



Index

A

Addition operations, 169–172
 Alcatraz, 272
 alloca() function, 107
The Annotated C++ Reference Manual, 16
 ANSI C standard arguments, 205–208
 Application Verifier tool, 145
 Arbitrary memory write, 80–81
 Arbitrary precision arithmetic, 196
 Arc injection, 48–51
 Architecture and design, software development, 286–287
 Argument pointers, 217–218
 Arguments

- direct access to, 227–230
- passing, 207

 Arithmetic. *See also* Integers.

- arbitrary precision, 196
- GMP (GNU Multiple Precision Arithmetic Library), 196
- integer conversions, 164
- machine-level, 169

Arithmetic operations

- addition, 169–172
- division, 177–181
- error detection, 168
- machine-level arithmetic, 169
- multiplication, 174–177
- postconditions, 168
- preconditions, 168
- subtraction, 172–174
- unexpected values, 167–168

Arrays

- bounds checking, 68–69
- overruns, 67
- scalars and, 106

Art of Computer Programming, 99
 Articles. *See* Books and publications.
 atexit() function, 88–90
 Atomic operations, 264
 Attackers

- competitive intelligence professionals, 8
- crackers, 7
- criminals, 7

definition, 12
 hackers, 7
 information warriors, 8–9
 insiders, 7
 potential attackers, 7–9
 terrorists, 8
 threat assessment, 6–9
 Audit Workbench, 296–297

B

Bash, integer vulnerabilities, 199–200
 Best-fit memory allocation, 99–100
 Bins, 109
 Black listing, 293–294
 Blaster worm, 1–4
 Books and publications

- The Annotated C++ Reference Manual*, 16
- Art of Computer Programming*, 99
- The C Programming Language*, 16
- Secure Programming in C and C++*, 16

 Brown, Preston, 260



- Buffer overflows
 - code injection, 44–48
 - data locations, 78
 - d1malloc() function
 - frontlink technique, 114–117
 - unlink technique, 111–114
 - formatted output, 214–215
 - function pointers, 78–80
 - heap-based exploits, 126–132, 147
 - pointers. *See* Pointers, overwriting.
 - RtlHeap, 126–132
 - stack smashing, 40–44, 69–70
 - strings. *See* Strings.
 - transferring program control
 - arc injection, 48–51
 - code injection, 44–48
- C**
- C and C++
 - alternative languages, 19
 - The Annotated C++ Reference Manual*, 16
 - history of, 16–17
 - legacy code, 18–19
 - portability, 17–18
 - preservation, 18
 - problems with C, 17–18
 - progress, 18
 - standards, 16–17
 - type safety, 18
- C language compatible library, 192–194
- The C Programming Language*, 16
- calloc() function, 98–99
- Canaries
 - pointer overwrite mitigation, 95
 - string mitigation, 69–70
- CDE ToolTalk, 243
- __cdecl convention, 207
- Change State Property, 248
- Checklists and guidelines, software development, 300–301
- chroot jail, 270–271
- Closing the race window, 262–266
- cmd.exe, 3
- Code audits, 300
- Code injection, 44–48
- Code segments, 36–37
- Comair, integer failure, 157
- Competitive intelligence professionals, 8. *See also* Attackers.
- Compiler checks, 236, 290
- Compiler-generated runtime checks, 190
- Complete mediation, 278–279
- Computer security, definition, 10
- Computer system, definition, 10
- Concatenating strings, 54–61
- Concurrency, file I/O
 - Change State Property, 248
 - Concurrency Property, 248
 - critical sections, 249
 - deadlocks, 248–250
 - file locking, 252–254
 - files as locks, 252–254
 - mandatory locks, 254
 - mutual exclusion, 248–250
 - named mutex object, 254
 - race conditions, 248
 - Shared Object Property, 248
 - synchronization primitives, 249
 - synchronizing across processes, 254
 - TOCTOU (time of check, time of use), 250–252
 - trusted/untrusted control flows, 250–252
- Concurrency Property, 248
- Conover, Matt, 123
- Control transfer instructions, 81–82
- Conversion specifiers, 209–210
- Conversions, integer
 - arithmetic, 164
 - promotions, 159–160
 - rank, 160–161
 - signed characters, 162–163
 - from signed types, 161–162
 - unsigned characters, 162–163
 - from unsigned types, 161
- Copying strings, 54–61
- Cost of damages
 - Blaster worm, 4
 - e-crimes in general, 5–7
 - financial losses, by type of attack, 6
- Crackers, 7. *See also* Attackers.
- Crashing programs, 216
- Crimes. *See* E-crimes.
- Criminals, 7. *See also* Attackers.
- Critical sections, 249
- CRT memory functions, 122
- Cryptography, string vulnerabilities, 72–73
- Cyber-terrorism, 8
- D**
- Data locations, buffer overflow, 78
- Data pointers, 80–81
- Data sanitization, 292–295
- Data segments, 36–37
- Data types. *See* Type.
- Date execution prevention (DEP), 303–304
- Deadlocks, 248–250
- Defense in depth, 304
- Defensive strategies. *See* Mitigation strategies.
- DEP (date execution prevention), 303–304
- Department of Homeland Security, 2
- Detection and recovery strategies, 67–71
- Developing software. *See* Software development.
- Development platforms
 - compilers, 21–23
 - GCC (GNU Compiler Collection), 22–23
 - Linux, 21, 22
 - Microsoft Windows, 20
 - operating systems, 20–21
 - Visual C++, 21–22
- Direct argument access, 227–230

- Division operations, 177–181
- `dllmalloc()` function
 - bins, 109
 - buffer overflows, 111–117
 - double-free vulnerabilities, 117–120
 - heads, 109
 - memory management, 108–111
 - writing to freed memory, 120
- `dmalloc` library, 143
- Double-freed memory
 - CVS double-free, 148
 - `dllmalloc()` function, 117–120
 - dynamic memory management, 104–106
 - `RtlHeap`, 134–137
- `.dtors` section, 84–86
- Dynamic analysis, 272
- Dynamic memory allocator
 - `alloca()` function, 107
 - `calloc()` function, 98–99
 - `dllmalloc()` function
 - bins, 109
 - buffer overflows, 111–117
 - double-free vulnerabilities, 117–120
 - heads, 109
 - memory management, 108–111
 - writing to freed memory, 120
 - `free()` function, 98
 - `malloc()` function, 98, 107
 - `realloc()` function, 98
- Dynamic memory management.
 - See also* Heap-based exploits.
 - `alloca()` function, 107
 - allocating memory, 98–99
 - best-fit allocation, 99–100
 - `calloc()` function, 98–99
 - common errors
 - allocation functions, 107
 - checking return values, 102–103
 - double-freeing memory, 104–106
 - improperly paired functions, 106
 - initialization, 100–101
 - referencing freed memory, 104
 - scalars and arrays, 106
 - error conditions
 - allocation functions, 107
 - checking return values, 102–103
 - double-freeing memory, 104–106
 - improperly paired functions, 106
 - initialization, 100–101
 - referencing freed memory, 104
 - scalars and arrays, 106
 - first-fit allocation, 99–100
 - `free()` function, 98
 - freeing memory, 98–99
 - functions for, 98–99
 - `malloc()` function, 98, 107
 - `phkmmalloc` program, 102–103
 - `realloc()` function, 98
 - reserving contiguous bytes, 99–100
- Dynamic prevention strategies, 51
- Dynamic use of static content, 231–232
- E**
- Economy of mechanism, 278
- E-crimes, average number reported, 5. *See also* Cost of damages; Threat assessment.
- Electric Fence tool, 143–144
- ELF (executable and linking format), 83
- Eliminating race objects, 266–269
- E-mail, string vulnerabilities, 73–74
- Eraser, 272
- Error conditions
 - dynamic memory management
 - allocation functions, 107
 - checking return values, 102–103
 - double-freeing memory, 104–106
 - improperly paired functions, 106
 - initialization, 100–101
 - referencing freed memory, 104
 - scalars and arrays, 106
 - integer operations, 168
 - integers
 - integer overflow, 164–166
 - nonexceptional logic errors, 186–187
 - sign errors, 166–167, 183–184
 - testing, 196–197
 - truncation errors, 167
 - strings
 - deprecated functions, 32–33
 - length errors, 26
 - null-termination errors, 31–32
 - off-by-one errors, 29–31
 - string truncation, 32
 - unbounded string copies, 27–29
- ESC (extended static checking), 271
- Exception handling
 - SEH (structured exception handling), 92–94
 - system default exception handling, 94–95
 - VEH (vectored exception handling), 92
- Exec Shield, 239–240
- Executable and linking format (ELF), 83

- Exploits
 - description, 13–14
 - file I/O
 - nonunique temporary file names, 261–262
 - symbolic linking, 255–257
 - symlink, 255–257
 - temporary file open, 257–260
 - tmp cleaners, 260
 - trusted filenames, 261
 - unlink() race, 260
 - heap-based. *See also* Dynamic memory management; RtlHeap.
 - Application Verifier tool, 145
 - CVS buffer overflow, 147
 - CVS double-free, 148
 - dmalloc library, 143
 - Electric Fence tool, 143–144
 - Gnu Checker tool, 144
 - guard pages, 142
 - heap integrity detection, 139–140
 - Insure ++ tool, 144–145
 - MDAC (Microsoft Data Access) components, 147–148
 - memory management conventions, 138–139
 - MIT Kerberos 5, 149
 - null pointers, 138
 - OpenBSD, 142–143
 - phkmallocc program, 140–141
 - Purify tool, 143
 - PurifyPlus tool, 143
 - randomization, 141–142
 - runtime analysis tools, 143–145
 - Valgrind tool, 144
 - Windows XP SP2, 145–146
 - Extended integers, 153, 155–156
 - Extended static checking (ESC), 271
 - dynamic analysis, 272
 - eliminating race objects, 266–269
 - Eraser, 272
 - ESC (extended static checking), 271
 - file descriptors *versus* file-names, 268
 - MultiRace, 272
 - mutual exclusion, 262–264
 - principle of least privilege, 269–270
 - race detection tools, 271–272
 - RaceGuard, 272
 - secure property checking, 264–266
 - shared directories, 268
 - shared resources, 266–268
 - static analysis tools, 271
 - temporary files, 268–269
 - ThreadChecker, 272
 - thread-safe functions, 264
 - trustworthy directories, 270
 - Warlock, 271
- File locking, 252–254
- Files as locks, 252–254
- Financial loss. *See* Cost of damages.
- First-fit memory allocation, 99–100
- Flags, 210
- Flawfinder, 295
- Format strings
 - description, 209–211
 - vulnerability, 213–214
- FormatGuard tool, 240
- Formatted output
 - ANSI C standard arguments, 205–208
 - argument passing, 207
 - argument pointers, 217–218
 - __cdecl convention, 207
 - conversion specifiers, 209–210
 - __fastcall convention, 207
 - flags, 210
- F
 - Fail-safe defaults, 278
 - __fastcall convention, 207
 - FedCIRC (Federal Computer Incident Response Center), 2
 - fgets() function, 52–54
 - File descriptors *versus* filenames, 268
 - File I/O
 - concurrency
 - Change State Property, 248
 - Concurrency Property, 248
 - critical sections, 249
 - deadlocks, 248–250
 - file locking, 252–254
 - files as locks, 252–254
 - mandatory locks, 254
 - mutual exclusion, 248–250
 - named mutex object, 254
 - race conditions, 248
 - Shared Object Property, 248
 - synchronization primitives, 249
 - synchronizing across processes, 254
 - TOCTOU (time of check, time of use), 250–252
 - trusted/untrusted control flows, 250–252
 - exploits
 - nonunique temporary file names, 261–262
 - symbolic linking, 255–257
 - symlink, 255–257
 - temporary file open, 257–260
 - tmp cleaners, 260
 - trusted filenames, 261
 - unlink() race, 260
 - mitigation strategies
 - Alcatraz, 272
 - atomic operations, 264
 - chroot jail, 270–271
 - closing the race window, 262–266
 - controlling access, 269–271

- format strings, 209–211
 - `fprintf()`, 208
 - GCC (GNU Compiler Collection), 211–212
 - length modifier, 211
 - mitigation strategies
 - compiler checks, 236
 - direct argument access, 227–230
 - dynamic use of static content, 231–232
 - Exec Shield, 239–240
 - FormatGuard tool, 240
 - `iostream` versus `stdio`, 234
 - ISO/IEC WDTR 24731, 233–234
 - lexical analysis, 236
 - Libsafe tool, 241
 - modifying variadic functions, 237–239
 - pscan tool, 236
 - restricting bytes written, 232–234
 - stack randomization, 225–230
 - static binary analysis, 241–242
 - static taint analysis, 237
 - testing, 234–235
 - `-wformat` flag, 236
 - `-wformat-nonliteral` flag, 236
 - `-wformat-security` flag, 236
 - writing addresses in two words, 227
 - naming conventions, 207
 - precision, 211
 - `printf()`, 208
 - security-enhanced functions, 233–234
 - `snprintf()`, 208–209
 - `sprintf()`, 208
 - `__stdcall` convention, 207
 - `syslog()`, 209
 - UNIX System V `varargs.h` macros, 208
 - variadic functions, 204–208
 - `vfprintf()`, 209
 - Visual C++, 212–213
 - `vprintf()`, 209
 - `vsprintf()`, 209
 - vulnerabilities
 - buffer overflows, 214–215
 - CDE ToolTalk, 243
 - crashing programs, 216
 - format string, 213–214
 - internationalization, 224–225
 - output streams, 215–216
 - overwriting memory, 220–224
 - viewing memory content, 218–220
 - viewing stack content, 216–218
 - Washington University FTP daemon, 242
 - width, 211
 - Fortify, 296–297
 - `fprintf()`, 208
 - Frames, 37
 - `free()` function, 98
 - Free lists, 123–125
 - Frontlink technique, 114–117
 - Function pointers, 78–80
 - Functions. *See also specific functions.*
 - deprecated, 32–33
 - formatted output. *See* Formatted output.
 - safe string handling, 59
 - security-enhanced, 233–234
 - string handling, 54–59, 64–66
 - virtual, 87–88
 - Fuzz testing, 299–300
- G**
- GCC (GNU Compiler Collection), 190, 192, 211–212
 - `gets_s()` function, 52–54
 - Global offset table (GOT), 83–84
 - GMP (GNU Multiple Precision Arithmetic Library), 196
 - Gnu Checker tool, 144
 - GNU Compiler Collection (GCC), 190, 192, 211–212
 - GNU Multiple Precision Arithmetic Library (GMP), 196
 - GOT (global offset table), 83–84
 - Guard pages, 142
 - Guidelines and checklists, software development, 300–301
- H**
- Hackers, 7. *See also* Attackers.
 - HD Moore, 2
 - Heads, 109
 - Heap integrity detection, 139–140
 - Heap memory API, 122
 - Heap segments, 36–37
 - Heap-based exploits. *See also* Dynamic memory management.
 - mitigation strategies
 - Application Verifier tool, 145
 - `dmalloc` library, 143
 - Electric Fence tool, 143–144
 - Gnu Checker tool, 144
 - guard pages, 142
 - heap integrity detection, 139–140
 - Insure ++ tool, 144–145
 - memory management conventions, 138–139
 - null pointers, 138
 - OpenBSD, 142–143
 - `phkmalloc` program, 140–141
 - Purify tool, 143
 - PurifyPlus tool, 143
 - randomization, 141–142
 - runtime analysis tools, 143–145
 - Valgrind tool, 144
 - Windows XP SP2, 145–146

- Heap-based exploits, *continued*
 - vulnerabilities. *See also*
 - Dynamic memory management; RtlHeap.
 - CVS buffer overflow, 147
 - CVS double-free, 148
 - MDAC (Microsoft Data Access) components, 147–148
 - MIT Kerberos 5, 149
 - Horovitz, Oded, 123–125
 - Howard, Michael, 192
- I**
- Information warriors, 8–9. *See also* Attackers.
- Input validation, 51–54, 291
- Insiders, 7. *See also* Attackers.
- Instruction pointers, 81–82
- Insure ++ tool, 144–145
- int, 155
- Integers
 - arithmetic operations
 - addition, 169–172
 - division, 177–181
 - error detection, 168
 - machine-level arithmetic, 169
 - multiplication, 174–177
 - postconditions, 168
 - preconditions, 168
 - subtraction, 172–174
 - unexpected values, 167–168
 - conversions
 - arithmetic, 164
 - promotions, 159–160
 - rank, 160–161
 - signed characters, 162–163
 - from signed types, 161–162
 - unsigned characters, 162–163
 - from unsigned types, 161
 - data types
 - extended, 153, 155–156
 - platform-specific, 156
 - signed, 154–155
 - standard, 153, 155–156
 - unsigned, 154–155
- error conditions
 - integer overflow, 164–166
 - nonexceptional logic errors, 186–187
 - sign errors, 166–167, 183–184
 - testing, 196–197
 - truncation errors, 167
 - failure example, 157
 - int, 155
 - intptr_t, 156
 - long int, 155
 - long long int, 155
- mitigation strategies
 - arbitrary precision arithmetic, 196
 - C language compatible library, 192–194
 - compiler-generated runtime checks, 190
 - GCC (GNU Compiler Collection), 192
 - GMP (GNU Multiple Precision Arithmetic Library), 196
 - Java BigInteger, 196
 - range checking, 188–189
 - RCSint class, 195–196
 - safe integer operations, 191–196
 - SafeInt class, 194–195
 - source code audit, 197
 - strong typing, 189–190
 - testing, 196–197
- negative, 152–153
- nonexceptional logic errors, 186–187
- one's complement, 152–153
- overflow, 164–166, 182–183
- overflow in XDR library, 197–198
- ptrdiff_t, 156
- ranges, 157–158
- representation, 152–153
- short int, 155
- signed char, 155
- signed-magnitude, 152–153
- size_t, 156
- two's complement, 152–153
- uintmax_t, 156
- uintptr_t, 156
- vulnerabilities
 - Bash, 199–200
 - integer overflow, 182–183
 - integer overflow in XDR library, 197–198
 - sign errors, 183–184
 - truncation errors, 184–186
 - Windows DirectX MIDI Library, 198–199
 - wchar_t, 156

J

- Java BigInteger, 196

K

- Kamp, Poul-Henning, 140
- Kerberos
 - heap-based vulnerabilities, 149
 - string vulnerabilities, 72–73

L

- Last Stage of Delirium (LSD) Research Group, 1–2
- Lea, Doug, 107
- Least common mechanism, 279–281
- Least privilege, 279–280
- LeBlanc, David, 194
- Legacy code, 18–19
- Length calculation, strings, 58–59
- Length errors, strings, 26
- Length modifier, 211
- Lexical analysis, 236
- Libsafe library, 71, 241
- Libverify library, 71

Index

- Local, global memory API, 122
- long int, 155
- long long int, 155
- longjmp() function, 90–92
- Look-aside lists, 125
- Look-aside table, 137–138
- LSD (Last Stage of Delirium)
 - Research Group, 1–2
- M**
- Machine-level arithmetic integer operations, 169
- malloc() function, 98, 107
- Mandatory locks, 254
- memcpy_s() function, 54
- memmove_s() function, 54
- Memory. *See also* Dynamic memory; Process memory.
 - chunks, 125–126
 - content, viewing, 218–220
 - management conventions, 138–139
 - overwriting, 220–224
 - permissions, 302–304
- Memory manager, 98
- Memory-mapped file API, 122–123
- Metamail package, string vulnerabilities, 73–74
- Metasploit Project, 2
- MIME types, string vulnerabilities, 73–74
- Mitigation strategies
 - description, 15–16
 - file I/O
 - Alcatraz, 272
 - atomic operations, 264
 - chroot jail, 270–271
 - closing the race window, 262–266
 - controlling access, 269–271
 - dynamic analysis, 272
 - eliminating race objects, 266–269
 - Eraser, 272
 - ESC (extended static checking), 271
 - file descriptors *versus* file-names, 268
 - MultiRace, 272
 - mutual exclusion, 262–264
 - principle of least privilege, 269–270
 - race detection tools, 271–272
 - RaceGuard, 272
 - secure property checking, 264–266
 - shared directories, 268
 - shared resources, 266–268
 - static analysis tools, 271
 - temporary files, 268–269
 - ThreadChecker, 272
 - thread-safe functions, 264
 - trustworthy directories, 270
 - Warlock, 271
 - formatted output
 - compiler checks, 236
 - direct argument access, 227–230
 - dynamic use of static content, 231–232
 - Exec Shield, 239–240
 - FormatGuard tool, 240
 - iostream *versus* stdio, 234
 - ISO/IEC WDTR 24731, 233–234
 - lexical analysis, 236
 - Libsafe tool, 241
 - modifying variadic functions, 237–239
 - pscan tool, 236
 - restricting bytes written, 232–234
 - stack randomization, 225–230
 - static binary analysis, 241–242
 - static taint analysis, 237
 - testing, 234–235
 - wformat flag, 236
 - wformat-nonliteral flag, 236
 - wformat-security flag, 236
 - writing addresses in two words, 227
- heap-based exploits
 - Application Verifier tool, 145
 - dmalloc library, 143
 - Electric Fence tool, 143–144
 - Gnu Checker tool, 144
 - guard pages, 142
 - heap integrity detection, 139–140
 - Insure ++ tool, 144–145
 - memory management conventions, 138–139
 - null pointers, 138
 - OpenBSD, 142–143
 - phkmallocc program, 140–141
 - Purify tool, 143
 - PurifyPlus tool, 143
 - randomization, 141–142
 - runtime analysis tools, 143–145
 - Valgrind tool, 144
 - Windows XP SP2, 145–146
- integers
 - arbitrary precision arithmetic, 196
 - C language compatible library, 192–194
 - compiler-generated runtime checks, 190
 - GCC (GNU Compiler Collection), 192
 - GMP (GNU Multiple Precision Arithmetic Library), 196
 - Java BigInteger, 196
 - range checking, 188–189
 - RCSint class, 195–196
 - safe integer operations, 191–196
 - SafeInt class, 194–195
 - source code audit, 197

- Mitigation strategies, *continued*
 - strong typing, 189–190
 - testing, 196–197
- pointer overwrite
 - canaries, 95
 - privilege reduction, 95
 - redeclaring stack buffers, 79–80
 - W^X, 95
- strings
 - array bounds checking, 68–69
 - canaries, 69–70
 - concatenating strings, 54–61
 - copying strings, 54–61
 - detection and recover, 67–71
 - dynamic, 51
 - fgets() function, 52–54
 - gets_s() function, 52–54
 - input validation, 51–54
 - length calculation, 58–59
 - Libsafe library, 71
 - Libverify library, 71
 - memcpy_s() function, 54
 - memmove_s() function, 54
 - moving strings, 54–58
 - nonexecutable stacks, 67
 - overruns of local arrays, 67
 - ProPolice, 70–71
 - runtime checks, 67
 - safe handling functions, 59
 - SafeStr library, 62–63
 - SSP (Stack Smashing Protector), 70–71
 - stack pointer verification, 67
 - stackgap, 68
 - static, 51
 - std::string class, 61–62
 - strcat() function, 54–55
 - strcat_s() function, 55–56
 - strcpy() function, 54–55
 - strcpy_s() function, 55–56
 - string streams, 64–66
 - strlcat() function, 55–56
 - strlcpy() function, 55–56
 - strlen() function, 58–59
 - strncat() function, 56–57
 - strncat_s() function, 57–58
 - strncpy() function, 56–57
 - strncpy_s() function, 57–58
 - Strsafe.h functions, 59
 - trusted *versus* untrusted data, 63
 - Vstr library, 63–64
- Mitigation strategies, software development
 - architecture and design, 286–287
 - Audit Workbench, 296–297
 - black listing, 293–294
 - code audits, 300
 - compiler checks, 290
 - data sanitization, 292–295
 - defense in depth, 304
 - DEP (date execution prevention), 303–304
 - development life cycle, 276
 - Flawfinder, 295
 - Fortify, 296–297
 - fuzz testing, 299–300
 - guidelines and checklists, 300–301
 - independent security review, 301–302
 - input validation, 291
 - It's the Software Stupid!, 295–296
 - memory permissions, 302–304
 - off-the-shelf software, 288–289
 - PaX, 303
 - penetration testing, 299
 - planning development, 305–306
 - PREfast, 298
 - PREfix, 298
 - Prevent, 297–298
 - Prexis, 297
 - quality assurance, 298–302
 - quality management, 306–307
 - quality requirements engineering, 282–283
 - RATS (Rough Auditing Tool for Security), 295
 - Rules Builder, 296–297
 - SCA (Source Code Analysis), 296–297
 - Secure Coding Rulepacks, 296–297
 - secure wrappers, 289
 - security principles
 - complete mediation, 278–279
 - economy of mechanism, 278
 - fail-safe defaults, 278
 - least common mechanism, 279–281
 - least privilege, 279–280
 - open design, 279
 - psychological acceptability, 281
 - separation of privilege, 279
 - summary of, 277
 - Security Scanner, 295–296
 - source code analysis, 295–298
 - static analysis, 295–298
 - testing software, 294–295
 - threat modeling, 283–284
 - tracking development, 305–306
 - TSP-Secure, 304–305
 - use/misuse cases, 284–286
 - vulnerabilities in existing code, 288
 - W^X policy, 302–303
 - white listing, 294
- Modifying variadic functions, 237–239
- Moving strings, 54–58
- msblast.exe, 3
- Multiplication operations, 174–177
- MultiRace, 272
- mutex object, 254
- Mutual exclusion, 248–250, 262–264

N

Named mutex object, 254
 Naming conventions, 207
 National Infrastructure Protection Center (NIPC), 2
 NCS (National Communications System), 2
 Negative integers, 152–153
 Network administrators, 12
 NIPC (National Infrastructure Protection Center), 2
 Nonexceptional integer logic errors, 186–187
 Nonexecutable stacks, 67
 Nonunique temporary file names exploits, 261–262
 Null pointers, 138
 Null-termination errors, 31–32

O

Off-by-one errors, 29–31
 Off-the-shelf software, 288–289
 One's complement, 152–153
`on_exit()` function, 88–90
 Open design, 279
 OpenBSD, heap-based exploit mitigation, 142–143
 Operating systems. *See* Development platforms.
 Output stream vulnerability, 215–216
 Overwriting pointers. *See* Pointers, overwriting.

P

Paller, Alan, 23
 Password example, string vulnerabilities, 33–34
 PaX, 303
 PEB (process environment block), 123
 Penetration testing, 299
 Pethia, Richard, 3–4
`phkmallocc` program, 102–103, 140–141
 Planning software development, 305–306

Platforms. *See* Development platforms; *specific platforms*.
 Platform-specific integers, 156
 Pointers, overwriting
 arbitrary memory write, 80–81
 `atexit()` function, 88–90
 control transfer instructions, 81–82
 data locations, 78
 data pointers, 80–81
 definition, 77
 `.ctors` section, 84–86
 exception handling
 SEH (structured exception handling), 92–94
 system default exception handling, 94–95
 VEH (vectored exception handling), 92
 function pointers, 78–80
 GOT (global offset table), 83–84
 instruction pointers, 81–82
 `longjmp()` function, 90–92
 mitigation
 canaries, 95
 privilege reduction, 95
 redeclaring stack buffers, 79–80
 W^X, 95
 `on_exit()` function, 88–90
 at program termination, 88–90
 restoring saved environment, 90–92
 virtual pointers, 87–88
 Portability, C and C++, 17–18
 Postconditions, 168
 Potential attackers, 7–9. *See also* Attackers.
 Precision, 211
 Preconditions, 168
`PREfast`, 298
`PREfix`, 298
 Preservation, C language, 18
 Prevent, 297–298
 Prexis, 297

Principle of least privilege, 269–270
`printf()`, 208
 Privilege reduction, 95
 Process environment block (PEB), 123
 Process memory
 code segments, 36–37
 data segments, 36–37
 frames, 37
 heap segments, 36–37
 organization, 36–37
 stack management, 37–39
 stack segments, 36–37
 Programmers, definition, 11
 Progress, C language, 18
 Promotions, integer conversions, 159–160
 ProPolice, 70–71
 Protection. *See* Mitigation strategies.
`pscan` tool, 236
 Psychological acceptability, 281
`ptrdiff_t`, 156
 Publications. *See* Books and publications.
 Purify tool, 143
 PurifyPlus tool, 143

Q

Quality assurance, software development, 298–302
 Quality management, software development, 306–307
 Quality requirements engineering, software development, 282–283

R

Race conditions, 248
 Race detection tools, 271–272
 RaceGuard, 272
 Randomization, 141–142
 Range checking, 188–189
 Ranges of integers, 157–158
 Rank, integer conversions, 160–161

- RATS (Rough Auditing Tool for Security), 295
- RCSint class, 195–196
- realloc() function, 98
- Red Hat Linux vulnerabilities, 2000 - 2004, 22
- Redeclaring stack buffers, 79–80
- Remote login, string vulnerabilities, 72
- Remote procedure call (RPC), 1–2
- Representation of integers, 152–153
- Restricting bytes written, 232–234
- Return-into-libc. *See* Arc injection.
- Risk assessment. *See* Threat assessment.
- rlogin program, 72
- Rough Auditing Tool for Security (RATS), 295
- RPC (remote procedure call), 1–2
- RTL (runtime linker), 83
- RtlHeap
 - buffer overflow, 126–132
 - CRT memory functions, 122
 - data structures
 - free lists, 123–125
 - look-aside lists, 125
 - memory chunks, 125–126
 - PEB (process environment block), 123
 - double-freed memory, 134–137
 - heap memory API, 122
 - local, global memory API, 122
 - look-aside table, 137–138
 - memory management, Win32, 120–123
 - memory-mapped file API, 122–123
 - virtual memory API, 122
 - writing to freed memory, 133–134
- Rules Builder, 296–297
- Runtime analysis tools, 143–145
- Runtime checks, 67
- Runtime linker (RTL), 83
- S**
- Safe integer operations, 191–196
- SafeInt class, 194–195
- SafeStr library, 62–63
- SCA (Source Code Analysis), 296–297
- Secure Coding Rulepacks, 296–297
- Secure Programming in C and C++*, 16
- Secure property checking, 264–266
- Secure wrappers, 289
- Security
 - actors *versus* artifacts, 11–12
 - computer, definition, 10
 - concepts, actions, and relationships, 11
 - damage costs. *See* Cost of damages.
 - developmental elements, 10
 - flaws
 - definition, 12–13
 - risks posed by, 13
 - string vulnerabilities, 34–35
 - operational elements, 10
 - review, software development, 301–302
 - risk assessment. *See* Threat assessment.
 - software development
 - principles
 - complete mediation, 278–279
 - economy of mechanism, 278
 - fail-safe defaults, 278
 - least common mechanism, 279–281
 - least privilege, 279–280
 - open design, 279
 - separation of privilege, 279
 - psychological acceptability, 281
 - summary of, 277
- Security analysts, definition, 12
- Security policies, definition, 12
- Security researcher, definition, 12
- Security Scanner, 295–296
- Security-enhanced functions, 233–234
- SEH (structured exception handling), 92–94
- Separation of privilege, 279
- Shalunov, Stanislav, 260
- Shared directories, 268
- Shared Object Property, 248
- Shared resources, 266–268
- short int, 155
- Sign errors, integer vulnerabilities, 183–184
- Sign errors, integers, 166–167
- signed char, 155
- Signed characters, integer conversions, 162–163
- Signed integers, 154–155
- Signed types, integer conversions, 161–162
- Signed-magnitude integers, 152–153
- size_t, 156
- snprintf(), 208–209
- Software components, definition, 10
- Software defects
 - definition, 13
 - risks posed by, 13
- Software development
 - architecture and design, 286–287
 - Audit Workbench, 296–297
 - black listing, 293–294
 - code audits, 300
 - compiler checks, 290
 - data sanitization, 292–295
 - defense in depth, 304
 - DEP (date execution prevention), 303–304
 - Flawfinder, 295
 - Fortify, 296–297
 - guidelines and checklists, 300–301

- independent security review, 301–302
- input validation, 291
- It's the Software Stupid!, 295–296
- life cycle, 276
- memory permissions, 302–304
- off-the-shelf software, 288–289
- PaX, 303
- planning, 305–306
- PREfast, 298
- PREFIX, 298
- Prevent, 297–298
- Prexis, 297
- quality assurance, 298–302
- quality management, 306–307
- quality requirements engineering, 282–283
- RATS (Rough Auditing Tool for Security), 295
- Rules Builder, 296–297
- SCA (Source Code Analysis), 296–297
- Secure Coding Rulepacks, 296–297
- secure wrappers, 289
- security principles
 - complete mediation, 278–279
 - economy of mechanism, 278
 - fail-safe defaults, 278
 - least common mechanism, 279–281
 - least privilege, 279–280
 - open design, 279
 - psychological acceptability, 281
 - separation of privilege, 279
 - summary of, 277
- Security Scanner, 295–296
- source code analysis, 295–298
- static analysis, 295–298
- testing, 294–295, 299–300
- threat modeling, 283–284
- tracking, 305–306
- TSP-Secure, 304–305
- use/misuse cases, 284–286
- vulnerabilities in existing code, 288
- W^X policy, 302–303
- white listing, 294
- Source code
 - analysis, 295–298
 - audit, 197
 - definition, 10–11
- Source Code Analysis (SCA), 296–297
- Spies. *See* Competitive intelligence professionals.
- printf(), 208
- SSP (Stack Smashing Protector), 70–71
- Stack content, viewing, 216–218
- Stack management, 37–39
- Stack pointer verification, 67
- Stack randomization, 225–230
- Stack segments, 36–37
- Stack smashing, 40–44, 69–70
- Stack Smashing Protector (SSP), 70–71
- stackgap, 68
- Standard integers, 153, 155–156
- Standards, C and C++, 16–17
- Static analysis, software development, 295–298
- Static analysis tools, 271
- Static binary analysis, 241–242
- Static prevention strategies, 51
- Static taint analysis, 237
- __stdcall convention, 207
- stdio versus iostream, 234
- std::string class, 61–62
- Storage. *See* Dynamic memory management; Memory; Process memory.
- strcat() function, 54–55
- strcat_s() function, 55–56
- strcpy() function, 54–55
- strcpy_s() function, 55–56
- String classes, 26
- String streams, 64–66
- String truncation, 32
- Strings
 - in C++, 26
 - common errors
 - deprecated functions, 32–33
 - null-termination errors, 31–32
 - off-by-one errors, 29–31
 - string truncation, 32
 - unbounded string copies, 27–29
 - error conditions
 - deprecated functions, 32–33
 - length errors, 26
 - null-termination errors, 31–32
 - off-by-one errors, 29–31
 - string truncation, 32
 - unbounded string copies, 27–29
 - length, 26
 - mitigation strategies
 - array bounds checking, 68–69
 - canaries, 69–70
 - concatenating strings, 54–61
 - copying strings, 54–61
 - detection and recover, 67–71
 - dynamic, 51
 - fgets() function, 52–54
 - gets_s() function, 52–54
 - input validation, 51–54
 - length calculation, 58–59
 - Libsafe library, 71
 - Libverify library, 71
 - memcpy_s() function, 54
 - memmove_s() function, 54
 - moving strings, 54–58
 - nonexecutable stacks, 67
 - overruns of local arrays, 67
 - ProPolice, 70–71
 - runtime checks, 67
 - safe handling functions, 59
 - SafeStr library, 62–63
 - SSP (Stack Smashing Protector), 70–71

- Strings, *continued*
- stack pointer verification, 67
 - stackgap, 68
 - static, 51
 - std::string class, 61–62
 - strcat() function, 54–55
 - strcat_s() function, 55–56
 - strcpy() function, 54–55
 - strcpy_s() function, 55–56
 - string streams, 64–66
 - strlcat() function, 55–56
 - strlcpy() function, 55–56
 - strlen() function, 58–59
 - strncat() function, 56–57
 - strncat_s() function, 57–58
 - strncpy() function, 56–57
 - strncpy_s() function, 57–58
 - Strsafe.h functions, 59
 - trusted *versus* untrusted data, 63
 - Vstr library, 63–64
- process memory
- code segments, 36–37
 - data segments, 36–37
 - frames, 37
 - heap segments, 36–37
 - organization, 36–37
 - stack management, 37–39
 - stack segments, 36–37
- stack smashing, 40–44, 69–70
- transferring program control
- arc injection, 48–51
 - code injection, 44–48
- value, 26
- vulnerabilities
- buffer overflows, 35–36
 - cryptography, 72–73
 - e-mail, 73–74
 - Kerberos, 72–73
 - metamail package, 73–74
 - MIME types, 73–74
 - password example, 33–34
 - remote login, 72
 - security flaws, 34–35
 - wide, 26
- strlcat() function, 55–56
- strlcpy() function, 55–56
- strlen() function, 58–59
- strncat() function, 56–57
- strncat_s() function, 57–58
- strncpy() function, 56–57
- strncpy_s() function, 57–58
- Strong typing, 189–190
- Stroustrup, Bjarne, 16–17
- Strsafe.h functions, 59
- Structured exception handling (SEH), 92–94
- Subtraction operations, 172–174
- svchost.exe, 3
- Symbolic linking exploits, 255–257
- Symlink exploits, 255–257
- Synchronization primitives, 249
- Synchronizing across processes, 254
- syslog(), 209
- System administrators, definition, 11
- System default exception handling, 94–95
- System integrators, definition, 11
- T**
- Temporary file open exploits, 257–260
- Temporary files, 268–269
- Terrorists, 8. *See also* Attackers.
- Testing
- formatted output, 234–235
 - fuzz, 299–300
 - integer errors, 196–197
 - penetration, 299
 - software development, 294–295, 299–300
- ThreadChecker, 272
- Thread-safe functions, 264
- Threat assessment
- competitive intelligence professionals, 8
 - crackers, 7
 - criminals, 7
 - hackers, 7
 - information warriors, 8–9
 - insiders, 7
 - potential attackers, 7–9
 - spies. *See* Competitive intelligence professionals.
 - terrorists, 8
- Threat modeling, 283–284
- tmp cleaners exploits, 260
- TOCTOU (time of check, time of use), 250–252
- Tracking software development, 305–306
- Trampolines, 132
- Transferring program control
- arc injection, 48–51
 - code injection, 44–48
- Troan, Erik, 260
- Truncation errors, integer vulnerabilities, 167, 184–186
- Trusted filenames exploits, 261
- Trusted *versus* untrusted data, 63
- Trusted/untrusted control flows, 250–252
- Trustworthy directories, 270
- TSP-Secure, 304–305
- Two's complement, 152–153
- Type, safety, 18
- Types of integers
- extended, 153, 155–156
 - platform-specific, 156
 - signed, 154–155
 - standard, 153, 155–156
 - unsigned, 154–155
- U**
- uintmax_t, 156
- uintptr_t, 156
- Unbounded string copies, 27–29
- UNIX System V varargs.h macros, 208

- unlink() race exploits, 260
 - Unlink technique, 111–114
 - Unsigned characters, integer conversions, 162–163
 - Unsigned integers, 154–155
 - Use/misuse cases, 284–286
- V**
- Valgrind tool, 144
 - varargs.h macros, 208
 - Variadic functions
 - description, 204–208
 - modifying, 237–239
 - VEH (vectored exception handling), 92
 - vfprintf(), 209
 - Viewing
 - memory content, 218–220
 - stack content, 216–218
 - Virtual functions, 87–88
 - Virtual memory API, 122
 - Virtual pointers, 87–88
 - Visual C++
 - compiler-generated runtime checks, 190
 - formatted output, 212–213
 - vprintf(), 209
 - vsprintf(), 209
 - Vstr library, 63–64
 - Vulnerabilities
 - definition, 13
 - description, 13–14
 - double-free, 117–120
 - e-mail, 73–74
 - in existing code, 288
 - format strings, 213–214
 - formatted output
 - buffer overflows, 214–215
 - CDE ToolTalk, 243
 - crashing programs, 216
 - format string, 213–214
 - internationalization, 224–225
 - output streams, 215–216
 - overwriting memory, 220–224
 - viewing memory content, 218–220
 - viewing stack content, 216–218
 - Washington University
 - FTP daemon, 242
 - heap-based exploits. *See also*
 - Dynamic memory management; RtlHeap.
 - CVS buffer overflow, 147
 - CVS double-free, 148
 - MDAC (Microsoft Data Access) components, 147–148
 - MIT Kerberos 5, 149
 - integers
 - Bash, 199–200
 - integer overflow, 182–183
 - integer overflow in XDR library, 197–198
 - sign errors, 183–184
 - truncation errors, 184–186
 - Windows DirectX MIDI Library, 198–199
 - internationalization, 224–225
 - Kerberos, 149
 - Linux, 2000 - 2004, 22
 - metamail package, 73–74
 - MIME types, 73–74
 - output stream, 215–216
 - password example, 33–34
 - Red Hat Linux, 2000 - 2004, 22
 - remote login, 72
 - reported to CERT/CC, 9–10
 - security flaws, 34–35
 - strings
 - buffer overflows, 35–36
 - cryptography, 72–73
 - e-mail, 73–74
 - Kerberos, 72–73
 - metamail package, 73–74
 - MIME types, 73–74
 - password example, 33–34
 - remote login, 72
 - security flaws, 34–35
 - Windows, 1996 - 2005, 20
 - Windows DirectX MIDI Library, 198–199
 - Vulnerability analysts, 12
- W**
- W^X policy, 95, 302–303
 - W32.Blaster.Worm, 1–4
 - Warlock, 271
 - Washington University FTP daemon, 242
 - wchar_t, 156
 - wformat flag, 236
 - wformat-nonliteral flag, 236
 - wformat-security flag, 236
 - White listing, 294
 - Wide strings, 26
 - Width, 211
 - Windows DirectX MIDI Library, integer vulnerabilities, 198–199
 - Windows XP SP2, heap-based exploit mitigation, 145–146
 - Writing addresses in two words, 227
 - Writing to freed memory, 120, 133–134
- X**
- Xfocus, 2
- Z**
- Zalewski, Michal, 260