz, 207-208 fuzzing MOTT protocol, 209-212 fuzzing MQTT PUBLISH packet, 212 - 214fuzzing NanoMQ, 214-219 MQTT protocol overview, 208-209 bootstrapped fuzzing, 223–229 branches, visualizing in code coverage, 177 - 178brokers, paying for zero days, 285 browser sandbox, 108 buffer overflow sink-to-source analysis on dhcp6relay, 20 building proof of concept, 34-37 confirming exploitability, 24-26 confirming reachable attack surfaces, 29 filtering for exploitable scenarios, 22 - 24identifying attacker-controlled sources, 26-29 selecting sinks, 20-22 testing exploits, 29-33

taint analysis example, 14–16 applying taint analysis, 18–20 triggering buffer overflow, 16–18 bug bounty programs, 284–287, 293 bugs, 2 vs. vulnerabilities, 3–4 building targets in sink-to-source analysis, 30–31 testing environments, 29–33 build instructions, LibreDWG, 250–252 bundled script-based executables, 122 bytecode, Python, 124–126

C

C programming language comparing machine code, assembly, and pseudocode, 137-138 dynamically linked binaries, 140-141 macros in, 90 C++ programming language, 225–226 C# programming language intermediate representations, 126-137 named pipes in, 64-65 C11 standard, 21 Calderon, Paulino, xxv canary strings, using to evaluate exploitability, 168-169 Canna server, 62 captive portals, 41 CefSharp API calls, 42–43 **CERT** (Computer Emergency Response Team), 292–293 CFG (control flow graph), 76 Chantzis, Fotios, xxv child format-related vulnerabilities, 70 Chromium Embedded Framework, 41-43 chunked data, TLV for, 67-68 CIA (confidentiality, integrity, and availability) triad, 2 CIFuzz GitHub action, 223 CIL (Common Intermediate Language), 108-109, 127-130 CISA (Cybersecurity and Infrastructure Security Agency), 284-285, 293 Claripy frontend, 190 Claroty Team82, 207 CLE component, angr, 190–191

CLI (command line interface) tool, GitHub. 49 client-side vulnerabilities. See web client vulnerabilities closed source binaries, fuzzing, 258-262 with Frida mode, 259-262 with QEMU mode, 258–259 cloud storage applications, monitoring events in, 166 CLR (Common Language Runtime) assemblies, 127-131 ClusterFuzz, 223, 246 CMPLOG mode, AFL++, 259 CNAs (CVE Numbering Authorities), 3, 292-294 CodeBrowser, Ghidra. See Ghidra CodeBrowser code coverage, 172 afl-cov coverage reports, 249 afl-plot tool graphs of, 276–277 applying for compiled binary analysis, 172 - 175visualizing with Lighthouse, 175–178 when emulating firmware with Qiling, 181-185 CodeQL, 77 multifile taint tracking example, 77 - 81multi-repository variant analysis, 101-103 vs. Semgrep, 85-86 VS Code extension, 81-84 code review, xxii–xxiii, 6–7, 104, 297. See also attack surfaces, mapping code to; automated code analysis; taint analysis command injection vulnerabilities static code analysis, 78-84 testing for possibility of, 157 ways to exploit, 160-161 command line interface (CLI) tool, GitHub, 49 Common Intermediate Language (CIL), 108-109, 127-130 Common Language Runtime (CLR) assemblies, 127-131 **Common Vulnerabilities and Exposures** (CVE) records, 3, 87, 292-294

Common Vulnerability Scoring System (CVSS), 2, 287-288 compilation, by interpreters, 109 compiled binary analysis, applying code coverage for, 172-175 compiled languages, 110 compiled Python files, 124 **Computer Emergency Response Team** (CERT), 292-293 conditional branches, visualizing in code coverage, 177-178 confidentiality, integrity, and availability (CIA) triad, 2 confused deputy problem, 58 connections in boofuzz, 209 CONNECT packet, MQTT protocol, 209-212, 215-216 consortiums as CNAs, 293 constraints, solving in symbolic analysis, 193 - 195containers images, 30, 36 Podman container management tool, 30 - 32privileged, 32 control flow graph (CFG), 76 controllers in MVC framework, 45-47 control packet types, MQTT protocol, 208 - 209coordinated vulnerability disclosure (CVD), 283–285 assigning CVE identifiers, 292-294 bug bounty programs, 285-287 disclosing vulnerabilities, 291-292 writing vulnerability reports, 287-291 coverage-guided fuzzing, xxiv, 7, 231-232, 255 advantages of, 232-234 with AFL++, 234-238 Fuzz Introspector, 250–252 analyzing function complexity, 253 - 255identifying fuzz blockers, 252-253 measuring coverage with afl-cov, 248-250 optimization techniques, 238 fuzzing in parallel, 248 minimizing seed corpus, 243-246

patching validation checks, 238-243 writing harnesses, 246-247 CRC (cyclic redundancy check) checksum, 224, 238-243 cross-site scripting (XSS) bug, 108 curl project, CVE for, 3-4 custom fields as attack surfaces, 70-71 custom uniform resource identifier (URI), 41 CVD. See coordinated vulnerability disclosure CVE Numbering Authorities (CNAs), 3, 292-294 CVE (Common Vulnerabilities and Exposures) records, 3, 87, 292–294 CVSS (Common Vulnerability Scoring System), 2, 287-288 Cybersecurity and Infrastructure Security Agency (CISA), 284-285, 293 cyclic redundancy check (CRC) checksum, 224, 238-243 cyclomatic complexity, 254

D

"d4rkn3ss" security researcher, 48 databases, CodeQL, 82-83, 103 data flow analysis, 78, 81, 86, 101–103 data flow graph (DFG), 76 data structures, network protocol, 52-53 dBase database file (DBF) format, 68, 207, 226-229 DbGate application, reverse engineering, 109–114 analyzing dangerous sink, 120–122 unpacking source maps, 114-119 using beautifiers on minified code, 119 - 120Debian package, 110-111 debuggers. See GNU Debugger decompiling with Ghidra, 148–155 intermediate representations, 130-131, 132-137 Python files, 124-126 Decompyle++, 124-125 DEF CON archives, xxv Deirmentzoglou, Evangelos, xxv

desktop applications, embedded browsers in, 41-42 DesktopLauncher.java file, Pixel Wheels, 135 DFG (data flow graph), 76 dhcp6relay server data structure discrepancies, 52-53 sink-to-source analysis on, 20 building proof of concept, 34-37 confirming exploitability, 24-26 confirming reachable attack surfaces, 29 filtering for exploitable scenarios, 22 - 24identifying attacker-controlled sources, 26-29 selecting sinks, 20-22 testing exploits, 29-33 DHCP for IPv6 protocol, 52–53 dictionaries, fuzzing with, 274-278 directory-based file formats, 67, 69-70 disassembling CLR assemblies, 128-129 with Ghidra, 148-155 machine code, 138 disclosing vulnerabilities, 291-292. See also coordinated vulnerability disclosure Docker, 252 Dockerfile, 32-33, 36, 250-251 Domain Name System (DNS), 41 domain validation function, Go fuzzing, 268-271 Dostoevsky, Fyodor, 145 dpkg-deb tool, 110 drawing (DWG) format, 235, 239-240, 243. See also LibreDWG, fuzzing drcov tool, 173-175 dumb fuzzers, 206, 215-216, 234 dump-ast feature, Semgrep, 98-99 dumping strings in static analysis, 147 - 148Dune (Herbert), 1 dwgread program AFL++ Frida mode, 260 fuzzing with AFL++, 235–238 patching validation checks, 238-243 dynamically linked binaries, 140-141, 149

dynamic analysis, 155, 170. See also hybrid analysis in reverse engineering analyzing library function calls, 158–161 instrumenting functions with Frida, 161–165 monitoring higher-level events, 165–167 tracing library and system calls, 156–158 dynamic symbol table, dumping, 149 DynamoRIO, 172–175

E

edge collisions, 235 EDR (endpoint detection and response) products, 296 Electron framework, 110 cross-site scripting bug in, 108 reverse engineering applications, 109 - 114analyzing dangerous sink, 120–122 unpacking source maps, 114-119 using beautifiers on minified code, 119 - 120**Electronic Frontier Foundation** (EFF), 285 ELF (Executable and Linkable Format), 108 Eliot, George, 171 embedded browsers in desktop applications, 41-42 emulation, 178 binding virtual paths, 187-189 of firmware with Qiling, 178–185 hijacking API calls, 185–187 encryption, 54 endpoint detection and response (EDR) products, 296 error handling, 54 errors, analyzing to evaluate exploitability, 167–169 Ethical Hacking (Graham), xxii European Union Agency for Cybersecurity, 284 eval sink, DbGate, 120-122 events, monitoring higher-level, 165-167 Excel, Microsoft, 40 Executable and Linkable Format (ELF), 108 executable binaries, 108–109, 111, 122 Expat C library, 88, 102 exploitable vulnerabilities evaluating, 167–169 in sink-to-source analysis confirming, 24–26 filtering exploitable scenarios, 22–24 testing exploits, 29–33 exploit development, 5, 16–18, 30 Express framework, 44–45, 81

F

Facebook Gameroom, 41–42 familiarity, role in target selection, 8 feedback loop fuzzing, 206-207 Fernflower decompiler, IntelliJ IDEA, 134 - 135ffmpeg application, 158–160, 167 fields, custom, as attack surfaces, 70-71 file2fuzz tool, 271–272 file command. 149 file formats as attack surfaces, 66-67 custom fields, 70-71 directory-based, 69-70 TLV pattern, 67–68 binaries in. 108 footers in, 66 file fuzzers, 206 files in IPC, 56–57 exploiting hardcoded path in Apport, 57 - 59exploiting race condition in Paramiko, 60-61 financial incentives for vulnerability disclosure, 285 firmware, emulating with Qiling, 178 - 185fixed header, MQTT packets, 208 flow control, 54 fmt package functions, 140 fork call, 189 forked processes, 185-186 FormatFuzzer, 224-229 Forshaw, James, xxv, 169

Fprintln function, 140-141 Fraser, Gordon, 207 FreshTomato firmware emulating with Oiling, 179–185 evaluating exploitability, 169 hijacking API calls, 185-186 static analysis of, 146 disassembling and decompiling with Ghidra, 148-155 dumping strings, 147–148 Frida mode, AFL++, 259-262 Frida toolkit instrumenting functions with, 161–165 Stalker code tracing engine, 173 frida-trace tool, 161-163 from bytes method, 52 frontend components, DbGate, 121 functions complexity of, analyzing, 253-255 hijacking, 185–187 instrumenting with Frida, 161-165 fuzz blockers, identifying, 252-253 fuzz coverage, analyzing with OSS-Fuzz, 222-223 fuzzers API, 206 based on information about targets, 205 dumb, 206, 215-216, 234 file, 206 FormatFuzzer, 224–229 generation-based, 205, 208, 223-224, 229 grammar-based, 205, 278-280 libFuzzer, 221-222, 267 mutation-based, 205 Peach Fuzzer, 207, 224 protocol, 206 smart, 206 Sulley fuzzer, 207 Fuzzilli, 281 fuzzing, xxii-xxiv, 7-8, 203-204, 257-258, 297. See also coverage-guided fuzzing with boofuzz, 207-208 fuzzing MQTT protocol, 209-212 fuzzing MQTT PUBLISH packet, 212-214

fuzzing NanoMQ, 214-219 MQTT protocol overview, 208-209 bootstrapped, 223-229 closed source binaries, 258-262 criteria and approaches for, 204-207 harnesses, 8, 232, 246-247, 267 managed memory binaries, 262-263 with Go, 268-273 with Jazzer, 263-268 mutation-based, 219-223 in parallel, 248 text-based formats, 273-274 with dictionaries, 274-278 with grammars, 278-280 with intermediate representations, 280 - 281Fuzzing Book, The (Zeller, Gopinath, Böhme, Fraser, and Holler), 207 "Fuzzing Like a Caveman" series (h0mbre), 207 Fuzz Introspector, 250–252 analyzing function complexity, 253 - 255Auto-Fuzz feature, 255 identifying fuzz blockers, 252-253

G

g++ compiler, 31–32 Galaxy Attack application, reverse engineering, 122-126 Gameroom, Facebook, 41–42 GCC (GNU Compiler Collection), 235 gcc command, 16–17, 137–138 GCC plug-in, AFL++, 235 GDB. See GNU Debugger gdb function, 36–37 generation approach to fuzzing, 205 generation-based fuzzers, 205, 208, 223-224, 229 getAtts member function, 97 getMacroFunction function, 120 getopt function, 183, 196–198 getRequest method, 264 Ghidra CodeBrowser disassembling and decompiling with, 148 - 155pseudocode in, 138, 140-141

stripped binaries, 142 visualizing code coverage in, 175–178, 181 - 185GitHub exploring projects on, 10 multi-repository variant analysis with, 103 OAuth flow in, 49 global taint tracking, 78-81 GNU Compiler Collection (GCC), 235 GNU Debugger (GDB) and AFL++ Frida mode, 260 buffer overflow, 17-18 when fuzzing with AFL++, 237-238 when minimizing seed corpus, 244-245 Google disclosure policies, 291–292 Project Zero, 294, 296 Gopinath, Rahul, 207 Go (Golang) programming language binaries packed, 142-143 statically linked, 139-140 stripped, 141-142 fuzzing feature, 262–263, 268–273 Graham, Daniel G., xxii grammar-based fuzzers, 205, 278-280 Grammar Mutator project, AFL++, 279-280 gray-box fuzzing, 5-6, 205 grep command, 24

H

h0mbre, 207 HackerOne, 291, 293–294 Hack In The Box archives, xxv handshaking, 54 hardcoded path, exploiting in Apport, 57–59 *Hardware Hacking Handbook, The* (van Woudenberg and O'Flynn), xxv harnesses, fuzzing, 8, 232, 246–247, 267–268 headers in file formats, 66 HTTP requests, 47 Herbert, Frank, 1 Holler, Christian, 207 hosted service as CNA, 293 HTML format custom fields in, 70-71 fuzzing, 273-276 HTTP client libraries, 43 OAuth flow, 49 requests, 47, 150-153 responses, 48 strings related to, 150–153 httpd binary in firmware binding virtual paths, 187–188 canary strings, 169 emulating firmware with Qiling, 178 - 185hijacking API calls, 185-186 static analysis, 147-148 symbolic execution, 194-195 hybrid analysis in reverse engineering, xxiv, 171–172, 199 code coverage, 172 for compiled binary analysis, 172 - 175visualizing with Lighthouse, 175-178 emulation, 178 binding virtual paths, 187-189 of firmware with Oiling, 178–185 hijacking API calls, 185-187 symbolic analysis, 189-191 performing symbolic execution, 191-193 solving constraints, 193–195 writing SimProcedures, 195–199

I

iCalendar (ICS) format, 70 ICCP (Inter-Chassis Communication Protocol), 50–51 iccpd server, 50–51 IDA Pro, 146 if statements, 97 IL Disassembler tool, Visual Studio, 128–130 ILSpy decompiler, 130–131 ImageMagick dynamic analysis of, 155 analyzing library function calls, 158–161

instrumenting functions with Frida, 161-165 monitoring higher-level events, 165 - 167tracing library and system calls, 156 - 158evaluating exploitability, 167–169 impact, role in target selection, 9 index.js file, 77-78, 83 Indutny, Fedor, 294 information leaks, exploitable, 22 infosec.exchange website, xxv input type fuzzing, 206 "insecurity through obscurity," 258 instrumentation modes, AFL++, 235-236 instrumenting functions with Frida, 161 - 165integer overflow vulnerability variants in Expat. See single-repository variant analysis IntelliJ IDEA Fernflower decompiler, 134-135 Intel Pin, 172–173 Inter-Chassis Communication Protocol (ICCP), 50-51 intermediate representations (IRs), 126 - 127Common Language Runtime assemblies, 127-131 fuzzing with, 280-281 Java bytecode, 131-137 internet as attack surface, 40 web client vulnerabilities, 40-43 web server vulnerabilities, 43 MVC architecture, 45-47 nontraditional web attack surfaces, 48 - 50unknown or unfamiliar frameworks, 47-48 web frameworks, 43-45 internet layer, TCP/IP, 50 interpreters, 109 inter-process communication (IPC) artifacts, examining, 169 local attack surfaces, 55-56 files in IPC, 56-61 other IPC methods, 65-66 temporary web servers for, 49 ip argument, ping function, 77–78

IPv6 addresses, 29, 31–32 IRs. *See* intermediate representations item geometry metric, AFL++, 237

J

JADX Android application decompiler, 69 - 70jailbreaking, 41 Java bytecode, 109, 131–137 fuzzing managed memory binaries written in, 262-268 Java Archive (JAR) files, 126, 131-132, 134 Java Development Kit (JDK), 132 Java virtual machine (JVM), 126 Java Naming and Directory Interface (JNDI), 6, 268 JavaScript grammars, 278-280 instrumenting functions with Frida, 161 - 165reverse engineering Node.js Electron applications, 109-114 analyzing dangerous sinks, 120 - 122unpacking source maps, 114–119 using beautifiers on minified code, 119 - 120Jazzer, fuzzing with, 263-268 JD-GUI decompiler, 134 js-beautify package, 119 JSON documents, Grammar Mutator grammar for, 279-280 js-yaml package, 9

K

Kaitai Struct format, 224 Kali Linux, installing Sasquatch on, 146 kmalloc function, 23

L

Laphroaig, Manul, 9 last new find metric, AFL++, 236 len argument, relay_relay_reply function, 27 L'Engle, Madeleine, 203 length in TLV pattern, 67–68 libevent library, 31

libFuzzer, 221-222, 267 libheif library, 21 liblzma software library, 6-7 libraries, shared, 108 library function calls analyzing, 158–161 tracing, 156-158 in web client functionality, 42 LibreDWG, fuzzing with AFL++, 235-238 with AFL++ Frida mode, 259–262 Fuzz Introspector and, 250–255 measuring coverage with afl-cov, 248 - 250minimizing seed corpus, 243-246 patching validation checks, 240-243 writing harnesses, 246-247 LibreOffice, 6 LibTIFF open source project, 259 libxls C library, fuzzing, 220–223 Lighthouse, visualizing code coverage with, 175-178 when emulating firmware with Qiling, 181–185 Light Keeper for Ghidra, 175–177, 181–185 hk> element, HTML format, 70–71 link layer, TCP/IP, 50 Link Layer Discovery Protocol (LLDP), 50 link time optimization (LTO), 235–236 Linux disassembling machine code in, 138 in6_addr struct type, 34 Kali, installing Sasquatch on, 146 libraries, installing, 31 Nimbuspwn collection of vulnerabilities, 61 open system call, 58 LiteDB Studio, reverse engineering, 127 - 131Liu Cixin, 39 LLVMFuzzerTestOneInput function, 222-223, 246-247 local attack surfaces, 55-56 files in IPC, 56-57 exploiting hardcoded paths in Apport, 57-59 exploiting race conditions in Paramiko, 60-61

local attack surfaces (continued) named pipes, 63–65 other IPC methods, 65–66 sockets, 61–63 local HTTP server, 49 local transport protocols, 55–56 lock files, 56–59 Log4j vulnerability, Apache, 6, 9, 268 log data, analyzing, 169 logging, monitoring, 166 Low Level Virtual Machine (LLVM) compiler, 235 lpSecurityAttributes argument, 64 LTO (link time optimization), 235–236 ltrace tool, 156–161

Μ

machine code, reverse engineering, 137-139. See also source and sink discovery in reverse engineering dynamically linked binaries, 140-141 packed binaries, 142-143 statically linked binaries, 139-140 stripped binaries, 141-142 Mach object (Mach-O) file format, 108 macro argument, compileMacroFunction function, 120-121 macros. 90 magick binary, using ltrace on, 158-161 main function fork call, 189 pseudocode for, 140 stripped binaries and, 142 visualizing code coverage, 181-182, 184 Makefile, dhcp6relay struct, 30-32 make function, 31 managed memory binaries, fuzzing, 262-263 with Go, 268-273 with Jazzer, 263-268 manifest files, 112-113, 126-127, 135 man-in-the-middle (MITM) attacks, 41,62 man ntohs command, 26 man recv command, 18 manual code review, 19-20, 24, 76 manual dictionary approach, AFL++, 275 - 278

map coverage metric, AFL++, 236 mapping code to attack surfaces. See attack surfaces, mapping code to markup-based formats, 66-67 memcpy function, 18–19, 21–28 memory corruption, 22 memory leak, 272 message buffer argument, parse dhcpv6 hdr function, 29 Message Queuing Telemetry Transport (MQTT) protocol, 208-209 fuzzing, 209–212 PUBLISH packet, fuzzing, 212–214 metadata in intermediate representations, 126-128 metavariables, 85, 99-100 metrics, to gauge impact, 9 Microsoft. See also Windows banned functions, 20 Bug Bounty Program, 286 Excel, 40 .NET framework, 42, 126-131 Office, 67, 70 Microsoft 365 Defender Research Team. 61 Middlemarch (Eliot), 171 middlemen, paying for zero days, 285 minified code reversing, 114-119 using beautifiers on, 119-120 MIPS architecture, 146–147, 179 Mirosh, Oleksandr, 6 misconfigurations in named pipes, 64 - 65MITM (man-in-the-middle) attacks, 41, 62 MITRE Corporation, 3, 294 mkfifo API call, Unix, 65 mode field, Semgrep, 84-85 model-view-controller (MVC) architecture, 45–47 _mode parameter, FUN_0001abc0 function, 154 MQTT protocol. See Message Queuing Telemetry Transport protocol msg argument, relay relay reply function, 27-28 multifile taint tracking example, 77-81 multi-repository variant analysis, 101 - 103

Muñoz, Alvaro, 6 Murakami, Haruki, 13 mutation-based fuzzers, 205 bootstrapping, 223–224 vs. generation-based fuzzers, 208 radamsa, 219–223 MVC (model-view-controller) architecture, 45–47

N

named pipes, 63-65 NanoMQ, fuzzing, 214-219 NanoNNG files, 214-215 national CVD policies, 284 National Institute of Standards and Technology (NIST), 2 native applications, 40 NConvert, 258-259 Netatalk, 7, 295 .NET framework, 42 binaries, reverse engineering, 126 - 131NETGEAR Nighthawk R6700v3 router, 48 network attached storage (NAS) devices, 7, 295 network events, monitoring, 166 network protocols as attack surfaces, 50-52 data structures, 52-53 procedures, 53-55 overlap with local transport protocols, 55 - 56Nimbuspwn collection of vulnerabilities, 61 Nintendo Switch, 41 NIST (National Institute of Standards and Technology), 2 nMaxInstances argument, CreateNamedPipe function, 63 node-ip package, 294 Node.js runtime environment, 43-44, 108. See also Electron framework nontraditional web attack surfaces, 48 - 50npm registry, 9 ntohs function, 26–27 null dereference, 22

0

OAuth flow, 49 objdump command, 138-139 octet strings, 53 OffensiveCon archives, xxv Office, Microsoft, 67, 70 O'Flynn, Colin, xxv OleViewDotNet tool, 169 1Q84 (Murakami), 13 **Open Design Alliance**, 240 OpenOffice, Apache, 6, 68, 287 open source CNAs, 293 open source code dependencies, 6-7 open source software, 87 open system call, Linux, 58 option length parameter, 35 option->option_length parameter, 26 - 27organizations, securing with vulnerability research, 294-296 Ormandy, Tavis, 296 **OSS-Fuzz** analyzing fuzz coverage with, 222-223 Fuzz Introspector within, 250-252 out-of-bounds read vulnerability, 53

P

package.json files, DbGate, 112-113 PACKED attribute, struct definitions, 34 - 35packed binaries, reverse engineering, 142 - 143packet types, MQTT protocol, 208-209 pack function, 34-35 Padioleau, Yoann, 84 parallel fuzzing, 248 parameters, HTTP requests, 47 Paramiko, exploiting race condition in, 60 - 61parse dhcpv6 hdr function, 28-29 parse_dhcpv6_opt function, 26, 34 parse dhcpv6 relay function, 34 patches insufficient, 88 root cause analysis, 88-90 validation checks, 238-243 path explosion, 18-19 pattern-either operator, 94, 100

pattern-inside operator, 99–100 pattern operator, 94, 100 patterns Semgrep, 84-86 variant pattern matching, 92–100 payload, MQTT packets, 208 PDU (protocol data unit), 52–55 PDUHeaderTags class, 52 Peach Fuzzer, 207, 224 Peach Pit format, 224 PE-Bear tool, 128 PE (Portable Executable) file format, 108, 127-128 penetration testing, 5-6 persistent mode, AFL++, 246-247 PF INET argument, socket function, 51 Pi, Pavel, 146 ping function, 77–78 Pixel Wheels, reverse engineering, 134-137 PoC. See proof of concept PoC || GTFO (Laphroaig), 9 Podman container management tool, 30 - 32polyfills, 117 popen function dynamic analysis, 161, 163-165 evaluating exploitability, 167–168 static analysis, 149-151, 153-155 Portable Executable (PE) file format, 108, 127-128 Portable Network Graphics (PNG) format bootstrapped fuzzing, 223-226 coverage-guided fuzzing, 232–234 TLV pattern in, 67-68 Postgres project, CVE for, 3-4 Practical Binary Analysis (Andriesse), xxv Practical IoT Hacking (Chantzis, Stais, Calderon, Deirmentzoglou, and Woods), xxv prepare socket function, 29 primitives in boofuzz, 209, 211-212 printf function, 138 Println function, 140 private keys, 60-61 privileged containers, 32 procedures of network protocols as attack surfaces, 53-55

processes as local attack surfaces, 55 process monitors, 218-219 Procyon decompiler, 134 product security assessments, 296 project owners, accessibility of, 9 projects, exploring, 10 Project Zero, Google, 294, 296 proof of concept (PoC), 5 in root cause analysis, 92 sink-to-source analysis strategy, 30, 34-37 in vulnerability reports, 288-289 propagation, taint, 13, 19 protocol data unit (PDU), 52–55 protocol fuzzers, 206 pseudocode vs. machine code, 137-138 visualizing code coverage, 177, 182-183, 184-185 pspy tool, 166 PUBLISH packet, MQTT protocol, 208, 212-214, 215 puts function, 138, 141 Pwn2Own Tokyo 2019, 48 pyi-archive viewer utility, 122-124 PyInstaller executables, 122-123, 126 Python ast module, 74-77 bytecode, 124, 126 exploiting race condition in Paramiko, 60-61 instrumenting functions with Frida, 163 - 165reverse engineering, 109, 122-126 PyVEX bindings, 190

Q

QEMU mode, AFL++, fuzzing with, 258–259 Qiling, emulating firmware with, 178–185, 188–189 QL programming language, 81 query-oriented syntax. *See* CodeQL

R

race condition, exploiting in Paramiko, 60–61 radamsa, mutation-based fuzzing with, 219–223 radius_copy_pw function, 21–22 rand standard library function, 195-196 RDS (Remote Desktop Services), Windows, 63-64 reachability analysis, 76 reachable attack surfaces, confirming, 29 read byte function, 216-217 read data section function, 245 REALLOC macro, 90, 93-101 realloc standard library function, 89-90, 101 Real-World Bug Hunting (Yaworski), xxii recommendations in vulnerability reports, 291 recv from function, 29 recv function, 18-19 regex, 24, 75, 78 regression, 88 rel attribute, HTML format, 70-71 relay_relay_reply function, 24-27, 37 release build, LibreDWG, 245 remote command injection, 83-84 Remote Desktop Services (RDS), Windows, 63-64 reports, vulnerability. See vulnerability reports, writing reproduction steps in vulnerability reports, 288-289 @RequestMapping annotation, Spring framework, 46 requests in boofuzz, 209 HTTP, 47, 150-153 Requests for Comments (RFCs), 51, 280 researcher CNAs, 293 resources in .NET binaries, 127 responses, HTTP, 150-153 responsible disclosure. See coordinated vulnerability disclosure --retry-delay command line option, curl, 3-4 reverse engineering, xxii-xxiv, 7, 104, 107–109. See also hybrid analysis in reverse engineering; source and sink discovery in reverse engineering intermediate representations, 126-127 Common Language Runtime assemblies, 127-131 Java bytecode, 131–137

machine code, 137-139 dynamically linked binaries, 140 - 141packed binaries, 142-143 statically linked binaries, 139 - 140stripped binaries, 141-142 scripts, 109 Node.js Electron applications, 109 - 122Python applications, 122–126 RFC 9116, 291–292 RFCs. See Requests for Comments Rollup, 114-115 root cause analysis of, 88-92 in vulnerability reports, 289-291 route strings in web frameworks, 46 runButtonClick function, 152-153 runMacroOnChangeSet function, 120-121 runtime behavior, dynamic analysis of. See dynamic analysis

S

sanitizers vs. canary strings, 168-169 and fuzzing performance, 235 Jazzer, 264–268 in taint analysis, 19 Sasquatch tool, 146 satisfiability modulo theories (SMT) problems, 190 scripting languages, 109 scripts AFL++ Frida mode, 261–262 in Frida, 161-165 reverse engineering, 109 scripts (continued) Node.js Electron applications, 109 - 122Python application, 122–126 SDKs (software development kits), 42 Secure Shell (SSH) service, 7 security assessments, product, 296 security boundaries in network protocols, 53 "security by obscurity," 7

security misconfigurations in named pipes, 64-65 security.txt file format, 291-292 sed tool. 32 seed corpus, 205 adding in OSS-Fuzz, 251 minimizing for fuzzing, 243–246 selling zero days, 285 semantic parsing, 274. See also text-based formats, fuzzing Semgrep code analysis tool OSS, 86-87 Playground, 86, 93–100 scanning thousands of repositories with, 102 static analysis with, 84-87 SendMessage function, Windows, 66 sentinel validation, 240 server callback function, 28-29 server-side request forgery (SSRF), 263-268 server-side vulnerabilities. See web server vulnerabilities session management, 54 session termination, 54 sessions in boofuzz, 209 set api function, 187 setupLogging function, 135–136 Shah, Shubham (shubs), xix-xx shared libraries, 108 shorts, 52 SimProcedures, writing, 195–199 simulated program state (SimState), 193 simulation manager, 193 single-repository variant analysis, 87-88 root cause analysis, 88-92 variant pattern matching, 92-100 sinks. See also source and sink discovery in reverse engineering; taint analysis analyzing when reverse engineering, 120 - 122identifying in taint analysis, 18 selecting in sink-to-source analysis, 20 - 22in variant pattern matching, 93 sink-to-source analysis strategy, 20 building proof of concept, 34-37

confirming exploitability, 24-26 confirming reachable attack surfaces, 29 filtering for exploitable scenarios, 22 - 24identifying attacker-controlled sources, 26-29 when reverse engineering, 120–122 selecting sinks, 20-22 testing exploits, 29-33 smart devices, attack surfaces on, 48 smart fuzzers, 206 SMT (satisfiability modulo theories) problems, 190 snappy Golang library, 271–272 socket function, 51 socket library, 35 sockets as local attack surfaces, 61-63 software development kits (SDKs), 42 software development life cycle, vulnerability research in, 295 solving constraints in symbolic analysis, 193-195 SONiC (Software for Open Networking in the Cloud) build process, 30-31 network protocol attack surface, 50 sink-to-source analysis, 20-22 Switch State Service, 52 sonic-snmpagent PDU procedure code, 52, 54-55 sonic-swss-common library, 31 Soo, Jacob, xvii-xviii source and sink analysis. See taint analysis source and sink discovery in reverse engineering, xxiv, 145-146, 169-170 dynamic analysis, 155 analyzing library function calls, 158 - 161instrumenting functions with Frida, 161-165 monitoring higher-level events, 165 - 167tracing library and system calls, 156 - 158evaluating exploitability, 167-169

static analysis, 146-147 disassembling and decompiling with Ghidra, 148–155 dumping strings, 147-148 source code. See also code review accessing with source maps, 114-119 for book, xxiv-xxv closed source targets, 258 intermediate representations, 124 reverse engineering Python applications, 123-126 source-map library, 115–116 source maps, unpacking, 114-119 sources. See also sink-to-source analysis strategy; source and sink discovery in reverse engineering attacker-controlled sources, identifying, 26-29 in taint analysis, 18-19 spaceraccoon.dev blog, xxv SPOR (Beard), 107 Spring MVC web framework, 45-46 sprintf function, 20 SSH (Secure Shell) service, 7 s_size primitive, boofuzz, 211 SSRF (server-side request forgery), 263-268 stability metric, AFL++, 237 stack canary, 16 stage process metric, AFL++, 237 Stais, Ioannis, xxv Stalker code tracing engine, Frida, 173 state management, 54 statically linked binaries, reverse engineering, 139–140 static analysis, 77. See also hybrid analysis in reverse engineering with CodeQL, 77 multifile taint tracking example, 77 - 81VS Code extension, 81-84 in reverse engineering, 146-147, 170 disassembling and decompiling with Ghidra, 148-155 dumping strings, 147-148 with Semgrep, 84-87 Stenberg, Daniel, 3 storeAtts function, 96, 98

strcat function, 20 strcpy function, 20 strided copy function, 21 strings canary, using to evaluate exploitability, 168-169 dumping in static analysis, 147–148 HTTP-related, 150-153 strings function, 134, 147-148 stripped binaries, 141-142, 149 strncat function, 20 strong name signature in .NET binaries, 127 - 128subagent processing, 53-55 Sulley fuzzer, 207 summaries in vulnerability reports, 287 - 288Sun Tzu, 257 .svelte files, 121 swap files, 57 Swift, Graham, 283 Switch, Nintendo, 41 Switch State Service (SWSS), SONiC, 52 s word primitive, boofuzz, 211 symbolic analysis, 189–191 performing symbolic execution, 191 - 193solving constraints, 193-195 writing SimProcedures, 195–199 symbolic link (symlink) attacks, 58-59 symbol table, dumping, 139, 149 Symbol Tree panel, Ghidra CodeBrowser, 148-150 Synamtec Endpoint Protection, 296 syntactic parsing, 274. See also text-based formats, fuzzing system calls, tracing in dynamic analysis, 156-158, 161 system events, monitoring, 166 system function, 149–150, 157–158, 162 - 163

T

taint analysis, xxiii, 13–14 buffer overflow example, 14–16 applying taint analysis, 18–20 triggering buffer overflow, 16–18

taint analysis (continued) vs. fuzzing, 204 sink-to-source analysis strategy, 20 building proof of concept, 34-37 confirming exploitability, 24-26 confirming reachable attack surfaces, 29 filtering for exploitable scenarios, 22 - 24identifying attacker-controlled sources, 26-29 selecting sinks, 20-22 testing exploits, 29-33 taint propagation, 13, 19 taint tracking multifile, 77-81 multi-repository variant analysis, 101-103 targets in boofuzz. 209 fuzzers based on information about, 205 selecting for vulnerability research, 8 - 10TCP/IP (Transmission Control Protocol/Internet Protocol) model. 50 Team82, Claroty, 207 temporary web servers, 49 testing in sink-to-source analysis strategy, 29 - 33in vulnerability research, 9 text-based formats, fuzzing, 273-274 with dictionaries, 274-278 with grammars, 278–280 with intermediate representations, 280-281 third parties, paying for zero days, 285 ThrowMagickException function, 168 thunk functions, 141, 150 TIFF files, 258-259 token-based dictionaries, 275-278 Transmission Control Protocol/Internet Protocol (TCP/IP) model, 50 transpiled JavaScript in DbGate, 116-118 transport layer, TCP/IP, 50 Trending page, GitHub, 10 TRX file format, 146

type declarations, 117 type metadata in .NET binaries, 127 type–length–value (TLV) pattern, 67–68 TypeScript in DbGate, 116–118

U

Ubuntu, privilege escalation vulnerability in, 57–59 uint16_t variables, 26 UltimatePacker for eXecutables (UPX), 142–143 undiscovered complexity, 254–255 Unicorn emulator framework, 178–179 uniform resource identifiers (URIs), 41, 47 Unix, creating named pipes in, 65 Unix domain sockets (UDSs), 62–63 unpacking source maps, 114–119 unsigned integer types, 91 *utdbf* program, 226–229 *utils.js* file, 77–78, 81

V

V8 engines, 110 validation checks, patching, 238-243 validators in taint analysis, 19 value in TLV patterns, 67 van Woudenberg, Jasper, xxv variable header, MQTT packets, 208 variant analysis, 87-88 multi-repository, 101-103 single-repository, 87-88 root cause analysis, 88-92 variant pattern matching, 92-100 vendors as CNAs, 293 paying for zero days, 285 views in MVC framework, 45 Vim editor, 56–57 virtual local area network (VLAN), 32 virtual machine runtime environments, 126 virtual paths, binding in emulation, 187 - 189virtual private network (VPN), 296 Visual Studio CodeQL extension for, 81-84, 102 - 103IL Disassembler tool, 128-130

vi test command, 56-57 vsprintf function, 20 vulnerabilities, 2-4 bugs vs., 3-4 CVE records, 3 disclosing, 291-292 vulnerability reports, writing, 287-288 recommendations, 291 reproduction steps, 288-289 root cause, 289-291 vulnerability research, 1-2, 4-5, 10. 297. See also coordinated vulnerability disclosure; zero-day vulnerabilities disciplines and techniques in, 6-8 Jacob Soo on, xvii-xviii vs. penetration testing, 5-6 securing organizations with, 294-296 Shubham Shah on, xix-xx target selection, 8-10

W

w3m web browser, 275 WeasyPrint HTML-to-PDF conversion engine, 70-71 web applications, 40 WebAssembly binary code, 108-109 web client vulnerabilities, 40-41 attack vectors, 41-42 identification and classification, 42 - 43web frameworks as attack surfaces, 43 - 45MVC architecture, 45-47 unknown or unfamiliar frameworks, 47 - 48WebKit, 41 Webpack, 114, 119-120 web server vulnerabilities, 43 nontraditional web attack surfaces, 48 - 50

web frameworks, 43–45 white-box fuzzing, 5, 8, 205 Windows. *See also* Microsoft named pipe filesystem, 63–64 Remote Desktop Services, 63–64 SendMessage function, 66 Wireshark, 211–212 Woods, Beau, xxv world-readable and -writable named pipes, 64–65 wrapper functions, 21–22 *Wrinkle in Time, A* (L'Engle), 203 _write_private_key_file method, Paramiko, 60

X

X key in IDA Pro, 146 XML, 67 directory-based formats, 69–70 Expat integer overflow vulnerability variants, 88–100 XML External Entity (XXE) injection, 69–70 XmlGetAttributes macro, 97 XSS (cross-site scripting) bug, 108

Y

YAML language, 84–85 Yaworski, Peter, xxii

Z

Zalewski, Michal, 233–234 Zeller, Andreas, 207 Zero Day Initiative (ZDI), xxv, 91, 286 zero-day vulnerabilities, xxi–xxii. *See also* coordinated vulnerability disclosure; vulnerability research 010 Editor Binary Template format, 224–229

ZIP archive format, 67, 69–70