

Index

- 19 *Deadly Sins* . . . , 292,
296–297
and seven kingdoms, 296
- A**
- Abstract syntax tree, 111
- Abuse cases, 205–222
anti-requirements,
213–215
attack models, 216–217
attack patterns, 218–221
attacker motivation, 208
benefits of, 222
constructive/destructive
nature, 90
creating, 211–212
description, 88
development team,
210–211
example, 217, 222
flyover, 88
history of, 208–209
identifying and docu-
menting threats, 213
overview, 205–209
process diagram, 214
software developers and
information security
practitioners,
225–226
touchpoint process,
213–217
- Academic software security,
98
- Access control policies,
modeling, 155
- Adversarial security testing,
193
- Aitel, Dave, 180
- ALE (Annualized Loss
Expectancy), 152
- Ambiguity analysis, in
architectural risk
analysis, 165–167
- Anti-requirements,
213–215
- API Abuse vulnerability
kingdom
description, 279–280
example, 290–292
phyla, 285
- APISPY32, 181
- AST. *See* Abstract syntax
tree
- Application security
badness-ometers, 22–23
limitations of, 20–21
versus software security,
20–21
testing tools, 22–23,
230
- Applied risk management
pillar, 26–27
- Arc injection attacks, 191
- Architectural risk analysis,
139–170
access control policies,
modeling, 155
ad hoc, 161
- assets, 144
bugs, 17, 145
checklists, 169
commercial, 142
common themes,
140–142
constructive/destructive
nature, 90
countermeasures, 145
description, 86
flaws, 17, 145
flyover, 86
forest-level view,
148–152, 156–157
getting started, 169
impact, 145–146
knowledge requirements,
147–148
major activities, 141
necessity of, 170
.NET security model
overview, 149–150
one page design
overview, 148–152,
156–157
practical applications,
142
probability, 146
process diagram, 162
ambiguity analysis,
165–167
attack resistance analy-
sis, 163–165
weakness analysis,
167–169

- Architectural risk analysis
(*cont.*)
risk analysis, definition, 140
risk calculation
 impact, 155–156
 modern model, 154–161
 traditional model, 152–154
risk management, definition, 140
risks, 144–145
in the RMF, 143–144
ROI (return on investment), 153
safeguards, 145
software developers and information security practitioners, 227–228
standards-based, 142
STRIDE, 147
terminology, 144–146
threat modeling *versus* risk analysis, 146–147
threats, 145
touchpoint process
 ad hoc approach, 161
 ambiguity analysis, 165–167
 attack resistance analysis, 163–165
 critical steps, 162
 examples of flaws, 163–164, 166–167, 168–169
 exploit graphs, 164–165
 process diagram, 162
 weakness analysis, 167–169
 vulnerabilities, 145
Anderson, Ross, 188
Arciniegas, Fabio, 215–216
Arkin, Brad, 166
Array out of bounds, 114, 118
- Articles. *See* Bibliography.
Artifacts, software, 28, 34, 393
ASP.NET Misconfiguration phylum, 289
Assets, definition, 144
ASSET, 142
Assume nothing, 210–211
Attack classes, 294–295
Attack models, 216–217
Attack patterns
 knowledge catalog, 264, 266
 list of, 218–221
 taxonomy of, 277
Attack resistance analysis, in architectural risk analysis, 163–165
Attacker motivation, 208
Attackers' tools, 180–181, 201
 APISPY32, 181
 breakpoint setters, 181
 control flow, 181
 coverage, 181
 decompilers, 181
 disassemblers, 181
 fault injectors, 180
 rootkits, 181–182
 shell code, 181
Auditing open source applications, tutorial, 342–344
Authentication phylum, 285, 290–292
Automation
 Cigital Workbench, 76–78
 risk-based security testing, 196
- B**
Badness-ometers, 22–23
Bellovin, Steve, 3
Berkman, Ariel, 177–178
Bernstein, D. J., 177–178
Best practices. *See* Touchpoints
- Bibliography
 19 *Deadly Sins* . . . , 292
 academic literature, on bugs, 293–295
 citations in this book, 300–312
 government and standards publications, 312–313
 “OWASP Top Ten . . . Vulnerabilities,” 292
 required reading, 299–300
 RISOS project, 293–294
 software security literature, 299–318
 top five publications, 299–300
 vulnerabilities, 293–294
Binary analysis, for security, 106–108
Bishop, Matt, 111, 112
Black box testing, 194
Black hat activities, touchpoints, 89–91, 172
BLAST tool, 123
Bob, 17
Books and publications. *See* Bibliography.
BOON tool, 114, 118
Breakpoint setters, 181
Buffer overflow, described, 15
Buffer Overflow phylum, 283
Bugs. *See also* Defects; Taxonomy of coding errors.
 architectural risk analysis, 145
 code review. *See* Code review.
 buffer overflow, 15–16
 causes. *See* Causes of problems.
 definition, 14
 examples, 17

- versus* flaws, 18–19, 191
 - implementation, 106–108
 - more lines, more bugs, 10–13
 - parade, 259
 - BugScan, 107
 - BugTraq, 168, 173, 259
 - Building a software security program. *See* Enterprise software security.
 - Business context, RME, 43, 49–50
 - Business-level security. *See* Enterprise software security.
- C**
- Can'ts and won'ts, 211–212, 222
 - CANVAS tool, 180
 - Carrying out fixes and validation, RME, 43, 73
 - Catch NullPointerException Exception phylum, 287
 - Causes of problems
 - complexity, 8–10
 - connectivity, 6–7
 - design flaws, 139
 - extensibility, 7–8
 - legacy applications, 6–7
 - mobile code, 7–8
 - “more lines, more bugs,” 10–11
 - SOA (Service Oriented Architecture), 6–7
 - software vulnerability, 4–5
 - Web Services, 6–7
 - Cenzic, 180
 - CERT incidents, 3–5
 - Champions, for best practice adoption, 244, 247
 - Change maturity path, 243, 246
 - Checklists, architectural risk analysis, 169. *See also* STRIDE.
 - Chess, Brian, 133
 - Cheswick, Bill, 3
 - Cigital, 39, 113, 127, 143, 166
 - Cigital Workbench, 76–78
 - COBIT, 142
 - CISSP, 225
 - Code review manual, 106
 - Code Quality vulnerability kingdom, 281, 287–288
 - Code review, software developers and information security practitioners, 229–230, 231
 - Code review, tools. *See also* Tools.
 - array out of bounds, 114, 118
 - binary analysis, 106–108
 - BLAST tool, 123
 - BOON tool, 114, 118
 - code scanners, 107, 109–110
 - commercial tool vendors. *See also* Fortify.
 - code source analyzers, 124–125
 - Coverity, 123
 - Fortify, 123
 - Ounce Labs, 123
 - Secure Software, 123
 - tool characteristics, 125–127
 - tool problems, 127
 - constructive/destructive nature, 90
 - consultants as mentors, 99
 - CQual tool, 118
 - description, 86
 - Eau Claire tool, 122
 - ESP tool, 123
 - false negatives/positives, 109
 - FindBugs tool, 123
 - Flyover, 86
 - global analysis, 111
 - good *versus* perfect, 108–109
 - Hoglund's BugScan, 107
 - human evaluation, 108–109
 - implementation bugs, 106–108
 - integer range analysis, 114, 118
 - ITS4
 - code scanner, 109–110
 - rules, history, 112–114
 - kernel vulnerabilities, 118, 122
 - local analysis, 111
 - module-level analysis, 111
 - MOPS tool, 122
 - RATS code scanner, 109–110
 - rules
 - coverage, 112–114
 - example, 119–122
 - ITS4, 112–114
 - schema, 115–118
 - safety property violations, 122
 - SLAM tool, 123
 - specification checking, 122
 - Splint tool, 122–123
 - static code analysis
 - example, 135–137
 - history, 110–114
 - taint analysis, 118
 - TOCTOU (time-of-check-time-of-use), 111
 - touchpoint process, 135–137
 - xg++ tool, 118, 122
- Command Injection phylum, 283

- Commercial architectural risk analysis, 142
- Commercial off-the-shelf software (COTS), 251–256
- Commercial source code analysis tool vendors
- Coverity, 123
 - Fortify, 123
 - Ounce Labs, 123
 - Secure Software, 123
 - source code analyzers, 124–125
 - tool characteristics, 125–127
 - tool problems, 127
- Comparing Classes by Name phylum, 288
- comp.risks, 168
- Complexity
- linux/open source code base growth, 10
 - major operating systems, 10
 - metrics, 10–11
 - “more lines, more bugs,” 10–11
 - trinity of trouble, 8–13
 - Windows code base growth, 8–9
- Connectivity, trinity of trouble, 6–7.
- Constructive activities, touchpoints, 89–91
- Control flow tools, 181
- COTS (commercial off-the-shelf software), 167, 251–256
- Countermeasures, for risk mitigation, 145
- Coverage tools, 181
- Coverity, 123
- Cross site scripting, 201
- CQual tool, 118
- Creating Debug Binary phylum, 289
- Cross-Site Scripting phylum, 283
- Cultural change. *See* Enterprise software security.
- CVE, 277
- D**
- Danahy, Jack, 207
- Dangerous Functions phylum, 285
- Data Leaking Between Users phylum, 288
- Deadlock phylum, 286
- Decompilers, 181
- Defects. *See also* Bugs; Flaws; Taxonomy of coding errors.
- architectural risk analysis, *See* Architectural risk analysis
 - causes. *See* Causes of problems.
 - definition, 14
 - design-level vulnerabilities, 19–20
 - error detection, 19
 - failure recovery, 19
 - fifty/fifty, 139
 - midrange vulnerabilities, 19
 - range of, 18–19
- Defining the risk
- mitigation strategy, 45, 69–71
- Department of Homeland Security portal, 264–274
- Dependencies. *See* Weakness analysis.
- Deployment and operations, software developers and information security practitioners, 231–232
- Design-level vulnerabilities, 19–20, 191–192. *See also* Flaws.
- Destructive activities, touchpoints, 89–91
- Developer Desktop, 130–131
- DHS portal. *See* Department of Homeland Security portal.
- Diagnostic knowledge, 264
- Diebold insecurity, 161
- Dilger, Mike, 111, 112
- Directory Restriction phylum, 285
- Disassemblers, 181
- Double Free phylum, 287
- Duplicate Validation Forms phylum, 284
- E**
- Eau Claire tool, 122
- Electronic voting security, 159–161
- Empty Catch Block phylum, 287
- Empty Password in Configuration File phylum, 286
- Encapsulation vulnerability kingdom, 281–282, 288–289
- Engineer gone bad, 208
- Enterprise information architecture, 253–256
- Enterprise software security, 239–257
- basic steps, 241–242
 - business climate, 240–242
 - champions, for best practices, 244, 247
 - change maturity path, 243, 246
 - common pitfalls, 244–245
 - continuous improvement, 250–251
- COTS (commercial off-the-shelf software), 251–256
- cultural change, 242–243, 246

- enterprise information architecture, 253–256
- existing applications, 251–256
- general framework, 246–247
- improvement program, 246–247
- lack of high-level commitment, 245
- management without measurement, 244–245
- metrics program, 247–250
- over-reliance on late-life-cycle testing, 244
- SDL (Secure Development Lifecycle), 239–240, 256–257
- training without assessment, 245
- Environment vulnerability kingdom, 282, 289
- Erroneous validate()
 - Method phylum, 284
- Error detection, 19
- Error Handling vulnerability kingdom, 281, 287
- ESP tool, 123
- Examples
 - abuse cases, 217, 222
 - Adobe Reader, 282
 - Diebold voting machines, 159–161
 - flaws found in architectural risk analysis, 163–164, 166–167, 168–169
 - Java card, 195–200
 - KillerAppCo's iWare. *See* RMF (risk management framework), example.
 - malicious PDFs, 282
 - password security, 177–178
 - penetration testing, 176–177, 177–178
 - risk-based security testing, 195–200
 - smart cards, 195–200
 - Smurfware exercise, 385–391
 - software developers and information security practitioners, 234–235
- Exception Handling phylum, 285
- Exploits
 - graphs, 164–165
 - knowledge catalog, 264, 268
- Extensibility, trinity of trouble, 7–8
- External analysis, description, 88–89, 211
- eXtreme programming, 202
- F**
- Failure recovery, 19
- Failure to Begin a New Session . . . phylum, 286
- False negatives/positives, in source code analysis tools, 109
- Fault injection tools, 35, 180
- Features, security, not good enough, 209, 229
- Feel good security, 173
- File Access Race Condition phylum, 286
- FindBugs tool, 123
- Firewalls, 189–190
- Fixes and validation, RMF, 43, 73
- Flawfinder code scanner, 109
- Flaws. *See also* Defects.
 - architectural risk analysis, 145
 - versus* bugs, 18–19, 191
 - causes. *See* Causes of problems.
 - definition, 14, 16, 17, 18
 - examples, 17, 163–164, 166–167, 168–169
 - Microsoft Bob program, 17
- Forest-level view, architectural risk analysis, 148–152, 156–157, 203
- Form Field Without Validator phylum, 284
- Format String phylum, 283
- Fortify Source Code Analysis Suite
 - components
 - Audit Workbench, 129–130
 - Developer Desktop, 130–131
 - knowledge base, 132–134
 - Rules Builder, 129–130
 - Software Security Manager, 132
 - Source Code Analysis Engine, 128–129
 - Source Coding Rulepacks, 129
 - demonstration version, 134–135
 - tutorials
 - Audit Workbench, 324–326, 339–342
 - auditing code manually, 326–328
 - auditing open source applications, 342–344
 - automated build processes, 335–339
 - command line arguments, 332–333
 - ensuring a working environment, 328–329
 - raw analysis results, 333–335

Fortify Source Code Analysis Suite, tutorials
(*cont.*)
source code analysis engine, 329–332
Functional testing, 193
Fuzzing, 179

G

Gates memo, 29–34. *See also* Microsoft, high-level commitment.
Geer, Dan, 10–13, 180, 208
Foreword by, xix–xxi
getConnection() method phylum, 285
GLBA, 155
Global analysis, 111
Glorified grep, 110
Glossary of terms, 393
Good *versus* perfect, 108–109
GP (Global Platform), for Java Card smart cards, 196
Grep as scanner, 110
Guidelines, knowledge catalog, 263–264, 265

H

Hackers, reformed, as penetration testers, 173
Hacker in a box, 230
Hailstorm, 180
Halting problem, 109
Hard-Coded Passwords phylum, 286
Heap Inspection phylum, 285
HIPPA, 155, 210
Historical knowledge, 264
Historical risks knowledge catalog, 264, 267
Hoglund, Greg, 180
Hoglund's BugScan, 107
Holodeck tool, 180

HTTP Response Splitting phylum, 283
Human evaluation, as essential for risk analysis, 108–109

I

IBM, 74
Identifying business and technical risks, RMF, 43–44, 50–63
Illegal Pointer Value phylum, 283
Impact, architectural risk analysis calculation, 145–146
Implementation bugs. *See* Bugs.
Inconsistent Implementations phylum, 287
Information architecture. *See* Enterprise information architecture.
Information security practitioners. *See* Security professionals; Software developers and information security practitioners.
Input Validation and Representation kingdom, 279, 283–285
Insecure Compiler Optimization phylum, 289
Insecure Randomness phylum, 285
Insecure Temporary File phylum, 286
Inside→out approach, 190
Integer Overflow phylum, 283
Integer range analysis, 114, 118
ITS4
code scanner, 109–110
knowledge, 262

rules. *See also* Taxonomy of coding errors, kingdoms.
history, 112–114
list of, 345–383

J

J2EE Bad Practices phylum, 285, 286–287
J2EE Misconfiguration phylum, 289
Java card, example, 195–200

K

Kernel vulnerabilities, 118, 122
KillerAppCo. *See* RMF, example.
Kingdoms. *See* Taxonomy of coding errors, kingdoms.
Kocher, Paul, 95
Knowledge base, Fortify, 132–134
Knowledge catalogs, 263–268, 269, 270–271
Knowledge
attack patterns, 264, 266
Department of Homeland Security portal, 264–274
description, 35–37
diagnostic knowledge, 264
experience, 261–262
expertise, 261–262
exploits, 264, 268
guidelines, 263–264, 265
historical knowledge, 264
historical risks, 264, 267
hurdles to overcome, 259–261

- knowledge catalogs, 263–268, 269, 270–271
 - prescriptive knowledge, 263–264
 - principles, 263–264, 265, 270–271
 - rules, 263–264, 266, 270–271
 - in the touchpoint process, 268–269
 - vulnerabilities, 264, 267
 - Knowledge requirements, architectural risk analysis, 147–148
- L**
- Landwher, Carl, 191
 - Least Privilege Violation phylum, 286
 - Leftover Debug Code phylum, 288
 - Legacy applications, cause of problems, 6–7
 - Lines of code. *See* Source code, lines of.
 - Literature. *See* Bibliography.
 - Local analysis, in code review, 111
 - Log Forging phylum, 283
- M**
- Malicious input, 201–204
 - Management without measurement, 244–245
 - Measurement
 - importance of, 73–74
 - metrics in the RMF, 75
 - metrics program, 247–250
 - ROI (return on investment), 74–75
 - Memory Leaks phylum, 287
 - Metrics. *See* Measurement.
 - Microsoft
 - Bob, 17
 - Gates memo, 29–34
 - high-level commitment, 245. *See also* Gates memo.
 - Nomenclature problems, 146–147
 - SDL (Secure Development Lifecycle), 239–240
 - threat modeling *versus* risk analysis, 146–147
 - Trustworthy Computing initiative, 29–34
 - Missing Access Control phylum, 286
 - Missing Custom Error Handler phylum, 289
 - Missing Error Handling phylum, 289
 - Misuse cases. *See* Abuse cases.
 - Mitigation strategies
 - defining, 45, 69–73
 - penetration testing, 183–184
 - risks, 45, 69–71
 - RMF, 69–71
 - MLOCs3, 11–13
 - MLOCs3^2+1, 11–13
 - Mobile code, and extensibility, 7–8
 - Mobile Code phylum, 288
 - Monitor tools, 181
 - MOPS tool, 122
 - “More lines, more bugs,” 10–11
 - Moving left, 91–93
 - Multithreading, 203
- N**
- Negatives, testing for, 172
 - Nessus, 230
 - .NET security model architecture diagram, 149–150
 - Network security
 - connectivity, cause of problems, 6–7
 - e-crime increase, 3
 - market value, 3
 - software vulnerability, increase in, 4–5
 - 19 Deadly Sins . . .*, 292, 296–297
 - nmap, 230
 - Non-Final Public Field phylum, 288
 - No one would ever do that, 212
 - Null Dereference phylum, 287
 - NullPointerException phylum, 287
- O**
- Object Hijack phylum, 288
 - Obsolete phylum, 287
 - OCTAVE, 142
 - Often Misused phylum, 285, 290–292
 - One page overview of architecture. *See* Forest-level view.
 - Open Platform, for Java
 - Card smart cards, 196
 - Opportunity, definition, 11
 - Ounce Labs, 123
 - Outside → in approach, 37, 174, 189–190, 252
 - Overly Broad Catch Block phylum, 287
 - Overly Broad Throws Declaration phylum, 287
 - “OWASP Top Ten . . . Vulnerabilities,” 292, 297

- Resource Injection phylum, 284
- Return on investment (ROI), 74–75, 153
- Risk analysis
 architectural level. *See* Architectural risk analysis.
 definition, 140
 exercise, 385–391
versus threat modeling, 146–147
- Risk calculation
 impact, 155–156
 modern model, 154–161
 traditional model, 152–154
- Risk management. *See also* RMF (risk management framework).
 applied risk management pillar, 26–27
 definition, 140
 risk-based security testing, 192–193
- Risk management framework (RMF). *See* RMF (risk management framework).
- Risk-based security testing
 adversarial testing, 193
 automation, 196
 conditions tested, 203
 constructive/destructive nature, 90
 description, 87
 example, 195–200
 eXtreme programming, 202
 firewalls, 189–190
 flyover, 87
 functional testing, 193
 Inside→out approach, 189
 Java card, example, 195–200
 malicious input, 201–204
 methodology, 194, 201
 multithreading, 203
 outside → in approach, 189–190
 and penetration testing, 204
 perimeter defense, 189–190
 personnel involved, 193–194
 process overview, 187–189
 risk management, 192–193. *See also* RMF (risk management framework).
 smart cards, example, 195–200
 SOAP protocol, 190
 “test-driven” design, 202
 timing, 203
- Risks
 analysis report, 71–72
 architectural risk analysis, 144–145
 business and technical, identifying, 43–44, 50–63
 data review, 68–69
 definition, 18
 impacts, 57
 indicators, 53
 likelihood scale, 56
 management framework. *See* RMF
 measuring and reporting, 46
 mitigation strategies
 defining, 45, 69–73
 penetration testing, 183–184
 risks, 45, 69–71
 RMF, 69–71
 questionnaires, 50–51
 ranking, 44–45
 severity key, 59
 synthesizing, 44–45
 synthesizing and ranking, 44–45, 63–68
- RMF (risk management framework)
 example
 business goal rankings, 53
 business impact scale, 58
 business peer review, 69
 business risk indicators, 55–56
 business risks, 50, 54, 60
 carrying out fixes and validation, 73
 defining a mitigation strategy, 69–73
 deliverables, 72–73
 fixes, 73
 gathering artifacts, 49
 goal-to-risk relationship, 67
 identifying business and technical risks, 50–63
 likelihood of occurrence, 55–56
 prioritized business goals, 52
 product risks, 50
 project research, 49–50
 project risks, 50
 ranking risks, 63–68
 research and interview data analysis, 52–59
 risk analysis report, 71–72
 risk data review, 68–69
 risk impacts, 57
 risk indicators, 53
 risk likelihood scale, 56
 risk mitigation, 69–71
 risk questionnaires, 50–51
 risk severity key, 59

- RMF (risk management framework), example (*cont.*)
 software artifact analysis, 61–63
 synthesizing and ranking risks, 63–68
 target project team, 51
 technical peer review, 69
 technical risks, 59–60, 61–62, 64–65, 68
 understanding business context, 49–50
 validation, 73
 iterative processing, 46–48
 measuring and reporting risk, 46
 process diagram, 42
 stages of activity
 carrying out fixes and validation, 43, 73
 defining the risk mitigation strategy, 45, 69–71
 identifying business and technical risks, 43–44, 50–63
 synthesizing and ranking risks, 44–45, 63–68
 understanding business context, 43, 49–50
- ROI (return on investment), 74–75, 153
- Rootkits, 181–182
- Rubin, Avi, 159–161
- Rules
 coverage, 112–114
 example, 119–122
 Fortify. *See* Fortify Source Code Analysis Suite.
 ITS4. *See also* Taxonomy of coding errors, kingdoms. history of, 112–114
 list of, 345–383
 knowledge catalog, 263–264, 266, 270–271
 schema, 115–118
- Rules Builder, 129–130
- S**
- Safeguards, architectural risk analysis, 145.
See also Mitigation strategies.
- Safety property violations, 122
- SATAN, 230
- SBI386, 155, 210
- SD West, 261
- SDL (Secure Development Lifecycle), 239–240, 256–257
- SDLC (Software Development Lifecycle), 240
- Secure Development Lifecycle (SDL), 239–240, 256–257
- Secure Software, 123
- SecureUML, 155
- Security
 built-in *versus* bolted on, 209–210
 defending the perimeter, does not work, 25
 versus software, 24–25
- Security engineering, rise of, 37–38
- Security Features vulnerability kingdom, 280, 285–286
- Security Band-Aid, 174
- Security operations
 constructive/destructive nature, 90–91
 description, 88
 flyover, 88
 inter-group cooperation. *See* Software developers and information security practitioners.
- Security professionals
 abuse case development, 210–211
 inter-group communication barriers, 224–225
 origin of, 101–102
 risk-based security testing, 193–194
 and software developers. *See* Software developers and information security practitioners.
 team building, 97
- Security requirements
 constructive/destructive nature, 90
 description, 88
 flyover, 88
 recommended reading, 89
- Security testing. *See* Risk-based security testing.
- Security tracker, 168
- Setting Manipulation phylum, 284
- Seven kingdoms. *See* Taxonomy of coding errors, kingdoms.
- Signal Handling Race Conditions phylum, 287
- SLA (service level agreement), 206–207, 253
- SLAM tool, 123
- Smart card, example, 195–200
- Smurfware exercise, 385–391
- SOA (Service Oriented Architecture), 6–7, 168, 184, 190
- SOAP protocol, 190
- Sockets phylum, 285
- Software
 artifacts. *See* Artifacts.
 process and religion, 239

- testing. *See* Penetration testing; Risk-based security testing.
 - vulnerability, cause of problems, 4–5
 - Software architect catfights, 165
 - Software developers and information security practitioners
 - abuse cases, 225–226
 - architectural risk analysis, 227–228
 - business risk analysis, 226–227
 - code review, 229–230, 231
 - deployment and operations, 231–232
 - example, 234–235
 - information security as Boogey man, 233
 - inter-group communication barriers, 224–225
 - inter-group cooperation, 232–234
 - penetration testing, 230–231
 - security testing, 228–229
 - “ugly baby” problem, 233
 - Software security
 - academic courses in, 98
 - versus* application security, 20–21
 - best practices. *See* Touchpoints.
 - definition, 3, 394
 - enterprise-wide. *See* Enterprise software security.
 - multidisciplinary effort, 100
 - and operations, 23–25
 - people, 96–100
 - pillars of, 25–28, 34–37
 - potential research areas, 318–320
 - responsibility for, 96–100
 - versus* software safety, 191
 - team building, 97, 99–100
 - three pillars, 25–28, 34–37
 - unique qualities of, 191–192
 - Software Security Manager, 132
 - Software security people, 99–100
 - Software security touchpoints, 27–28, 34–35
 - Source code
 - analysis, tutorial, 329–332
 - analyzers, commercial vendors, 124–125. *See also* Fortify Source Code Analysis Suite.
 - lines of
 - major operating systems, 10
 - normalizing, 11–13
 - relation to vulnerabilities, 11–13
 - Windows, 8–9
 - reviewing. *See* Code review, tools.
 - scanners, 107, 109–110
 - Specification checking, 122
 - Splint tool, 122–123
 - SQL injection, 201
 - SQL Injection phylum, 284
 - Standards-based architectural risk analysis, 142
 - Static code analysis
 - example, 135–137
 - history, 110–114
 - STRIDE, 147, 163, 169, 260
 - related to attack resistance analysis, 163
 - String Manipulation phylum, 285
 - String Termination Error phylum, 284
 - Struts phylum, 284
 - Synthesizing and ranking risks, RMF, 44–45, 63–68
 - System Information Leak phylum, 288
 - System.exit() phylum, 286
 - System testing, 183
- T**
- Taint analysis, 118
 - Taxonomy of coding errors
 - 19 Deadly Sins . . .*, 292, 296–297
 - attack classes, 294–295
 - hierarchy of, 278
 - “OWASP Top Ten . . . Vulnerabilities,” 292, 297
 - PLOVER (Preliminary List of Vulnerability Examples for Researchers), 295
 - versus* taxonomy of attack patterns, 277
 - Taxonomy of coding errors, kingdoms
 - API Abuse
 - description, 279–280
 - example, 290–292
 - phyla, 285
 - Code Quality, 281, 287–288
 - description, 279–280
 - example, 290–292
 - phyla, 285
 - definition, 278
 - Encapsulation, 281–282, 288–289
 - description, 279–280
 - example, 290–292
 - phyla, 285

- Taxonomy of coding errors,
 - kingdoms, Encapsulation (*cont.*)
 - Environment, 282, 289
 - description, 279–280
 - example, 290–292
 - phyla, 285
 - Error Handling, 281, 287
 - description, 279–280
 - example, 290–292
 - phyla, 285
 - Input Validation and Representation, 279, 283–285
 - description, 279–280
 - example, 290–292
 - phyla, 285
 - mapped to *19 Deadly Sins* . . . , 296–297
 - mapped to “OWASP Top Ten . . . Vulnerabilities,” 297
 - Security Features, 280, 285–286
 - description, 279–280
 - example, 290–292
 - phyla, 285
 - summary list of, 279
 - Time and State, 280–281, 286–287
 - description, 279–280
 - example, 290–292
 - phyla, 285
- Taxonomy of coding errors, phyla
 - API Abuse Kingdom, 285
 - ASP.NET Misconfiguration, 289
 - Authentication, 285, 290–292
 - Buffer Overflow, 283
 - Catch `NullPointerException`, 287
 - Code Quality Kingdom, 287–288
 - Command Injection, 283
 - Comparing Classes by Name, 288
 - Creating Debug Binary, 289
 - Cross-Site Scripting, 283
 - Dangerous Functions, 285
 - Data Leaking Between Users, 288
 - Deadlock, 286
 - definition, 278
 - Directory Restriction, 285
 - Double Free, 287
 - Duplicate Validation Forms, 284
 - Empty Catch Block, 287
 - Empty Password in Configuration File, 286
 - encapsulation kingdom, 288–289
 - Environment kingdom, 289
 - Erroneous `validate()` Method, 284
 - Error Handling kingdom, 287
 - Exception Handling, 285
 - Failure to Begin a New Session . . . , 286
 - File Access Race Condition, 286
 - Form Field Without Validator, 284
 - Format String, 283
 - `getConnection()` method, 285
 - Hard-Coded Passwords, 286
 - Heap Inspection, 285
 - HTTP Response Splitting, 283
 - Illegal Pointer Value, 283
 - Inconsistent Implementations, 287
 - Input Validation And Representation kingdom, 283–285
 - Insecure Compiler Optimization, 289
 - Insecure Randomness, 285
 - Insecure Temporary File, 286
 - Integer Overflow, 283
 - J2EE Bad Practices, 285, 286–287
 - J2EE Misconfiguration, 289
 - Least Privilege Violation, 286
 - Leftover Debug Code, 288
 - Log Forging, 283
 - Memory Leaks, 287
 - Missing Access Control, 286
 - Missing Custom Error Handler, 289
 - Missing Error Handling, 289
 - Mobile Code, 288
 - need for additional, 289–290
 - Non-Final Public Field, 288
 - Null Dereference, 287
 - Object Hijack, 288
 - Obsolete, 287
 - Often Misused, 285, 290–292
 - Overly Broad Catch Block, 287
 - Overly Broad Throws Declaration, 287
 - Password in Configuration File, 286, 289
 - Password Management, 286
 - Path Manipulation, 285
 - Path Traversal, 283
 - Privacy Violation, 286
 - Private Array-Type Field . . . , 288
 - Privilege Management, 285

- Process Control, 283
- Public Data
 - Assigned . . . , 288
- Race Condition, 286
- Resource Injection, 284
- Security Features kingdom, 285–286
- Setting Manipulation, 284
- Signal Handling Race
 - Conditions, 287
- Sockets, 285
- SQL Injection, 284
- String Manipulation, 285
- String Termination Error, 284
- Struts, 284
- System Information
 - Leak, 288
- System.exit(), 286
- Threads, 287
- Time And State kingdom, 286–287
- TOCTOU (time-of-check-time-of-use), 286
- Trust Boundary Violation, 289
- Unchecked Return Value, 285, 287
- Undefined Behavior, 288
- Uninitialized Variable, 288
- Unreleased Resource, 288
- Unsafe Bean Declaration, 289
- Unsafe JNI, 284
- Unsafe Reflection, 285
- Unused Validation Form, 284
- Unvalidated Action
 - Form, 284
- Use After Free, 288
- Use Of Inner Class, 288
- Validation Class Not
 - Extended, 284
- Validator Turned Off, 284
- Validator Without Form
 - Field, 284
- Weak Access Permissions, 289
- Weak Cryptography, 286
- XML Validation, 285
- Taxonomy of vulnerabilities, 191
- Teams. *See* Security professionals.
- Tent example, 209
- “Test-driven” design, 202
- Testing. *See* Penetration testing; Risk-based security testing.
- Think like a bad guy. *See* Black hat activities.
- Threads phylum, 287
- Threat modeling *versus* risk analysis, 146–147
- Threats, architectural risk analysis, 145
- Three pillars. *See* Pillars of software security.
- Time and State vulnerability kingdom, 280–281, 286–287
- Time as essential issue, 203
- Timing, risk-based security testing, 203
- TOCTOU (time-of-check-time-of-use), 111, 203, 286
- Tools
 - characteristics of, 125–127
 - code review. *See* Code review, tools.
 - commercial vendors. *See* Commercial source code analysis tool vendors.
 - Nessus, 189–190
 - penetration testing
 - APISPY32, 181
 - breakpoint setters, 181
 - CANVAS, 180
 - Cenzic, 180
 - control flow, 181
 - coverage, 181
 - decompilers, 181
 - disassemblers, 181
 - fault injection, 180
 - Hailstorm, 180
 - Holodeck, 180
 - rootkits, 181–182
 - shell code, 181
 - port scanning, 189–190
 - problems with, 127
- Touchpoints, 83–103. *See also specific touchpoints.*
 - as best practices, 94, 96
 - black hat activities, 89–91
 - constructive activities, 89–91
 - destructive activities, 89–91
 - example, 95
 - list of, 85
 - abuse cases, 205
 - architectural risk analysis, 139
 - code review, 105
 - penetration testing, 171
 - risk-based security testing, 187
 - security operations, 223
 - order of effectiveness, 85
 - overview, 83–85
 - sequence of, 84
 - timing in the lifecycle, 91–94
 - white hat activities, 89–91
- Training, academic courses, 98
- Training, software security, 250
- Training without assessment, 245
- Trinity of trouble
 - complexity, 8–13

Trinity of trouble (*cont.*)
 connectivity, 6–7
 extensibility, 7–8
 Trust Boundary Violation
 phylum, 289
 Trustworthy Computing
 Initiative, 29–34

U

“Ugly baby” problem, 233
 UML, 205
 UMLsec, 155
 Unchecked Return Value
 phylum, 285, 287
 Undefined Behavior phy-
 lum, 288
 Understanding business
 context, RMF, 43,
 49–50
 Unicode attacks, 260
 Unit testing, 188
 Uninitialized Variable phy-
 lum, 288
 Unreleased Resource phy-
 lum, 288
 Unsafe Bean Declaration
 phylum, 289
 Unsafe JNI phylum, 284
 Unsafe Reflection phylum,
 285
 Unused Validation Form
 phylum, 284
 Unvalidated Action Form
 phylum, 284
 Use After Free phylum, 288
 Use cases, 205. *See also*
 Abuse cases.
 Use Of Inner Class phylum,
 288

V

`validate()` method phy-
 lum, 284

Validation and fixes, RMF,
 43, 73

Validation Class Not
 Extended phylum,
 284

Validator Turned Off phy-
 lum, 284

Validator Without Form
 Field phylum, 284

van Wyk, Ken, 224,
 234–235

Vendors, software
 accountability, 207
 security tools. *See* Com-
 mercial tool ven-
 dors.

Viega, John, 166

Voas, Jeff, 180

Voting machine security,
 159–161

VP of yadda yadda, 173

Vulnerabilities
 architectural risk analy-
 sis, 145
 bibliography, 293–294
 categories, 293–294
 causes of problems, 4–5
 definition, 191, 394
 design-level, 19–20,
 191–192

factor of lines of source
 code, 11–13

flaws *versus* bugs, 191

increase in, 4–5

kernel, 118, 122

knowledge catalog, 264,
 267

“OWASP Top Ten . . .
 Vulnerabilities,”
 292, 297

PLOVER (Preliminary
 List of Vulnerability
 Examples for
 Researchers), 295

Seven kingdoms. *See*
 Taxonomy of doing
 errors, kingdoms.
 taxonomy of, 191
 VulnWatch, 168

W

Wagner, Dave, 15

Weak Access Permissions
 phylum, 289

Weak Cryptography phy-
 lum, 286

Weakness analysis, in archi-
 tectural risk analy-
 sis, 167–169

Web Services, cause of
 problems, 6–7

West, Jacob, 133

White box testing,
 194–195

White hat activities, touch-
 points, 89–91, 172

Whittaker, James, 180

Who cares, 40, 227

Windows, complexity, 8–9

Wing, Jeannette, 319

X

xg++ tool, 118, 122

XML Validation phylum,
 285

XSS. *See* cross-site script-
 ing.

Y

Yin/yang, 89–91

YP/Samba/Squid hole, 177

Z

Zero-day exploit, 191