SECOND EDITION

COMPUTER SECURITY

[ART and SCIENCE]

MATT BISHOP

With contributions from ELISABETH SULLIVAN *and* MICHELLE RUPPEL

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Computer Security

Second Edition

This page intentionally left blank

Computer Security
Art and Science

Second Edition

Matt Bishop
with contributions from
Elisabeth Sullivan and Michelle Ruppel

Addison-Wesley

Boston • Columbus • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2018950017

Copyright © 2019 Pearson Education, Inc.

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearsoned.com/permissions/.

ISBN-13: 978-0-321-71233-2
ISBN-10: 0-321-71233-1

1 18

To my dear Holly; our children Heidi, Steven, David, and Caroline; our grandchildren Skyler and Sage; and our friends Seaview, Tinker Belle, Stripe, Baby Windsor, Scout, Fur, Puff, Mouse, Shadow, Fuzzy, Dusty, and the rest of the menagerie.

This page intentionally left blank

Contents

Preface . . . . **xxix**
Acknowledgments . . . . **xlv**
About the Author . . . . **xlix**

PART I : INTRODUCTION **1**

Chapter 1 An Overview of Computer Security **3**
1.1 The Basic Components 3
  1.1.1 Confidentiality 4
  1.1.2 Integrity 5
  1.1.3 Availability 6
1.2 Threats 6
1.3 Policy and Mechanism 9
  1.3.1 Goals of Security 10
1.4 Assumptions and Trust 11
1.5 Assurance 12
  1.5.1 Specification 14
  1.5.2 Design 14
  1.5.3 Implementation 15
1.6 Operational Issues 16
  1.6.1 Cost-Benefit Analysis 16
  1.6.2 Risk Analysis 17
  1.6.3 Laws and Customs 19
1.7 Human Issues 20
  1.7.1 Organizational Problems 20
  1.7.2 People Problems 21
1.8 Tying It All Together 22
1.9 Summary 24
1.10 Research Issues 24
1.11 Further Reading 25
1.12 Exercises 25

PART II : FOUNDATIONS 29

Chapter 2 Access Control Matrix 31
2.1 Protection State 31
2.2 Access Control Matrix Model 32
    2.2.1 Access Control by Boolean Expression Evaluation 35
    2.2.2 Access Controlled by History 36
2.3 Protection State Transitions 37
    2.3.1 Conditional Commands 40
2.4 Copying, Owning, and the Attenuation of Privilege 42
    2.4.1 Copy Right 42
    2.4.2 Own Right 42
    2.4.3 Principle of Attenuation of Privilege 43
2.5 Summary 44
2.6 Research Issues 44
2.7 Further Reading 44
2.8 Exercises 45

Chapter 3 Foundational Results 49
3.1 The General Question 49
3.2 Basic Results 51
3.3 The Take-Grant Protection Model 56
    3.3.1 Sharing of Rights 57
    3.3.2 Interpretation of the Model 61
    3.3.3 Theft in the Take-Grant Protection Model 62
    3.3.4 Conspiracy 66
    3.3.5 Summary 68
3.4 Closing the Gap: The Schematic Protection Model 68
    3.4.1 Link Predicate 69
    3.4.2 Filter Function 70
    3.4.3 Putting It All Together 71
    3.4.4 Demand and Create Operations 72
    3.4.5 Safety Analysis 75
3.5 Expressive Power and the Models 81
    3.5.1 Brief Comparison of HRU and SPM 82
    3.5.2 Extending SPM 83
    3.5.3 Simulation and Expressiveness 88
    3.5.4 Typed Access Matrix Model 92
3.6 Comparing Security Properties of Models 94
    3.6.1 Comparing Schemes and Security Properties 95
    3.6.2 Augmented Typed Access Matrix Model 99

3.7 Summary . . . 101
3.8 Research Issues . . . 102
3.9 Further Reading . . . 102
3.10 Exercises . . . 103

PART III : POLICY 107

Chapter 4 Security Policies . . . 109
4.1 The Nature of Security Policies . . . 109
4.2 Types of Security Policies . . . 113
4.3 The Role of Trust . . . 115
4.4 Types of Access Control . . . 117
4.5 Policy Languages . . . 118
  4.5.1 High-Level Policy Languages . . . 119
  4.5.2 Low-Level Policy Languages . . . 125
4.6 Example: Academic Computer Security Policy . . . 126
  4.6.1 General University Electronic Communications Policy . . . 127
  4.6.2 Implementation at UC Davis . . . 130
4.7 Security and Precision . . . 131
4.8 Summary . . . 136
4.9 Research Issues . . . 136
4.10 Further Reading . . . 137
4.11 Exercises . . . 138

Chapter 5 Confidentiality Policies . . . 141
5.1 Goals of Confidentiality Policies . . . 141
5.2 The Bell-LaPadula Model . . . 142
  5.2.1 Informal Description . . . 142
  5.2.2 Example: Trusted Solaris . . . 146
  5.2.3 Formal Model . . . 151
  5.2.4 Example Model Instantiation: Multics . . . 158
5.3 Tranquility . . . 161
  5.3.1 Declassification Principles . . . 163
5.4 The Controversy over the Bell-LaPadula Model . . . 164
  5.4.1 McLean's †-Property and the Basic Security Theorem . . . 164
  5.4.2 McLean's System Z and More Questions . . . 166
5.5 Summary . . . 169
5.6 Research Issues . . . 169
5.7 Further Reading . . . 170
5.8 Exercises . . . 171

Chapter 6 Integrity Policies **173**

6.1 Goals 173
6.2 The Biba Model 175
  6.2.1 Low-Water-Mark Policy 176
  6.2.2 Ring Policy 177
  6.2.3 Biba's Model (Strict Integrity Policy) 177
6.3 Lipner's Integrity Matrix Model 178
  6.3.1 Lipner's Use of the Bell-LaPadula Model 178
  6.3.2 Lipner's Full Model 181
  6.3.3 Comparison with Biba 182
6.4 Clark-Wilson Integrity Model 183
  6.4.1 The Model 184
  6.4.2 Comparison with the Requirements 187
  6.4.3 Comparison with Other Models 188
6.5 Trust Models 189
  6.5.1 Policy-Based Trust Management 191
  6.5.2 Reputation-Based Trust Management 194
6.6 Summary 196
6.7 Research Issues 196
6.8 Further Reading 197
6.9 Exercises 198

Chapter 7 Availability Policies **201**

7.1 Goals of Availability Policies 201
7.2 Deadlock 202
7.3 Denial of Service Models 203
  7.3.1 Constraint-Based Model 204
  7.3.2 State-Based Modes 210
7.4 Example: Availability and Network Flooding 215
  7.4.1 Analysis 216
  7.4.2 Intermediate Systems 216
  7.4.3 TCP State and Memory Allocations 218
  7.4.4 Other Flooding Attacks 221
7.5 Summary 222
7.6 Research Issues 222
7.7 Further Reading 223
7.8 Exercises 224

Chapter 8 Hybrid Policies **227**

8.1 Chinese Wall Model 227
  8.1.1 Informal Description 228
  8.1.2 Formal Model 230

8.1.3 Aggressive Chinese Wall Model 233
8.1.4 Bell-LaPadula and Chinese Wall Models 234
8.1.5 Clark-Wilson and Chinese Wall Models 236
8.2 Clinical Information Systems Security Policy 236
8.2.1 Bell-LaPadula and Clark-Wilson Models 239
8.3 Originator Controlled Access Control 239
8.3.1 Digital Rights Management 241
8.4 Role-Based Access Control 244
8.5 Break-the-Glass Policies 249
8.6 Summary 250
8.7 Research Issues 250
8.8 Further Reading 251
8.9 Exercises 252

Chapter 9 Noninterference and Policy Composition 255

9.1 The Problem 255
9.1.1 Composition of Bell-LaPadula Models 256
9.2 Deterministic Noninterference 259
9.2.1 Unwinding Theorem 263
9.2.2 Access Control Matrix Interpretation 266
9.2.3 Security Policies That Change over Time 268
9.2.4 Composition of Deterministic Noninterference-Secure Systems 270
9.3 Nondeducibility 271
9.3.1 Composition of Deducibly Secure Systems 273
9.4 Generalized Noninterference 274
9.4.1 Composition of Generalized Noninterference Systems 275
9.5 Restrictiveness 277
9.5.1 State Machine Model 277
9.5.2 Composition of Restrictive Systems 279
9.6 Side Channels and Deducibility 280
9.7 Summary 282
9.8 Research Issues 283
9.9 Further Reading 283
9.10 Exercises 285

PART IV : IMPLEMENTATION I: CRYPTOGRAPHY 287

Chapter 10 Basic Cryptography 289

10.1 Cryptography 289
10.1.1 Overview of Cryptanalysis 290

10.2 Symmetric Cryptosystems . . . 291
- 10.2.1 Transposition Ciphers . . . 291
- 10.2.2 Substitution Ciphers . . . 292
- 10.2.3 Data Encryption Standard . . . 299
- 10.2.4 Other Modern Symmetric Ciphers . . . 302
- 10.2.5 Advanced Encryption Standard . . . 303

10.3 Public Key Cryptography . . . 306
- 10.3.1 El Gamal . . . 307
- 10.3.2 RSA . . . 309
- 10.3.3 Elliptic Curve Ciphers . . . 312

10.4 Cryptographic Checksums . . . 315
- 10.4.1 HMAC . . . 317

10.5 Digital Signatures . . . 318
- 10.5.1 Symmetric Key Signatures . . . 319
- 10.5.2 Public Key Signatures . . . 319

10.6 Summary . . . 323

10.7 Research Issues . . . 324

10.8 Further Reading . . . 325

10.9 Exercises . . . 326

Chapter 11 Key Management . . . 331

11.1 Session and Interchange Keys . . . 332

11.2 Key Exchange . . . 332
- 11.2.1 Symmetric Cryptographic Key Exchange . . . 333
- 11.2.2 Kerberos . . . 337
- 11.2.3 Public Key Cryptographic Key Exchange and Authentication . . . 338

11.3 Key Generation . . . 341

11.4 Cryptographic Key Infrastructures . . . 343
- 11.4.1 Merkle’s Tree Authentication Scheme . . . 344
- 11.4.2 Certificate Signature Chains . . . 346
- 11.4.3 Public Key Infrastructures . . . 350

11.5 Storing and Revoking Keys . . . 353
- 11.5.1 Key Storage . . . 353
- 11.5.2 Key Revocation . . . 358

11.6 Summary . . . 359

11.7 Research Issues . . . 360

11.8 Further Reading . . . 361

11.9 Exercises . . . 362

Chapter 12 Cipher Techniques 367

12.1 Problems 367
  12.1.1 Precomputing the Possible Messages 367
  12.1.2 Misordered Blocks 368
  12.1.3 Statistical Regularities 368
  12.1.4 Type Flaw Attacks 369
  12.1.5 Summary 370
12.2 Stream and Block Ciphers 370
  12.2.1 Stream Ciphers 371
  12.2.2 Block Ciphers 374
12.3 Authenticated Encryption 377
  12.3.1 Counter with CBC-MAC Mode 377
  12.3.2 Galois Counter Mode 379
12.4 Networks and Cryptography 381
12.5 Example Protocols 384
  12.5.1 Secure Electronic Mail: PEM and OpenPGP 384
  12.5.2 Instant Messaging 389
  12.5.3 Security at the Transport Layer: TLS and SSL 393
  12.5.4 Security at the Network Layer: IPsec 402
  12.5.5 Conclusion 410
12.6 Summary 410
12.7 Research Issues 411
12.8 Further Reading 411
12.9 Exercises 413

Chapter 13 Authentication 415

13.1 Authentication Basics 415
13.2 Passwords 416
13.3 Password Selection 418
  13.3.1 Random Selection of Passwords 418
  13.3.2 Pronounceable and Other Computer-Generated Passwords 420
  13.3.3 User Selection of Passwords 421
  13.3.4 Graphical Passwords 425
13.4 Attacking Passwords 426
  13.4.1 Off-Line Dictionary Attacks 428
  13.4.2 On-Line Dictionary Attacks 430
  13.4.3 Password Strength 432

13.5 Password Aging 434
 13.5.1 One-Time Passwords 436
13.6 Challenge-Response 438
 13.6.1 Pass Algorithms 438
 13.6.2 Hardware-Supported Challenge-Response Procedures 439
 13.6.3 Challenge-Response and Dictionary Attacks 439
13.7 Biometrics 441
 13.7.1 Fingerprints 442
 13.7.2 Voices 443
 13.7.3 Eyes 443
 13.7.4 Faces 444
 13.7.5 Keystrokes 444
 13.7.6 Combinations 445
13.8 Location 445
13.9 Multifactor Authentication 446
13.10 Summary 448
13.11 Research Issues 449
13.12 Further Reading 450
13.13 Exercises 451

PART V : IMPLEMENTATION II: SYSTEMS 453

Chapter 14 Design Principles 455

14.1 Underlying Ideas 455
14.2 Principles of Secure Design 457
 14.2.1 Principle of Least Privilege 457
 14.2.2 Principle of Fail-Safe Defaults 458
 14.2.3 Principle of Economy of Mechanism 459
 14.2.4 Principle of Complete Mediation 460
 14.2.5 Principle of Open Design 461
 14.2.6 Principle of Separation of Privilege 463
 14.2.7 Principle of Least Common Mechanism 463
 14.2.8 Principle of Least Astonishment 464
14.3 Summary 466
14.4 Research Issues 466
14.5 Further Reading 467
14.6 Exercises 468

Chapter 15 Representing Identity 471

15.1 What Is Identity? 471
15.2 Files and Objects 472
15.3 Users 473

15.4 Groups and Roles 475
15.5 Naming and Certificates 476
  15.5.1 Conflicts 479
  15.5.2 The Meaning of the Identity 481
  15.5.3 Trust 482
15.6 Identity on the Web 484
  15.6.1 Host Identity 484
  15.6.2 State and Cookies 488
15.7 Anonymity on the Web 490
  15.7.1 Email Anonymizers 491
  15.7.2 Onion Routing 495
15.8 Summary 501
15.9 Research Issues 502
15.10 Further Reading 503
15.11 Exercises 504

Chapter 16 Access Control Mechanisms 507

16.1 Access Control Lists 507
  16.1.1 Abbreviations of Access Control Lists 508
  16.1.2 Creation and Maintenance of Access Control Lists 511
  16.1.3 Revocation of Rights 514
  16.1.4 Example: NTFS and Access Control Lists 515
16.2 Capabilities 518
  16.2.1 Implementation of Capabilities 519
  16.2.2 Copying and Amplifying Capabilities 520
  16.2.3 Revocation of Rights 522
  16.2.4 Limits of Capabilities 522
  16.2.5 Comparison with Access Control Lists 523
  16.2.6 Privileges 524
16.3 Locks and Keys 526
  16.3.1 Type Checking 528
  16.3.2 Sharing Secrets 529
16.4 Ring-Based Access Control 531
16.5 Propagated Access Control Lists 533
16.6 Summary 535
16.7 Research Issues 535
16.8 Further Reading 536
16.9 Exercises 536

Chapter 17 Information Flow 539

17.1 Basics and Background 539
  17.1.1 Entropy-Based Analysis 540
  17.1.2 Information Flow Models and Mechanisms 541

17.2 Nonlattice Information Flow Policies 542
  17.2.1 Confinement Flow Model 543
  17.2.2 Transitive Nonlattice Information Flow Policies 544
  17.2.3 Nontransitive Information Flow Policies 545
17.3 Static Mechanisms 548
  17.3.1 Declarations 549
  17.3.2 Program Statements 550
  17.3.3 Exceptions and Infinite Loops 557
  17.3.4 Concurrency 558
  17.3.5 Soundness 561
17.4 Dynamic Mechanisms 562
  17.4.1 Fenton's Data Mark Machine 562
  17.4.2 Variable Classes 565
17.5 Integrity Mechanisms 566
17.6 Example Information Flow Controls 567
  17.6.1 Privacy and Android Cell Phones 568
  17.6.2 Firewalls 570
17.7 Summary 574
17.8 Research Issues 574
17.9 Further Reading 575
17.10 Exercises 576

Chapter 18 Confinement Problem 579

18.1 The Confinement Problem 579
18.2 Isolation 582
  18.2.1 Controlled Environment 582
  18.2.2 Program Modification 590
18.3 Covert Channels 594
  18.3.1 Detection of Covert Channels 596
  18.3.2 Analysis of Covert Channels 610
  18.3.3 Mitigation of Covert Channels 616
18.4 Summary 619
18.5 Research Issues 620
18.6 Further Reading 620
18.7 Exercises 622

PART VI : ASSURANCE 625
Contributed by Elisabeth Sullivan and Michelle Ruppel

Chapter 19 Introduction to Assurance 627

19.1 Assurance and Trust 627
  19.1.1 The Need for Assurance 629

- 19.1.2 The Role of Requirements in Assurance . . . 631
- 19.1.3 Assurance throughout the Life Cycle . . . 632
- 19.2 Building Secure and Trusted Systems . . . 634
  - 19.2.1 Life Cycle . . . 634
  - 19.2.2 The Waterfall Life Cycle Model . . . 639
  - 19.2.3 Agile Software Development . . . 641
  - 19.2.4 Other Models of Software Development . . . 644
- 19.3 Summary . . . 645
- 19.4 Research Issues . . . 645
- 19.5 Further Reading . . . 646
- 19.6 Exercises . . . 647

Chapter 20 Building Systems with Assurance . . . 649

- 20.1 Assurance in Requirements Definition and Analysis . . . 649
  - 20.1.1 Threats and Security Objectives . . . 650
  - 20.1.2 Architectural Considerations . . . 651
  - 20.1.3 Policy Definition and Requirements Specification . . . 657
  - 20.1.4 Justifying Requirements . . . 660
- 20.2 Assurance during System and Software Design . . . 662
  - 20.2.1 Design Techniques That Support Assurance . . . 662
  - 20.2.2 Design Document Contents . . . 665
  - 20.2.3 Building Documentation and Specification . . . 675
  - 20.2.4 Justifying That Design Meets Requirements . . . 677
- 20.3 Assurance in Implementation and Integration . . . 685
  - 20.3.1 Implementation Considerations That Support Assurance . . . 685
  - 20.3.2 Assurance through Implementation Management . . . 686
  - 20.3.3 Justifying That the Implementation Meets the Design . . . 687
- 20.4 Assurance during Operation and Maintenance . . . 695
- 20.5 Summary . . . 696
- 20.6 Research Issues . . . 696
- 20.7 Further Reading . . . 697
- 20.8 Exercises . . . 698

Chapter 21 Formal Methods . . . 699

- 21.1 Formal Verification Techniques . . . 699
- 21.2 Formal Specification . . . 702
- 21.3 Early Formal Verification Techniques . . . 705
  - 21.3.1 The Hierarchical Development Methodology . . . 705
  - 21.3.2 Enhanced HDM . . . 710
  - 21.3.3 The Gypsy Verification Environment . . . 711

21.4 Current Verification Systems 713
 21.4.1 The Prototype Verification System 713
 21.4.2 The Symbolic Model Verifier 716
 21.4.3 The Naval Research Laboratory Protocol Analyzer 720
21.5 Functional Programming Languages 721
21.6 Formally Verified Products 722
21.7 Summary 723
21.8 Research Issues 724
21.9 Further Reading 725
21.10 Exercises 725

Chapter 22 Evaluating Systems 727

22.1 Goals of Formal Evaluation 727
 22.1.1 Deciding to Evaluate 728
 22.1.2 Historical Perspective of Evaluation Methodologies 729
22.2 TCSEC: 1983–1999 730
 22.2.1 TCSEC Requirements 731
 22.2.2 The TCSEC Evaluation Classes 733
 22.2.3 The TCSEC Evaluation Process 734
 22.2.4 Impacts 735
22.3 International Efforts and the ITSEC: 1991–2001 737
 22.3.1 ITSEC Assurance Requirements 739
 22.3.2 The ITSEC Evaluation Levels 740
 22.3.3 The ITSEC Evaluation Process 741
 22.3.4 Impacts 741
22.4 Commercial International Security Requirements: 1991 742
 22.4.1 CISR Requirements 743
 22.4.2 Impacts 743
22.5 Other Commercial Efforts: Early 1990s 744
22.6 The Federal Criteria: 1992 744
 22.6.1 FC Requirements 745
 22.6.2 Impacts 745
22.7 FIPS 140: 1994–Present 746
 22.7.1 FIPS 140 Requirements 746
 22.7.2 FIPS 140-2 Security Levels 747
 22.7.3 Additional FIPS 140-2 Documentation 748
 22.7.4 Impact 748
 22.7.5 Future 749
22.8 The Common Criteria: 1998–Present 749
 22.8.1 Overview of the Methodology 751
 22.8.2 CC Requirements 756
 22.8.3 CC Security Functional Requirements 756

22.8.4 Assurance Requirements 759
22.8.5 Evaluation Assurance Levels 759
22.8.6 Evaluation Process 761
22.8.7 Other International Organizations 762
22.8.8 Impacts 763
22.8.9 Future of the Common Criteria 764
22.9 SSE-CMM: 1997–Present 765
22.9.1 The SSE-CMM Model 765
22.9.2 Using the SSE-CMM 767
22.10 Summary 768
22.11 Research Issues 769
22.12 Further Reading 769
22.13 Exercises 770

PART VII : SPECIAL TOPICS 773

Chapter 23 Malware 775

23.1 Introduction 775
23.2 Trojan Horses 776
23.2.1 Rootkits 777
23.2.2 Propagating Trojan Horses 779
23.3 Computer Viruses 780
23.3.1 Infection Vectors 782
23.3.2 Concealment 785
23.3.3 Summary 790
23.4 Computer Worms 790
23.5 Bots and Botnets 793
23.6 Other Malware 796
23.6.1 Rabbits and Bacteria 796
23.6.2 Logic Bombs 797
23.6.3 Adware 797
23.6.4 Spyware 799
23.6.5 Ransomware 800
23.6.6 Phishing 802
23.7 Combinations 803
23.8 Theory of Computer Viruses 803
23.9 Defenses 808
23.9.1 Scanning Defenses 808
23.9.2 Data and Instructions 811
23.9.3 Containment 812
23.9.4 Specifications as Restrictions 817

23.9.5 Limiting Sharing . . . 817
23.9.6 Statistical Analysis . . . 819
23.9.7 The Notion of Trust . . . 819
23.10 Summary . . . 820
23.11 Research Issues . . . 820
23.12 Further Reading . . . 821
23.13 Exercises . . . 822

Chapter 24 Vulnerability Analysis . . . 825

24.1 Introduction . . . 825
24.2 Penetration Studies . . . 827
24.2.1 Goals . . . 827
24.2.2 Layering of Tests . . . 828
24.2.3 Methodology at Each Layer . . . 829
24.2.4 Flaw Hypothesis Methodology . . . 830
24.2.5 Versions . . . 833
24.2.6 Example: Penetration of the Michigan Terminal System . . . 837
24.2.7 Example: Compromise of a Burroughs System . . . 839
24.2.8 Example: Penetration of a Corporate Computer System . . . 840
24.2.9 Example: Penetrating a UNIX System . . . 841
24.2.10 Example: Penetrating a Windows System . . . 843
24.2.11 Debate . . . 844
24.2.12 Conclusion . . . 845
24.3 Vulnerability Classification . . . 845
24.3.1 Two Security Flaws . . . 846
24.4 Frameworks . . . 849
24.4.1 The RISOS Study . . . 849
24.4.2 Protection Analysis Model . . . 851
24.4.3 The NRL Taxonomy . . . 857
24.4.4 Aslam's Model . . . 859
24.4.5 Comparison and Analysis . . . 860
24.5 Standards . . . 864
24.5.1 Common Vulnerabilities and Exposures (CVE) . . . 864
24.5.2 Common Weaknesses and Exposures (CWE) . . . 866
24.6 Gupta and Gligor's Theory of Penetration Analysis . . . 868
24.6.1 The Flow-Based Model of Penetration Analysis . . . 869
24.6.2 The Automated Penetration Analysis Tool . . . 872
24.6.3 Discussion . . . 873
24.7 Summary . . . 873
24.8 Research Issues . . . 874
24.9 Further Reading . . . 875
24.10 Exercises . . . 876

Chapter 25 Auditing . . . 879

25.1 Definition . . . 879
25.2 Anatomy of an Auditing System . . . 880
  25.2.1 Logger . . . 881
  25.2.2 Analyzer . . . 883
  25.2.3 Notifier . . . 883
25.3 Designing an Auditing System . . . 884
  25.3.1 Implementation Considerations . . . 886
  25.3.2 Syntactic Issues . . . 887
  25.3.3 Log Sanitization . . . 888
  25.3.4 Application and System Logging . . . 891
25.4 A Posteriori Design . . . 893
  25.4.1 Auditing to Detect Violations of a Known Policy . . . 893
  25.4.2 Auditing to Detect Known Violations of a Policy . . . 895
25.5 Auditing Mechanisms . . . 897
  25.5.1 Secure Systems . . . 897
  25.5.2 Nonsecure Systems . . . 899
25.6 Examples: Auditing File Systems . . . 900
  25.6.1 Audit Analysis of the NFS Version 2 Protocol . . . 900
  25.6.2 The Logging and Auditing File System (LAFS) . . . 905
  25.6.3 Comparison . . . 907
  25.6.4 Audit Browsing . . . 908
25.7 Summary . . . 910
25.8 Research Issues . . . 911
25.9 Further Reading . . . 912
25.10 Exercises . . . 913

Chapter 26 Intrusion Detection . . . 917

26.1 Principles . . . 917
26.2 Basic Intrusion Detection . . . 918
26.3 Models . . . 920
  26.3.1 Anomaly Modeling . . . 920
  26.3.2 Misuse Modeling . . . 932
  26.3.3 Specification Modeling . . . 938
  26.3.4 Summary . . . 941
26.4 Architecture . . . 942
  26.4.1 Agent . . . 943
  26.4.2 Director . . . 945
  26.4.3 Notifier . . . 946
26.5 Organization of Intrusion Detection Systems . . . 948
  26.5.1 Monitoring Network Traffic for Intrusions: NSM . . . 948
  26.5.2 Combining Host and Network Monitoring: DIDS . . . 949
  26.5.3 Autonomous Agents: AAFID . . . 952

26.6 Summary 954
26.7 Research Issues 954
26.8 Further Reading 955
26.9 Exercises 956

Chapter 27 Attacks and Responses 959

27.1 Attacks 959
27.2 Representing Attacks 960
  27.2.1 Attack Trees 961
  27.2.2 The Requires/Provides Model 965
  27.2.3 Attack Graphs 969
27.3 Intrusion Response 971
  27.3.1 Incident Prevention 971
  27.3.2 Intrusion Handling 975
27.4 Digital Forensics 987
  27.4.1 Principles 987
  27.4.2 Practice 990
  27.4.3 Anti-Forensics 994
27.5 Summary 996
27.6 Research Issues 997
27.7 Further Reading 998
27.8 Exercises 999

PART VIII : PRACTICUM 1003

Chapter 28 Network Security 1005

28.1 Introduction 1005
28.2 Policy Development 1006
  28.2.1 Data Classes 1007
  28.2.2 User Classes 1008
  28.2.3 Availability 1010
  28.2.4 Consistency Check 1010
28.3 Network Organization 1011
  28.3.1 Analysis of the Network Infrastructure 1013
  28.3.2 In the DMZ 1017
  28.3.3 In the Internal Network 1021
  28.3.4 General Comment on Assurance 1025
28.4 Availability 1026

28.5 Anticipating Attacks 1027
28.6 Summary 1028
28.7 Research Issues 1028
28.8 Further Reading 1029
28.9 Exercises 1030

Chapter 29 System Security 1035

29.1 Introduction 1035
29.2 Policy 1036
  29.2.1 The WWW Server System in the DMZ 1036
  29.2.2 The Development System 1037
  29.2.3 Comparison 1041
  29.2.4 Conclusion 1041
29.3 Networks 1042
  29.3.1 The WWW Server System in the DMZ 1042
  29.3.2 The Development System 1045
  29.3.3 Comparison 1047
29.4 Users 1048
  29.4.1 The WWW Server System in the DMZ 1048
  29.4.2 The Development System 1050
  29.4.3 Comparison 1052
29.5 Authentication 1053
  29.5.1 The WWW Server System in the DMZ 1053
  29.5.2 Development Network System 1054
  29.5.3 Comparison 1055
29.6 Processes 1055
  29.6.1 The WWW Server System in the DMZ 1055
  29.6.2 The Development System 1059
  29.6.3 Comparison 1060
29.7 Files 1061
  29.7.1 The WWW Server System in the DMZ 1061
  29.7.2 The Development System 1063
  29.7.3 Comparison 1065
29.8 Retrospective 1066
  29.8.1 The WWW Server System in the DMZ 1066
  29.8.2 The Development System 1067
29.9 Summary 1068
29.10 Research Issues 1068
29.11 Further Reading 1069
29.12 Exercises 1070

Chapter 30 User Security 1073

30.1 Policy 1073
30.2 Access 1074
  30.2.1 Passwords 1074
  30.2.2 The Login Procedure 1076
  30.2.3 Leaving the System 1079
30.3 Files and Devices 1080
  30.3.1 Files 1080
  30.3.2 Devices 1084
30.4 Processes 1087
  30.4.1 Copying and Moving Files 1087
  30.4.2 Accidentally Overwriting Files 1088
  30.4.3 Encryption, Cryptographic Keys, and Passwords 1089
  30.4.4 Startup Settings 1090
  30.4.5 Limiting Privileges 1091
  30.4.6 Malicious Logic 1091
30.5 Electronic Communications 1092
  30.5.1 Automated Electronic Mail Processing 1092
  30.5.2 Failure to Check Certificates 1093
  30.5.3 Sending Unexpected Content 1094
30.6 Summary 1094
30.7 Research Issues 1095
30.8 Further Reading 1095
30.9 Exercises 1096

Chapter 31 Program Security 1099

31.1 Problem 1099
31.2 Requirements and Policy 1100
  31.2.1 Requirements 1100
  31.2.2 Threats 1102
31.3 Design 1104
  31.3.1 Framework 1104
  31.3.2 Access to Roles and Commands 1106
31.4 Refinement and Implementation 1111
  31.4.1 First-Level Refinement 1111
  31.4.2 Second-Level Refinement 1112
  31.4.3 Functions 1114
  31.4.4 Summary 1117
31.5 Common Security-Related Programming Problems 1117
  31.5.1 Improper Choice of Initial Protection Domain 1118
  31.5.2 Improper Isolation of Implementation Detail 1123

31.5.3 Improper Change 1125
31.5.4 Improper Naming 1129
31.5.5 Improper Deallocation or Deletion 1131
31.5.6 Improper Validation 1132
31.5.7 Improper Indivisibility 1138
31.5.8 Improper Choice of Operand or Operation 1139
31.5.9 Summary 1141
31.6 Testing, Maintenance, and Operation 1141
31.6.1 Testing 1142
31.6.2 Testing Composed Modules 1145
31.6.3 Testing the Program 1145
31.7 Distribution 1146
31.8 Summary 1147
31.9 Research Issues 1147
31.10 Further Reading 1148
31.11 Exercises 1148

PART IX : APPENDICES **1151**

Appendix A Lattices 1153

A.1 Basics 1153
A.2 Lattices 1154
A.3 Exercises 1155

Appendix B The Extended Euclidean Algorithm 1157

B.1 The Euclidean Algorithm 1157
B.2 The Extended Euclidean Algorithm 1158
B.3 Solving $ax \bmod n = 1$ 1160
B.4 Solving $ax \bmod n = b$ 1161
B.5 Exercises 1161

Appendix C Entropy and Uncertainty 1163

C.1 Conditional and Joint Probability 1163
C.2 Entropy and Uncertainty 1165
C.3 Joint and Conditional Entropy 1166
C.3.1 Joint Entropy 1166
C.3.2 Conditional Entropy 1167
C.3.3 Perfect Secrecy 1168
C.4 Exercises 1169

Appendix D Virtual Machines 1171
D.1 Virtual Machine Structure 1171
D.2 Virtual Machine Monitor 1171
  D.2.1 Privilege and Virtual Machines 1172
  D.2.2 Physical Resources and Virtual Machines 1175
  D.2.3 Paging and Virtual Machines 1175
D.3 Exercises 1176

Appendix E Symbolic Logic 1179
E.1 Propositional Logic 1179
  E.1.1 Natural Deduction in Propositional Logic 1180
  E.1.2 Rules 1180
  E.1.3 Derived Rules 1181
  E.1.4 Well-Formed Formulas 1182
  E.1.5 Truth Tables 1182
  E.1.6 Mathematical Induction 1183
E.2 Predicate Logic 1184
  E.2.1 Natural Deduction in Predicate Logic 1185
E.3 Temporal Logic Systems 1186
  E.3.1 Syntax of CTL 1186
  E.3.2 Semantics of CTL 1186
E.4 Exercises 1188

Appendix F The Encryption Standards 1191
F.1 Data Encryption Standard 1191
  F.1.1 Main DES Algorithm 1191
  F.1.2 Round Key Generation 1195
F.2 Advanced Encryption Standard 1196
  F.2.1 Background 1196
  F.2.2 AES Encryption 1197
  F.2.3 Encryption 1199
  F.2.4 Round Key Generation 1201
  F.2.5 Equivalent Inverse Cipher Implementation 1203
F.3 Exercises 1205

Appendix G Example Academic Security Policy 1207
G.1 Acceptable Use Policy 1207
  G.1.1 Introduction 1208
  G.1.2 Rights and Responsibilities 1208
  G.1.3 Privacy 1208

G.1.4 Enforcement of Laws and University Policies 1209
G.1.5 Unacceptable Conduct 1209
G.1.6 Further Information 1212
G.2 University of California Electronic Communications Policy 1212
G.2.1 Introduction 1212
G.2.2 General Provisions 1213
G.2.3 Allowable Use 1216
G.2.4 Privacy and Confidentiality 1220
G.2.5 Security 1225
G.2.6 Retention and Disposition 1227
G.2.7 Appendix A: Definitions 1227
G.2.8 Appendix B: References 1230
G.2.9 Appendix C: Policies Relating to Access Without Consent 1232
G.3 User Advisories 1234
G.3.1 Introduction 1234
G.3.2 User Responsibilities 1234
G.3.3 Privacy Expectations 1235
G.3.4 Privacy Protections 1236
G.3.5 Privacy Limits 1237
G.3.6 Security Considerations 1239
G.4 Electronic Communications—Allowable Use 1241
G.4.1 Purpose 1241
G.4.2 Definitions 1242
G.4.3 Policy 1242
G.4.4 Allowable Users 1242
G.4.5 Allowable Uses 1243
G.4.6 Restrictions on Use 1245
G.4.7 References and Related Policies 1246

Appendix H Programming Rules 1247

H.1 Implementation Rules 1247
H.2 Management Rules 1249

References 1251

Index 1341

This page intentionally left blank

Preface

HORTENSIO: Madam, before you touch the instrument
To learn the order of my fingering,
I must begin with rudiments of art
To teach you gamouth in a briefer sort,
More pleasant, pithy and effectual,
Than hath been taught by any of my trade;
And there it is in writing, fairly drawn.
— *The Taming of the Shrew*, III, i, 62–68.

Preface to the Second Edition

Since the first edition of this book was published, the number of computer and information security incidents has increased dramatically, as has their seriousness. In 2010, a computer worm infected the software controlling a particular type of centrifuge used in uranium-enrichment sites [1116, 1137]. In 2013, a security breach at Target, a large chain of stores in the United States, compromised 40 million credit cards [1497, 1745, 2237]. Also in 2013, Yahoo reported that an attack compromised more than 1 billion accounts [779]. In 2017, attackers spread ransomware that crippled computers throughout the world, including computers used in hospitals and telecommunications companies [1881]. Equifax estimated that attackers also compromised the personal data of over 100,000,000 people [176].

These attacks exploit vulnerabilities that have their roots in vulnerabilities of the 1980s, 1970s, and earlier. They seem more complex because systems have become more complex, and thus the vulnerabilities are more obscure and require more complex attacks to exploit. But the principles underlying the attacks, the vulnerabilities, and the failures of the systems have not changed—only the arena in which they are applied has.

Consistent with this philosophy, the second edition continues to focus on the principles underlying the field of computer and information security. Many newer examples show how these principles are applied, or not applied, today; but the principles themselves are as important today as they were in 2002, and earlier. Some have been updated to reflect a deeper understanding of people and systems. Others have been applied in new and interesting ways. But they still ring true.

That said, the landscape of security has evolved greatly in the years since this book was first published. The explosive growth of the World Wide Web, and the consequent explosion in its use, has made security a problem at the forefront of our society. No longer can vulnerabilities, both human and technological, be relegated to the background of our daily lives. It is one of the elements at the forefront, playing a role in everyone's life as one browses the web, uses a camera to take and send pictures, and turns on an oven remotely. We grant access to our personal lives through social media such as Facebook, Twitter, and Instagram, and to our homes through the Internet of Things and our connections to the Internet. To ignore security issues, or consider them simply ancillary details that "someone will fix somehow" or threats unlikely to be realized personally is dangerous at best, and potentially disastrous at worst.

Ultimately, little has changed. The computing ecosystem of our day is badly flawed. Among the manifestations of these technological flaws are that security problems continue to exist, and continue to grow in magnitude of effect. An interesting question to ponder is what might move the paradigm of security away from the cycle of "patch and catch" and "let the buyer beware" to a stable and safer ecosystem.

But we must continue to improve our understanding of, and implementation of, security. Security nihilism—simply giving up and asserting that we cannot make things secure, so why try—means we accept these invasions of our privacy, our society, and our world. Like everything else, security is imperfect, and always will be—meaning we can improve the state of the art. This book is directed towards that goal.

Updated Roadmap

The dependencies of the chapters are the same as in the first edition (see p. xl), with two new chapters added.

Chapter 7, which includes a discussion of denial of service attack models, contains material useful for Chapters 23, 24, 27, and 28. Similarly, Chapter 27 draws on material from the chapters in Part III as well as Chapters 23, 25, 26, and all of Part VIII.

In addition to the suggestions in the preface to the first edition on p. xli about topics for undergraduate classes, the material in Chapter 27 will introduce undergraduates to how attacks occur, how they can be analyzed, and what their effects are. Coupled with current examples drawn from the news, this chapter should prove fascinating to undergraduates.

As for graduate classes, the new material in Chapter 7 will provide students with some background on resilience, a topic increasing in importance. Otherwise, the recommendations are the same as for the first edition (see p. xlii).

Changes to the First Edition

The second edition has extensively revised many examples to apply the concepts to technologies, methodologies, and ideas that have emerged since the first edition was published. Here, the focus is on new material in the chapters; changes to examples are mentioned only when necessary to describe that material. In addition to what is mentioned here, much of the text has been updated.

Chapter 1, "An Overview of Computer Security": This chapter is largely unchanged.

Chapter 2, "Access Control Matrix": Section 2.2.2, "Access Controlled by History" has been changed to use the problem of preventing downloaded programs from accessing the system in unauthorized ways, instead of updating a database. Section 2.4.3, "Principle of Attenuation of Privilege," has been expanded slightly, and exercises added to point out differing forms of the principle.

Chapter 3, "Foundational Results": Definition 3–1 has been updated to make clear that "leaking" refers to a right being added to an element of the access control matrix that did not contain it initially, and an exercise has been added to demonstrate the difference between this definition and the one in the first edition. Section 3.6 discusses comparing security properties of models.

Chapter 4, "Security Policies": Section 4.5.1, "High-Level Policy Languages," now uses Ponder rather than a Java policy constraint language. Section 4.6, "Example: Academic Computer Security Policy," has been updated to reflect changes in the university policy.

Chapter 5, "Confidentiality Policies": Section 5.3.1 discusses principles for declassifying information.

Chapter 6, "Integrity Policies": Section 6.5 presents trust models.

Chapter 7, "Availability Policies": This chapter is new.

Chapter 8, "Hybrid Policies": Section 8.1.3 modifies one of the assumptions of the Chinese Wall model that is unrealistic. Section 8.3.1 expands the discussion of ORCON to include DRM. Section 8.4 adds a discussion of several types of RBAC models.

Chapter 9, "Noninterference and Policy Composition": This chapter adds Section 9.6, which presents side channels in the context of deducibility.

Chapter 10, "Basic Cryptography": This chapter has been extensively revised. The discussion of the DES (Section 10.2.3) has been tightened and the algorithm moved to Appendix F. Discussions of the AES (Section 10.2.5) and elliptic curve cryptography (Section 10.3.3) have been added, and the section on digital signatures moved from Chapter 11 to Section 10.5. Also, the number of digits in the integers used in examples for public key cryptography has been increased from 2 to at least 4, and in many cases more.

Chapter 11, "Key Management": Section 11.4.3 discusses public key infrastructures. Section 11.5.1.4, "Other Approaches," now includes a brief discussion of identity-based encryption.

Chapter 12, "Cipher Techniques": Section 12.1, "Problems," now includes a discussion of type flaw attacks. Section 12.3 discusses authenticated encryption with associated data, and presents the CCM and GCM modes of block ciphers. A new section, Section 12.5.2, discusses the Signal Protocol. Section 12.5.3, "Security at the Transport Layer: TLS and SSL," has been expanded and focuses on TLS rather than SSL. It also discusses cryptographic weaknesses in SSL, such as the POODLE attack, that have led to the use of SSL being strongly discouraged.

Chapter 13, "Authentication": A discussion of graphical passwords has been added as Section 13.3.4. Section 13.4.3 looks at quantifying password strength in terms of entropy. The discussion of biometrics in Section 13.7 has been expanded to reflect their increasing use.

Chapter 14, "Design Principles": The principle of least authority follows the principle of least privilege in Section 14.2.1, and the principle of least astonishment now supersedes the principle of psychological acceptability in Section 14.2.8.

Chapter 15, "Representing Identity": Section 15.5, "Naming and Certificates," now includes a discussion of registration authorities (RAs). Section 15.6.1.3 adds a discussion of the DNS security extensions (DNSSEC). Section 15.7.2 discusses onion routing and Tor in the context of anonymity.

Chapter 16, "Access Control Mechanisms": Section 16.2.6 discusses sets of privileges in Linux and other UNIX-like systems.

Chapter 17, "Information Flow": In contrast to the confidentiality-based context of information flow in the main part of this chapter, Section 17.5 presents information flow in an integrity context. In Section 17.6, the SPI and SNSMG examples of the first edition have been replaced by Android cell phones (Section 17.6.1) and firewalls (Section 17.6.2).

Chapter 18, "Confinement Problem": Section 18.2 has been expanded to include library operating systems (Section 18.2.1.2) and program modification techniques (Section 18.2.2).

Chapter 19, "Introduction to Assurance": Section 19.2.3, which covers agile software development, has been added.

Chapter 20, "Building Systems with Assurance": The example decomposition of Windows 2000 into components has been updated to use Windows 10.

Chapter 21, "Formal Methods": A new section, Section 21.5, discusses functional programming languages, and another new section, 21.6, discusses formally verified products.

Chapter 22, "Evaluating Systems": Sections 22.7, on FIPS 140, and 22.8, on the Common Criteria, have been extensively updated.

Chapter 23, "Malware": Section 23.5 presents botnets, and Sections 23.6.3, 23.6.4, 23.6.5, and 23.6.6 discuss adware and spyware, ransomware, and phishing. While not malware, phishing is a common vector for getting malware onto a system and so it is discussed here.

Chapter 24, "Vulnerability Analysis": Section 24.2.5 reviews several penetration testing frameworks used commercially and based on the Flaw Hypothesis Methodology. Section 24.5 presents the widely used CVE and CWE standards.

Chapter 25, "Auditing": Section 25.3.3, which discusses sanitization, has been expanded.

Chapter 26, "Intrusion Detection": Section 26.3.1 has been expanded to include several widely used machine learning techniques for anomaly detection. Incident response groups are discussed in Section 27.3.

Chapter 27, "Attacks and Responses": This chapter is new.

Chapter 28, "Network Security": The discussion of what firewalls are has been moved to Section 17.6.2, but the discussion of how the Drib configures and uses them remains in this chapter. The Drib added wireless networks, which are discussed in Section 28.3.3.1. Its analysis of using the cloud is in Section 28.3.3.2. The rest of the chapter has been updated to refer to the new material in previous chapters.

Chapter 29, "System Security": This chapter has been updated to refer to the new material in previous chapters.

Chapter 30, "User Security": Section 30.2.2 describes the two-factor authentication procedure used by the Drib. The rest of the chapter has been updated to refer to the new material in previous chapters.

Chapter 31, "Program Security": This chapter has been updated to refer to the new material in previous chapters.

Two new appendices have been added. Appendix F presents the DES and AES algorithms, and Appendix H collects the rules in Chapter 31 for easy reference. In addition, Appendix D examines some hardware enhancements to aid virtualization, and Appendix G contains the full academic security policy discussed in Section 4.6.

Preface to the First Edition[1]

On September 11, 2001, terrorists seized control of four airplanes. Three were flown into buildings, and a fourth crashed, with catastrophic loss of life. In the aftermath, the security and reliability of many aspects of society drew renewed scrutiny. One of these aspects was the widespread use of computers and their interconnecting networks. The issue is not new. In 1988, approximately 5,000 computers throughout the Internet were rendered unusable within 4 hours by a program called a worm [842].[2] While the spread, and the effects, of this program alarmed computer scientists, most people were not worried because the worm did not affect their lives or their ability to do their jobs. In 1993, more users of computer systems were alerted to such dangers when a set of programs called sniffers were placed on many computers run by network service providers and recorded login names and passwords [670].

After an attack on Tsutomu Shimomura's computer system, and the fascinating way Shimomura followed the attacker's trail, which led to his arrest [1736], the public's interest and apprehension were finally aroused. Computers were now vulnerable. Their once reassuring protections were now viewed as flimsy.

Several films explored these concerns. Movies such as *War Games* and *Hackers* provided images of people who can, at will, wander throughout computers and networks, maliciously or frivolously corrupting or destroying information it may have taken millions of dollars to amass. (Reality intruded on *Hackers* when the World Wide Web page set up by MGM/United Artists was quickly altered to present an irreverent commentary on the movie and to suggest that viewers see *The Net* instead. Paramount Pictures denied doing this [869].) Another film, *Sneakers*, presented a picture of those who test the security of computer (and other) systems for their owners and for the government.

Goals

This book has three goals. The first is to show the importance of theory to practice and of practice to theory. All too often, practitioners regard theory as irrelevant and theoreticians think of practice as trivial. In reality, theory and practice are symbiotic. For example, the theory of covert channels, in which the goal is to limit the ability of processes to communicate through shared resources, provides a mechanism for evaluating the effectiveness of mechanisms that confine processes, such as sandboxes and firewalls. Similarly, business practices in the commercial world led to the development of several security policy models such as the Clark-Wilson model and the Chinese Wall model. These models in turn help the designers of security policies better understand and evaluate the mechanisms and procedures needed to secure their sites.

---

[1] Chapter numbers have been updated to correspond to the chapters in the second edition.

[2] Section 23.4 discusses computer worms.

The second goal is to emphasize that computer security and cryptography are different. Although cryptography is an essential component of computer security, it is by no means the only component. Cryptography provides a mechanism for performing specific functions, such as preventing unauthorized people from reading and altering messages on a network. However, unless developers understand the context in which they are using cryptography, and unless the assumptions underlying the protocol and the cryptographic mechanisms apply to the context, the cryptography may not add to the security of the system. The canonical example is the use of cryptography to secure communications between two low-security systems. If only trusted users can access the two systems, cryptography protects messages in transit. But if untrusted users can access either system (through authorized accounts or, more likely, by breaking in), the cryptography is not sufficient to protect the messages. The attackers can read the messages at either endpoint.

The third goal is to demonstrate that computer security is not just a science but also an art. It is an art because no system can be considered secure without an examination of how it is to be used. The definition of a "secure computer" necessitates a statement of requirements and an expression of those requirements in the form of authorized actions and authorized users. (A computer engaged in work at a university may be considered "secure" for the purposes of the work done at the university. When moved to a military installation, that same system may not provide sufficient control to be deemed "secure" for the purposes of the work done at that installation.) How will people, as well as other computers, interact with the computer system? How clear and restrictive an interface can a designer create without rendering the system unusable while trying to prevent unauthorized use or access to the data or resources on the system?

Just as an artist paints his view of the world onto canvas, so does a designer of security features articulate his view of the world of human/machine interaction in the security policy and mechanisms of the system. Two designers may use entirely different designs to achieve the same creation, just as two artists may use different subjects to achieve the same concept.

Computer security is also a science. Its theory is based on mathematical constructions, analyses, and proofs. Its systems are built in accordance with the accepted practices of engineering. It uses inductive and deductive reasoning to examine the security of systems from key axioms and to discover underlying principles. These scientific principles can then be applied to untraditional situations and new theories, policies, and mechanisms.

Philosophy

Key to understanding the problems that exist in computer security is a recognition that the problems are not new. They are old problems, dating from the beginning of computer security (and, in fact, arising from parallel problems in the noncomputer world). But the locus has changed as the field of computing has changed. Before the mid-1980s, mainframe and mid-level computers dominated the market, and computer security problems and solutions were phrased in terms of securing files or processes on a single system. With the rise of networking and the Internet, the arena has changed. Workstations and servers, and the networking infrastructure that connects them, now dominate the market. Computer security problems and solutions now focus on a networked environment. However, if the workstations and servers, and the supporting network infrastructure, are viewed as a single system, the models, theories, and problem statements developed for systems before the mid-1980s apply equally well to current systems.

As an example, consider the issue of assurance. In the early period, assurance arose in several ways: formal methods and proofs of correctness, validation of policy to requirements, and acquisition of data and programs from trusted sources, to name a few. Those providing assurance analyzed a single system, the code on it, and the sources (vendors and users) from which the code could be acquired to ensure that either the sources could be trusted or the programs could be confined adequately to do minimal damage. In the later period, the same basic principles and techniques apply, except that the scope of some has been greatly expanded (from a single system and a small set of vendors to the world-wide Internet). The work on proof-carrying code, an exciting development in which the proof that a downloadable program module satisfies a stated policy is incorporated into the program itself, is an example of this expansion.[3] It extends the notion of a proof of consistency with a stated policy. It advances the technology of the earlier period into the later period. But in order to understand it properly, one must understand the ideas underlying the concept of proof-carrying code, and these ideas lie in the earlier period.

As another example, consider Saltzer and Schroeder's principles of secure design.[4] Enunciated in 1975, they promote simplicity, confinement, and understanding. When security mechanisms grow too complex, attackers can evade or bypass them. Many programmers and vendors are learning this when attackers break into their systems and servers. The argument that the principles are old, and somehow outdated, rings hollow when the result of their violation is a non-secure system.

The work from the earlier period is sometimes cast in terms of systems that no longer exist and that differ in many ways from modern systems. This does not vitiate the ideas and concepts, which also underlie the work done today. Once these ideas and concepts are properly understood, applying them in a multiplicity of environments becomes possible. Furthermore, the current mechanisms and technologies will become obsolete and of historical interest themselves as new forms of computing arise, but the underlying principles will live on, to underlie the next generation—indeed the next era—of computing.

The philosophy of this book is that certain key concepts underlie all of computer security, and that the study of all parts of computer security enriches the understanding of all parts. Moreover, critical to an understanding of the applications of security-related technologies and methodologies is an understanding of the theory underlying those applications. Advances in the theory of computer protection have illuminated the foundations of security systems. Issues of abstract modeling, and modeling to meet specific environments, lead to systems designed to achieve a specific and rewarding goal. Theorems about composability of policies[5] and the undecidability of the general security question[6] have indicated the limits of what can be done. Much work and effort are continuing to extend the borders of those limits.

---

[3] Section 23.9.5.1 discusses proof-carrying code.

[4] Chapter 14 discusses these principles.

Application of these results has improved the quality of the security of the systems being protected. However, the issue is how compatibly the assumptions of the model (and theory) conform to the environment to which the theory is applied. Although our knowledge of how to apply these abstractions is continually increasing, we still have difficulty correctly transposing the relevant information from a realistic setting to one in which analyses can then proceed. Such abstraction often eliminates vital information. The omitted data may pertain to security in non-obvious ways. Without this information, the analysis is flawed.

The practitioner needs to know both the theoretical and practical aspects of the art and science of computer security. The theory demonstrates what is possible. The practical makes known what is feasible. The theoretician needs to understand the constraints under which these theories are used, how their results are translated into practical tools and methods, and how realistic are the assumptions underlying the theories. *Computer Security: Art and Science* tries to meet these needs.

Unfortunately, no single work can cover all aspects of computer security, so this book focuses on those parts that are, in the author's opinion, most fundamental and most pervasive. The mechanisms exemplify the applications of these principles.

Organization

The organization of this book reflects its philosophy. It begins with mathematical fundamentals and principles that provide boundaries within which security can be modeled and analyzed effectively. The mathematics provides a framework for expressing and analyzing the requirements of the security of a system. These policies constrain what is allowed and what is not allowed. Mechanisms provide the ability to implement these policies. The degree to which the mechanisms correctly implement the policies, and indeed the degree to which the policies themselves meet the requirements of the organizations using the system, are questions of assurance. Exploiting failures in policy, in implementation, and in assurance comes next, as well as mechanisms for providing information on the attack. The book concludes with the applications of both theory and policy focused on realistic situations. This natural progression emphasizes the development and application of the principles existent in computer security.

---

[5] See Chapter 9, "Noninterference and Policy Composition."

[6] See Section 3.2, "Basic Results."

Part I, "Introduction," describes what computer security is all about and explores the problems and challenges to be faced. It sets the context for the remainder of the book.

Part II, "Foundations," deals with basic questions such as how "security" can be clearly and functionally defined, whether or not it is realistic, and whether or not it is decidable. If it is decidable, under what conditions is it decidable, and if not, how must the definition be bounded in order to make it decidable?

Part III, "Policy," probes the relationship between policy and security. The definition of "security" depends on policy. In Part III we examine several types of policies, including the ever-present fundamental questions of trust, analysis of policies, and the use of policies to constrain operations and transitions.

Part IV, "Implementation I: Cryptography," discusses cryptography and its role in security. It focuses on applications and discusses issues such as key management and escrow, key distribution, and how cryptosystems are used in networks. A quick study of authentication completes Part III.

Part V, "Implementation II: Systems," considers how to implement the requirements imposed by policies using system-oriented techniques. Certain design principles are fundamental to effective security mechanisms. Policies define who can act and how they can act, and so identity is a critical aspect of implementation. Mechanisms implementing access control and flow control enforce various aspects of policies.

Part VI, "Assurance," presents methodologies and technologies for ascertaining how well a system, or a product, meets its goals. After setting the background, to explain exactly what "assurance" is, the art of building systems to meet varying levels of assurance is discussed. Formal verification methods play a role. Part VI shows how the progression of standards has enhanced our understanding of assurance techniques.

Part VII, "Special Topics," discusses some miscellaneous aspects of computer security. Malicious logic thwarts many mechanisms. Despite our best efforts at high assurance, systems today are replete with vulnerabilities. Why? How can a system be analyzed to detect vulnerabilities? What models might help us improve the state of the art? Given these security holes, how can we detect attackers who exploit them? A discussion of auditing flows naturally into a discussion of intrusion detection—a detection method for such attacks.

Part VIII, "Practicum," presents examples of how to apply the principles discussed throughout the book. It begins with networks and proceeds to systems, users, and programs. Each chapter states a desired policy and shows how to translate that policy into a set of mechanisms and procedures that support the policy. Part VIII tries to demonstrate that the material covered elsewhere can be, and should be, used in practice.

Each chapter in this book ends with a summary, descriptions of some research issues, and some suggestions for further reading. The summary highlights the important ideas in the chapter. The research issues are current "hot topics" or are topics that may prove to be fertile ground for advancing the state of the art and science of computer security. Interested readers who wish to pursue the topics in any chapter in more depth can go to some of the suggested readings. They expand on the material in the chapter or present other interesting avenues.

Roadmap

This book is both a reference book and a textbook. Its audience is undergraduate and graduate students as well as practitioners. This section offers some suggestions on approaching the book.

Dependencies

Chapter 1 is fundamental to the rest of the book and should be read first. After that, however, the reader need not follow the chapters in order. Some of the dependencies among chapters are as follows.

Chapter 3 depends on Chapter 2 and requires a fair degree of mathematical maturity. Chapter 2, on the other hand, does not. The material in Chapter 3 is for the most part not used elsewhere (although the existence of the first section's key result, the undecidability theorem, is mentioned repeatedly). It can be safely skipped if the interests of the reader lie elsewhere.

The chapters in Part III build on one another. The formalisms in Chapter 5 are called on in Chapters 20 and 21, but nowhere else. Unless the reader intends to delve into the sections on theorem proving and formal mappings, the formalisms may be skipped. The material in Chapter 9 requires a degree of mathematical maturity, and this material is used sparingly elsewhere. Like Chapter 3, Chapter 9 can be skipped by the reader whose interests lie elsewhere.

Chapters 10, 11, and 12 also build on one another in order. A reader who has encountered basic cryptography will have an easier time with the material than one who has not, but the chapters do not demand the level of mathematical experience that Chapters 3 and 9 require. Chapter 13 does not require material from Chapter 11 or Chapter 12, but it does require material from Chapter 10.

Chapter 14 is required for all of Part V. A reader who has studied operating systems at the undergraduate level will have no trouble with Chapter 16. Chapter 15 uses the material in Chapters 10 and 11; Chapter 17 builds on material in Chapters 5, 14, and 16; and Chapter 18 uses material in Chapters 4, 14, and 17.

Chapter 19 relies on information in Chapter 4. Chapter 20 builds on Chapters 5, 14, 16, and 19. Chapter 21 presents highly mathematical concepts and uses material from Chapters 19 and 20. Chapter 22 is based on material in Chapters 5, 19, and 20; it does not require Chapter 21. For all of Part VI, a knowledge of software engineering is very helpful.

Chapter 23 draws on ideas and information in Chapters 5, 6, 10, 14, 16, and 18 (and for Section 23.8, the reader should read Section 3.1). Chapter 24 is self-contained, although it implicitly uses many ideas from assurance. It also assumes a good working knowledge of compilers, operating systems, and in some cases networks. Many of the flaws are drawn from versions of the UNIX operating system, or from Windows systems, and so a working knowledge of either or both systems will make some of the material easier to understand. Chapter 25 uses information from Chapter 4, and Chapter 26 uses material from Chapter 25.

The practicum chapters are self-contained and do not require any material beyond Chapter 1. However, they point out relevant material in other sections that augments the information and (we hope) the reader's understanding of that information.

Background

The material in this book is at the advanced undergraduate level. Throughout, we assume that the reader is familiar with the basics of compilers and computer architecture (such as the use of the program stack) and operating systems. The reader should also be comfortable with modular arithmetic (for the material on cryptography). Some material, such as that on formal methods (Chapter 21) and the mathematical theory of computer security (Chapter 3 and the formal presentation of policy models), requires considerable mathematical maturity. Other specific recommended background is presented in the preceding section. Part IX, the appendices, contains material that will be helpful to readers with backgrounds that lack some of the recommended material.

Examples are drawn from many systems. Many come from the UNIX operating system or variations of it (such as Linux). Others come from the Windows family of systems. Familiarity with these systems will help the reader understand many examples easily and quickly.

Undergraduate Level

An undergraduate class typically focuses on applications of theory and how students can use the material. The specific arrangement and selection of material depends on the focus of the class, but all classes should cover some basic material—notably that in Chapters 1, 10, and 14, as well as the notion of an access control matrix, which is discussed in Sections 2.1 and 2.2.

Presentation of real problems and solutions often engages undergraduate students more effectively than presentation of abstractions. The special topics and the practicum provide a wealth of practical problems and ways to deal with them. This leads naturally to the deeper issues of policy, cryptography, non-cryptographic mechanisms, and assurance. The following are sections appropriate for non-mathematical undergraduate courses in these topics.

- *Policy*: Sections 4.1 through 4.4 describe the notion of policy. The instructor should select one or two examples from Sections 5.1, 5.2.1, 6.2, 6.4, 8.1.1, and 8.2, which describe several policy models informally. Section 8.4 discusses role-based access control.
- *Cryptography*: Key distribution is discussed in Sections 11.1 and 11.2, and a common form of public key infrastructures (called PKIs) is discussed in Section 11.4.2. Section 12.1 points out common errors

in using cryptography. Section 12.4 shows how cryptography is used in networks, and the instructor should use one of the protocols in Section 12.5 as an example. Chapter 13 offers a look at various forms of authentication, including non-cryptographic methods.

- *Non-cryptographic mechanisms*: Identity is the basis for many access control mechanisms. Sections 15.1 through 15.4 discuss identity on a system, and Section 15.6 discusses identity and anonymity on the Web. Sections 16.1 and 16.2 explore two mechanisms for controlling access to files, and Section 16.4 discusses the ring-based mechanism underlying the notion of multiple levels of privilege. If desired, the instructor can cover sandboxes by using Sections 18.1 and 18.2, but because Section 18.2 uses material from Section 4.5, the instructor will need to go over those sections as well.
- *Assurance*: Chapter 19 provides a basic introduction to the often overlooked topic of assurance.

Graduate Level

A typical introductory graduate class can focus more deeply on the subject than can an undergraduate class. Like an undergraduate class, a graduate class should cover Chapters 1, 10, and 14. Also important are the undecidability results in Sections 3.1 and 3.2, which require that Chapter 2 be covered. Beyond that, the instructor can choose from a variety of topics and present them to whatever depth is appropriate. The following are sections suitable for graduate study.

- *Policy models*: Part III covers many common policy models both informally and formally. The formal description is much easier to understand once the informal description is understood, so in all cases both should be covered. The controversy in Section 5.4 is particularly illuminating to students who have not considered the role of policy and the nature of a policy. Chapter 9 is a highly formal discussion of the foundations of policy and is appropriate for students with experience in formal mathematics. Students without such a background will find it quite difficult.
- *Cryptography*: Part IV focuses on the applications of cryptography, not on cryptography's mathematical underpinnings.[7] It discusses areas of interest critical to the use of cryptography, such as key management and some basic cryptographic protocols used in networking.
- *Non-cryptographic mechanisms*: Issues of identity and certification are complex and generally poorly understood. Section 15.5 covers these problems. Combining this with the discussion of identity on the Web

---

[7] The interested reader will find a number of books covering aspects of this subject [440, 787, 788, 914, 1092, 1093, 1318, 1826].

(Section 15.6) raises issues of trust and naming. Chapters 17 and 18 explore issues of information flow and confining that flow.

- *Assurance*: Traditionally, assurance is taught as formal methods, and Chapter 21 serves this purpose. In practice, however, assurance is more often accomplished by using structured processes and techniques and informal but rigorous arguments of justification, mappings, and analysis. Chapter 20 emphasizes these topics. Chapter 22 discusses evaluation standards and relies heavily on the material in Chapters 19 and 20 and some of the ideas in Chapter 21.
- *Miscellaneous Topics*: Section 23.8 presents a proof that the generic problem of determining if a generic program is a computer virus is in fact undecidable. The theory of penetration studies in Section 24.2, and the more formal approach in Section 24.6, illuminate the analysis of systems for vulnerabilities. If the instructor chooses to cover intrusion detection (Chapter 26) in depth, it should be understood that this discussion draws heavily on the material on auditing (Chapter 25).
- *Practicum*: The practicum (Part VIII) ties the material in the earlier part of the book to real-world examples and emphasizes the applications of the theory and methodologies discussed earlier.

Practitioners

Practitioners in the field of computer security will find much to interest them. The table of contents and the index will help them locate specific topics. A more general approach is to start with Chapter 1 and then proceed to Part VIII, the practicum. Each chapter has references to other sections of the text that explain the underpinnings of the material. This will lead the reader to a deeper understanding of the reasons for the policies, settings, configurations, and advice in the practicum. This approach also allows readers to focus on those topics that are of most interest to them.

This page intentionally left blank

Acknowledgments

It is not possible to separate those who contributed to the second edition from those who contributed to the first edition, because everything done for the first edition, especially after the first printing, has contributed to the second. So these acknowledgments apply to both editions. That said . . .

Special Acknowledgments

Elisabeth Sullivan and Michelle Ruppel contributed the assurance part of this book.

For the first edition, Liz wrote several drafts, all of which reflect her extensive knowledge and experience in that aspect of computer security. I am particularly grateful to her for contributing her real-world knowledge of how assurance is managed. Too often, books recount the mathematics of assurance without recognizing that other aspects are equally important and more widely used. These other aspects shine through in the assurance section, thanks to Liz. As if that were not enough, she made several suggestions that improved the policy part of this book. I will always be grateful for her contribution, her humor, and especially her friendship.

For the second edition, Michelle stepped in to update that part based on her extensive experience and real-world knowledge as a practitioner. She was careful to maintain the tone and style of Liz's writing, and her contributions strengthened the assurance part. I am grateful to her for agreeing to step in, for the exceptional effort she put forth, and the high quality that resulted.

In summary, I am very grateful for their contributions.

Acknowledgments

Many people offered comments, suggestions, and ideas for the second edition. Thanks to Marvin Schaefer, Sean Peisert, Prof. Christian Probst, Carrie Gates, and Richard Ford for their reviews of the various chapters. I appreciate Prof. Ken

Rosen and Prof. Alfred Menezes for their help with Chapter 10, Steven Templeton and Kara Nance for their suggestions on Chapter 27, Karl Levitt for his comments on Chapter 26, and Richard Ford for his many suggestions on Chapter 23. Their advice and suggestions were invaluable in preparing this edition. Of course, any errors in the text are my responsibility, and usually occurred because I did not always follow their advice.

Thanks also to Pasquale Noce, who sent me a thorough analysis of many of the theorems, proving them constructively as opposed to how they were done in the book. He made many other helpful comments and caught some errors.

The students in Peter Reiher's COM SCI 236-80, Computer Security, class at UCLA in the Spring Quarter 2018, and the students in my ECS 153, Computer Security, classes over the past few years at UC Davis used parts of this edition in various stages of preparation. I thank them for their feedback, which also improved the book.

Many others contributed to this book in various ways. Special thanks to Steven Alexander, Amin Alipour, Jim Alves-Foss, Bill Arbaugh, Andrew Arcilla, Alex Aris, Rebecca Bace, Belinda Bashore, Vladimir Berman, Rafael Bhatti, Ziad El Bizri, David Bover, Logan Browne, Terry Brugger, Gordon Burns, Serdar Cabuk, Raymond Centeno, Yang Chen, Yi Chen, HakSoo Choi, Lisa Clark, Michael Clifford, Christopher Clifton, Dan Coming, Kay Connelly, Crispin Cowan, Shayne Czyzewski, Tom Daniels, Dimitri DeFigueiredo, Joseph-Patrick Dib, Till Dörges, Felix Fan, Robert Fourney, Guillermo Francia III, Jeremy Frank, Conny Francke, Martin Gagne, Nina Gholami, Ron Gove, James Hinde, James Hook, Xuxian Jiang, Jesper Johansson, Mark Jones, Calvin Ko, Mark-Neil Ledesma, Ken Levine, Karl Levitt, Luc Longpre, Yunhua Lu, Gary McGraw, Alexander Meau, Nasir Memon, Katherine Moore, Mark Morrissey, Ather Nawaz, Iulian Neamtiu, Dan Nerenburg, Kimberly Nico, Stephen Northcutt, Rafael Obelheiro, Josko Orsulic, Holly Pang, Sean Peisert, Ryan Poling, Sung Park, Ashwini Raina, Jorge Ramos, Brennen Reynolds, Peter Rozental, Christoph Schuba, night SH, David Shambroom, Jonathan Shapiro, Clay Shields, Sriram Srinivasan, Mahesh V. Tripunitara, Vinay Vittal, Tom Walcott, James Walden, Dan Watson, Guido Wedig, Chris Wee, Han Weili, Patrick Wheeler, Paul Williams, Bonnie Xu, Charles Yang, Xiaoduan Ye, Xiaohui Ye, Lara Whelan, John Zachary, Linfeng Zhang, Aleksandr Zingorenko, and to everyone in my and others' computer security classes, who (knowingly or unknowingly) helped me develop and test this material.

The Pearson folks, in particular my editors Laura Lewin and Malobika Chakraborty, and Sheri Replin, were incredibly helpful and patient. Their patience and enthusiasm ensured this second edition was completed, although a bit later than expected. The production people, especially Julie Nahil, Ramya Gangadharan, and Charles Roumeliotis, moved the book smoothly into print, and I thank them for making it as painless as possible. I owe them many thanks. Similarly, for the first edition, the Addison-Wesley folks, Kathleen Billus, Susannah Buzard, Bernie Gaffney, Amy Fleischer, Helen Goldstein, Tom Stone, Asdis Thorsteinsson, and most especially my editor, Peter Gordon, were incredibly patient and

helpful, despite fears that this book would never materialize. The fact that it did so is in great measure attributable to their hard work and encouragement. I also thank the production people at Argosy, especially Beatriz Valdés and Craig Kirkpatrick, for their wonderful work.

Dorothy Denning, my advisor in graduate school, guided me through the maze of computer security when I was just beginning. Peter Denning, Barry Leiner, Karl Levitt, Peter Neumann, Marvin Schaefer, Larry Snyder, and several others influenced my approach to the subject. I hope this work reflects in some small way what they gave to me and passes a modicum of it along to my readers.

I also thank my parents, Leonard Bishop and Linda Allen. My father, a writer, gave me some useful tips on writing, which I tried to follow. My mother, a literary agent, helped me understand the process of getting the book published, and supported me throughout.

Finally, I would like to thank my family for their support throughout the writing. My wife Holly, our children Heidi, Steven, David, and Caroline, and grandchildren Skyler and Sage were very patient and understanding and made sure I had time to work on the book. They also provided delightful distractions. To them all, my love and gratitude.

This page intentionally left blank

About the Author

Matt Bishop is a professor in the Department of Computer Science at the University of California at Davis. He received his Ph.D. in computer science from Purdue University, where he specialized in computer security, in 1984. He was a systems programmer at Megatest Corporation, a research scientist at the Research Institute of Advanced Computer Science and was on the faculty at Dartmouth College.

His main research area is the analysis of vulnerabilities in computer systems, including modeling them, building tools to detect vulnerabilities, and ameliorating or eliminating them. This includes detecting and handling all types of malicious logic. He works in the areas of network security, the study of denial of service attacks and defenses, policy modeling, software assurance testing, resilience, and formal modeling of access control. He has worked extensively in electronic voting, was one of the members of the RABA study for Maryland, and was one of the two principle investigators of the California Top-to-Bottom Review, which performed a technical review of all electronic voting systems certified in the State of California.

He is active in information assurance education. He was co-chair of the Joint Task Force that developed the *Cybersecurity Curricula 2017: Curriculum Guidelines for Post-Secondary Degree Programs in Cybersecurity*, released in December 2017. He teaches introductory programming, software engineering, operating systems, and (of course) computer security.

This page intentionally left blank

Chapter 31

Program Security

CLOWN: What is he that builds stronger than either
the mason, the shipwright, or the carpenter?
OTHER CLOWN: The gallows-maker; for that frame outlives
a thousand tenants.
— *Hamlet*, V, i, 42–45.

The software on systems implements many mechanisms that support security. Some of these mechanisms reside in the operating system, whereas others reside in application and system programs. This chapter discusses the design and implementation of a program to grant users increased privileges. It also presents common programming errors that create security problems, and offers suggestions for avoiding those problems. Finally, testing and distribution are discussed.

This chapter shows the development of the program from requirements to implementation, testing, and distribution.

31.1 Problem

The purpose of this chapter is to provide a glimpse of techniques that provide better than ordinary assurance that a program's design and implementation satisfy its requirements. This chapter is not a manual on applying high-assurance techniques. In terms of the techniques discussed in Part VI, "Assurance," this chapter describes low-assurance techniques.

However, given the current state of programming and software development, these low-assurance techniques enable programmers to produce significantly better, more robust, and more usable code than they could produce without these techniques. So, using a methodology similar to the one outlined in this chapter will reduce vulnerabilities and improve both the quality and the security of code.

A specific problem will illustrate the methods in this chapter. On the Drib's development network infrastructure systems, numerous system administrators must assume certain roles, such as *bin* (the installers of software), *mail* (the manager of electronic mail), and *root* (the system administrator). Each of these roles is implemented as a separate account, called a *role account*. Unfortunately, this raises the problem of password management. To avoid this problem, as well as to control when access is allowed, the Drib will implement a program that verifies a user's identity, determines if the requested change of account is allowed, and, if so, places the user in the desired role.

31.2 Requirements and Policy

The problem of sharing a password arises when a system implements administrative roles as separate user accounts.

EXAMPLE: Linux systems implement the administrator role as the account *root* (and several other accounts that have more limited functionality).[1] All individuals who share access to the account know the account's password. If the password is changed, all must be notified. All these individuals must remember to notify the other individuals should they change the password.

An alternative to using passwords is to constrain access on the basis of identity and other attributes. With this scheme, a user would execute a special program that would check the user's identity and any ancillary conditions. If all these conditions were satisfied, the user would be given access to the role account.

31.2.1 Requirements

The first requirement comes directly from the description of the alternative scheme above. The system administrators choose to constrain access through known paths (locations) and at times of day when the user is expected to access the role account.

> **Requirement 31.1.** Access to a role account is based on user, location, and time of request.

Users often tailor their environments to fit their needs. This is also true of role accounts. For example, a role account may use special programs kept in a subdirectory of the role account's home directory. This new directory must be on the role account's search path, and would typically be set in the startup file executed when the user logged in. A question is whether the user's environment should be discarded and replaced by the role account's environment, or whether the two environments should be merged. The requirement chosen for this program is as follows.

---

[1] See Section 14.2.1, "Principle of Least Privilege," for an explanation of how the existence of the *root* account violates the principle of least privilege.

> **Requirement 31.2.** The settings of the role account's environment shall replace the corresponding settings of the user's environment, but the remainder of the user's environment shall be preserved.

The set of role accounts chosen for access using this scheme is critical. If unrestricted access is given (essentially, a full command interpreter), then any user in the role that maintains the access control information can change that information and acquire unrestricted access to the system. Presumably, if the access control information is kept accessible only to *root*, then the users who can alter the information—all of whom have access to *root*—are trusted. Thus:

> **Requirement 31.3.** Only *root* can alter the access control information for access to a role account.

In most cases, a user assuming a particular role will perform specific actions while in that role. For example, someone who enters the role of *oper* may perform backups but may not use other commands. This restricts the danger of commands interacting with the system to produce undesirable effects (such as security violations) and follows from the principle of least privilege.[2] This form of access is called "restricted access."

> **Requirement 31.4.** The mechanism shall allow both restricted access and unrestricted access to a role account. For unrestricted access, the user shall have access to a standard command interpreter. For restricted access, the user shall be able to execute only a specified set of commands.

Requirement 31.4 implicitly requires that access to the role account be granted to authorized users meeting the conditions in Requirement 31.1. Finally, the role account itself must be protected from unauthorized changes.

> **Requirement 31.5.** Access to the files, directories, and objects owned by any account administered by use of this mechanism shall be restricted to those authorized to use the role account, to users trusted to install system programs, and to *root*.

We next check that these requirements are complete with respect to the threats of concern.

---

[2] See Section 14.2.1, "Principle of Least Privilege."

31.2.2 Threats

The threats against this mechanism fall into distinct classes. We enumerate the classes and discuss the requirements that handle each threat.

31.2.2.1 Group 1: Unauthorized Users Accessing Role Accounts

There are four threats that involve attackers trying to acquire access to role accounts using this mechanism.

> **Threat 31.1.** An unauthorized user may obtain access to a role account as though she were an authorized user.
>
> **Threat 31.2.** An authorized user may use a nonsecure channel to obtain access to a role account, thereby revealing her authentication information to unauthorized individuals.
>
> **Threat 31.3.** An unauthorized user may alter the access control information to grant access to the role account.
>
> **Threat 31.4.** An authorized user may execute a Trojan horse (or other form of malicious logic),[3] giving an unauthorized user access to the role account.

Requirements 31.1 and 31.5 handle Threat 31.1 by restricting the set of users who can access a role account and protecting the access control data. Requirement 31.1 also handles Threat 31.2 by restricting the locations from which the user can request access. For example, if the set of locations contains only those on trusted or confidential networks, a passive wiretapper cannot discover the authorized user's password or hijack a session begun by an authorized user. Similarly, if an authorized user connects from an untrusted system, Requirement 31.1 allows the system administrator to configure the mechanism so that the user's request is rejected.

The access control information that Requirement 31.1 specifies can be changed. Requirement 31.3 acknowledges this but restricts changes to trusted users (defined as those with access to the root account). This answers Threat 31.3.

Threat 31.4 is more complex. This threat arises from an untrusted user, without authorization, planting a Trojan horse at some location at which an authorized user might execute it. If the attacker can write into a directory in the role account's search path, this attack is feasible. Requirement 31.2 states that the role account's search path may be selected from two other search paths: the default search path for the role account, and the user's search path altered to include those components of the role account's search path that are not present. This leads to Requirement 31.5 which states that, regardless of how the search path is derived, the final search path may contain only directories (and may access only programs) that trusted users or the role account itself can manipulate. In this case, the attacker cannot place a Trojan horse where someone using the role account may execute it.

---

[3] See Chapter 23, "Malware."

Finally, if a user is authorized to use the role account but is a novice and may change the search path, Requirement 31.4 allows the administrators to restrict the set of commands that the user may execute in that role.

31.2.2.2 Group 2: Authorized Users Accessing Role Accounts

Because access is allowed here, the threats relate to an authorized user changing access permissions or executing unauthorized commands.

> **Threat 31.5.** An authorized user may obtain access to a role account and perform unauthorized commands.

> **Threat 31.6.** An authorized user may execute a command that performs functions that the user is not authorized to perform.

> **Threat 31.7.** An authorized user may change the restrictions on the user's ability to obtain access to the account.

The difference between Threats 31.5 and 31.6 is subtle but important. In the former, the user deliberately executes commands that violate the site security policy. In the latter, the user executes authorized commands that perform covert, unauthorized actions as well as overt, authorized actions—the classic Trojan horse. Threat 31.6 differs from Threat 31.4 because the action may not give access to authorized users; it may simply damage or destroy the system.

Requirement 31.4 handles Threat 31.5. If the user accessing the role account should execute only a specific set of commands, then the access controls must be configured to restrict the user's access to executing only those commands.

Requirements 31.2 and 31.5 handle Threat 31.6 by preventing the introduction of a Trojan horse, as discussed in the preceding section.

Requirement 31.3 answers Threat 31.7. Because all users who have access to *root* are trusted by definition, the only way for an authorized user to change the restrictions on obtaining access to the role account is to implant a backdoor (which is equivalent to a Trojan horse) or to modify the access control information. But the requirement holds that only trusted users can do that, so the authorized user cannot change the information unless he is trusted—in which case, by definition, the threat is handled.

31.2.2.3 Summary

Because the requirements handle the threats, and because all requirements are used, the set of requirements is both necessary and sufficient. We now proceed with the design.

31.3 Design

To create this program, we build modules that fit together to supply security services that satisfy the requirements. First, we create a general framework to guide the development of each interface. Then we examine each requirement separately, and design a component for each requirement.

31.3.1 Framework

The framework begins with the user interface and then breaks down the internals of the program into modules that implement the various requirements.

31.3.1.1 User Interface

The user can run the program in two ways. The first is to request unrestricted access to the account. The second is to request that a specific program be run from the role account. Any interface must be able to handle both.

The simplest interface is a command line. Other interfaces, such as graphical user interfaces, are possible and may make the program easier to use. However, these GUIs will be built in such a way that they construct and execute a command-line version of the program.

The interface chosen is

```
role role_account [ command ]
```

where *role_account* is the name of the role account and *command* is the (optional) command to execute under that account. If the user wants unrestricted access to the role account, he omits *command*. Otherwise, the user is given restricted access and *command* is executed with the privileges of the role account.

The user need not specify the time of day using the interface, because the program can obtain such information from the system. It can also obtain the location from which the user requests access, although the method used presents potential problems (see Section 31.4.3.1). The individual modules handle the remainder of the issues.

31.3.1.2 High-Level Design

The basic algorithm is as follows.

1. Obtain the role account, command, user, location, and time of day. If the command is omitted, the user is requesting unrestricted access to the role account.

2. Check that the user is allowed to access the role account
    a. at the specified location;
    b. at the specified time; and
    c. for the specified command (or without restriction).

   If the user is not, log the attempt and quit.
3. Obtain the user and group information for the role account. Change the privileges of the process to those of the role account.
4. If the user has requested that a specific command be run, overlay the child process with a command interpreter that spawns the named command.
5. If the user has requested unrestricted access, overlay the child process with a command interpreter.

This algorithm points out an important ambiguity in the requirements. Requirements 31.1 and 31.4 do not indicate whether the ability of the user to execute a command in the given role account requires that the user work from a particular location or access the account at a particular time. This design uses the interpretation that a user's ability to run a command in a role account is conditioned on location and time.

The alternative interpretation, that access only is controlled by location and time, and that commands are restricted by role and user, is equally valid. But sometimes the ability to run commands may require that users work at particular times. For example, an operator may create the daily backups at 1 a.m. The operator is not to do backups at other times because of the load on the system. The interpretation of the design allows this. The alternative interpretation requires the backup program, or some other mechanism, to enforce this restriction. Furthermore, the design interpretation includes the alternative interpretation, because any control expressed in the alternative interpretation can be expressed in the design interpretation.

Requirement 31.4 can now be clarified. The addition is in boldface.

> **Requirement 31.6.** The mechanism shall allow both restricted access and unrestricted access to a role account. For unrestricted access, the user shall have access to a standard command interpreter. For restricted access, the user shall be able to execute only a specified set of commands. **The level of access (restricted or unrestricted) shall depend on the user, the role, the time, and the location.**

Thus, the design phase feeds back into the requirements phase, here clarifying the meaning of the requirements. It is left as an exercise for the reader to verify that the new form of this requirement counters the appropriate threats (see Exercise 2).

31.3.2 Access to Roles and Commands

The user attempting access, the location (host or terminal), the time of day, and the type of access (restricted or unrestricted) control access to the role account. The access checking module returns a value indicating success (meaning that access is allowed) or failure (meaning that access is not allowed). By the principle of fail-safe defaults, an error causes a denial of access.

We consider two aspects of the design of this module. The interface controls how information is passed to the module from its caller, and how the module returns success or failure. The internal structure of the module includes how it handles errors. This leads to a discussion of how the access control data is stored. We consider these issues separately to emphasize that the interface provides an entry point into the module, and that the entry point will remain fixed even if the internal design of the module is completely changed. The internal design and structures are hidden from the caller.

31.3.2.1 Interface

Following the practice of hiding information among modules,[4] we minimize the amount of information to be passed to the access checking module. The module requires the user requesting access, the role to which access is requested, the location, the time, and the command (if any). The return value must indicate success or failure. The question is how this information is to be obtained.

The command (or request for unrestricted access) must come from the caller, because the caller provides the interface for the processing of that command. The command is supplied externally, so the principles of layering require it to pass through the program to the module.

The caller could also pass the other information to the module. This would allow the module to provide an access control result without obtaining the information directly. The advantage is that a different program could use this module to determine whether or not access *had been* or *would be* granted at some past or future point in time, or from some other location. The disadvantage is a lack of portability, because the interface is tied to a particular representation of the data. Also, if the caller of the module is untrusted but the module is trusted, the module might make trusted decisions based on untrusted data, violating a principle of integrity.[5] So we choose to have the module determine all of the data.

This suggests the following interface:

```
boolean accessok(role rname, command cmd);
```

where *rname* is the name of the requested role and *cmd* is the command to be executed (or is empty if unrestricted access is desired). The routine returns **true** if access is to be granted, and **false** otherwise.

---

[4] This is one aspect of the principle of least common mechanism (see Section 14.2.7).

[5] This follows from Biba's low-water-mark policy (see Section 6.2.1).

31.3.2.2 Internals

This module has three parts. The first part gathers the data on which access is to be based. The second part retrieves the access control information. The third part determines whether the data and the access control information require access to be granted.

The module queries the operating system to determine the needed data. The real user identification data is obtained through a system call, as is the current time of day. The location consists of two components: the entry point (terminal or network connection) and the remote host from which the user is accessing the local system. The latter component may indicate that the entry point is directly connected to the system, rather than using a remote host.

Part I: Obtain user ID, time of day, entry point, and remote host.

Next, the module must access the access control information. The access control information resides in a file. The file contains a sequence of records of the following form:

```
role account
user names
locations from which the role account can be accessed
times when the role account can be accessed
command and arguments
```

If the "command and arguments" line is omitted, the user is granted unrestricted access. Multiple command lines may be listed in a single record.

Part II: Obtain a handle (or descriptor) to the access control information. The programmer will use this handle to read the access control records from the access control information.

Finally, the program iterates through the access control information. If the role in the current record does not match the requested role, it is ignored. Otherwise, the user name, location, time, and command are compared with the appropriate fields of the record. If they all match, the module releases the handle and returns success.[6] If any of them does not match, the module continues on to the next record. If the module reaches the end of the access control information, the handle is released and the module returns failure. Note that records never deny access, but only grant it. The default action is to deny. Granting access requires an explicit record.[7]

If any record is invalid (for example, if there is a syntax error in one of the fields or if the user field contains a nonexistent user name), the module logs the error and ignores the record. This again follows the principle of fail-safe defaults, in which the system falls into a secure state when there is an error.

---

[6] If the time interval during which access is allowed expires after the access control check but before the access is granted, Requirement 31.1 is met (as it refers to the time of request). This eliminates a possible race condition.

[7] See Section 14.2.2, "Principle of Fail-Safe Defaults."

Part III: Iterate through the records until one matches the data or there are no more records. In the first case, return success; in the second case, return failure.

31.3.2.3 Storage of the Access Control Data

The system administrators of the local system control access to privileged accounts. To keep maintenance of this information simple, the administrators store the access control information in a file. Then they need only edit the file to change a user's ability to access the privileged account. The file consists of a set of records, each containing the components listed above. This raises the issue of expression. How should each part of the record be written?

For example, must each entry point be listed, or are wildcards acceptable? Strictly speaking, the principle of fail-safe defaults[8] says that we should list explicitly those locations from which access may be obtained. In practice, this is too cumbersome. Suppose a particular user was trusted to assume a role from any system on the Internet. Requiring the administrators to list all hosts would be time-consuming as well as infeasible. Worse, if the user were not allowed to access the role account from one system, the administrators would need to check the list to see which system was missing. This would violate the principle of least astonishment.[9] Given the dynamic nature of the Internet, this requirement would be absurd. Instead, we allow the following special host names, both of which are illegal [1365]:

any (a wildcard matching any system)
local (matches the local host name)

In BNF form, the language used to express location is

location ::= '(' *location* ')' | 'not' *location* | *location* 'or' *location* | *basic*
basic ::= '*any*' | '*local*' | '.' *domain* | *host*

where *domain* and *host* are domain names and host names, respectively. The strings in single quotation marks are literals. The parentheses are grouping operators, the "not" complements the associated locations, and the "or" allows either location.

EXAMPLE: A user is allowed to assume a role only when logged into the local system, the system "control.fixit.com", and the domain "watchu.edu". The appropriate entry would be

```
*local* | control.fixit.com | .watchu.edu
```

---

[8] See Section 14.2.2.

[9] See Section 14.2.8.

A similar question arises for times. Ignoring how times are expressed, how do we indicate when users may access the role account? Considerations similar to those above lead us to the following language, in which the keyword

any

allows access at any time. In BNF form, the language used to express time is

time ::= ‘(’ *time* ‘)’ | ‘not’ *time* | *time* ‘or’ *time* | *time time* | *time* ‘–’ *time* | *basic*
basic ::= *day_of_year day_of_week time_of_day* | ‘*any*’
day_of_year ::= *month* [*day*] [‘,’*year*] | *nmonth* ‘/’ [*day* ‘/’] *year* | *empty*
day_of_week ::= ‘Sunday’ | . . . | ‘Saturday’ | ‘Weekend’ | ‘Weekday’ | *empty*
time_of_day ::= *hour* [‘:’ *min*] [‘:’ *sec*] [‘AM’ | ‘PM’] | *special* | *empty*
special ::= ‘noon’ | ‘midnight’ | ‘morning’ | ‘afternoon’ | ‘evening’
empty ::= ‘’

where *month* is a string naming the month, *nmonth* is an integer naming the month, *day* is an integer naming the day of the month, and *year* is an integer specifying the year. Similarly, *hour*, *min*, and *sec* are integers specifying the hour, minute, and second. If *basic* is empty, it is treated as not allowing access.[10]

Example: A user is allowed to assume a role between the hours of 9 o’clock in the morning and 5 o’clock in the evening on Monday through Thursday. An appropriate entry would be

Monday-Thursday 9a.m.-5p.m.

This is different than saying

Monday 9a.m.-Thursday 5p.m.

because the latter allows access on Monday at 10 p.m., whereas the former does not.

Finally, the users field of the record has a similar structure:

any

In BNF form, the language used to express the set of users who may access a role is

userlist ::= ‘(’ *userlist* ‘)’ | ‘not’ *userlist* | *userlist* ‘,’ *userlist* | *user*

where *user* is the name of a user on the system.

---

[10] By the principle of fail-safe defaults (see Section 14.2.2).

These "little languages" are straightforward and simple (but incomplete; see Exercise 5). Various implementation details, such as allowing abbreviations for day and month names, can be added, as can an option to change the American expression of days of the year to an international one. These points must be considered in light of where the program is to be used. Whatever changes are made, the administrators must be able to configure times and places quickly and easily, and in a manner that a reader of the access control file can understand quickly.[11]

The listing of commands requires some thought about how to represent arguments. If no arguments are listed, is the command to be run without arguments, or should it allow any set of arguments? Conversely, if arguments are listed, should the command be run only with those arguments? Our approach is to force the administrator to indicate how arguments are to be treated.

Each command line contains a command followed by zero or more arguments. If the first word after the command is an asterisk (" * "), then the command may be run with any arguments. Otherwise, the command must be run with the exact arguments provided.

EXAMPLE: Charles is allowed to run the install command when he accesses the *bin* role. He may supply any arguments. The line in the access control file is

```
    /bin/install *
```

He may also copy the file log from the current working directory to the directory */var/install*. The line for this is

```
/bin/cp log /var/install/log
```

Finally, he may run the *id* command to ensure that he is working as *bin*. He may not supply other arguments to the command, however. This would be expressed by

```
/usr/bin/id
```

The user must type the command as given in the access control file. The full path names are present to prevent the user from accidentally executing the command *id* with *bin* privileges when *id* is a command in the local directory, rather than the system *id* command.[12]

---

[11] See Section 14.2.8, "Principle of Least Astonishment."

[12] See Chapter 23, "Malware."

31.4 Refinement and Implementation

This section focuses on the access control module of the program. We refine the high-level design presented in the preceding section until we produce a routine in a programming language.

31.4.1 First-Level Refinement

Rather than use any particular programming language, we first implement the module in pseudocode. This requires two decisions. First, the implementation language will be block-structured, like C or Java, rather than functional, like Scheme or ML. Second, the environment in which the program will function will be a UNIX-like system such as FreeBSD or Linux.

The basic structure of the security module is

```
boolean accessok(role rname, command cmd);
    status ← false
    user ← obtain user ID
    timeday ← obtain time of day
    entry ← obtain entry point (terminal line, remote host)
    open access control file
    repeat
      currecord ← obtain next record from file; EOF if none
      if currecord ≠ EOF then
          status ← match(currecord, rname, cmd, user,
                         timeday, entry)
    until currecord = EOF or status = true
    close access control file
    return status
```

We now verify that this sketch matches the design. Clearly, the interface is unchanged. The variable *status* will contain the status of the access control file check, becoming true when a match is found. Initially, it is set to false (deny access) because of the principle of fail-safe defaults. If *status* were not set, and the access control file were empty, *status* would never be set and the returned value would be undefined.

The next three lines obtain the user ID, the current time of day, and the system entry point. The following line opens the access control file.

The routine then iterates through the records of that file. The iteration has two requirements—that if any record allows access, the routine is to return true, and that if no record grants access, the routine is to return false. From the structure of the file, one cannot create a record to deny access. By default, access is denied. Entries explicitly grant access. So, iterating over the records of the file either produces a record that grants access (in which case the match routine returns true, terminating the loop and causing *accessok* to return with a value of true) or produces no such record. In that case, *status* is false, and *currecord* is set to EOF when the records in the access control file are exhausted. The loop then terminates, and the routine returns the value of *status*, which is false. Hence, this pseudocode matches the design and, transitively, the requirements.

31.4.2 Second-Level Refinement

Now we will focus on mapping the pseudocode above to a particular language and system. The C programming language is widely available and provides a convenient interface to UNIX-like systems. Given that our target system is a UNIX-like system, C is a reasonable choice. As for the operating system, there are many variants of the UNIX operating system. However, they all have fundamental similarities. The Linux operating system will provide the interfaces discussed below, and they work on a wide variety of UNIX systems.

On these systems, roles are represented as normal user accounts. The *root* account is really a role account,[13] for example. Each user account has two distinct representations of identity:[14] an internal user type *uid_t*,[15] and a string (name). When a user specifies a role, either representation may be used. For our purposes, we will assume that the caller of the *accessok* routine provides the *uid_t* representation of the role identity. Two reasons make this representation preferable. First, the target systems are unable to address privilege in terms of names, because, within the kernel, process identity is always represented by a *uid_t*. So the routines will need to do the conversion anyway. The second reason is more complex. Roles in the access control file can be represented by numbers or names. The routine for reading the access control file records will convert the roles to *uid_t*s to ensure consistency of representation. This also allows the input routine to check the records for consistency with the system environment. Specifically, if the role name refers to a nonexistent account, the routine can ignore the record. So any comparisons would require the role from the interface to be converted to a *uid_t*.

This leads to a design decision: represent all user and role IDs as integers internally. Fortunately, none of the design decisions discussed so far depend on the representation of identity, so we need not review or change our design.

Next, consider the command. On the target system, a command consists of a program name followed by a sequence of words, which are the command-line arguments to the command. The command representation is an array of strings, in which the first string is the program name and the other strings are the command-line arguments.

---

[13] See Section 15.4, "Groups and Roles."

[14] See Section 15.3, "Users."

[15] On Linux systems, and on most UNIX-like systems, this is an integer.

Putting this all together, the resulting interface is

```
int accessok(uid_t rname, char *cmd[])
```

Next comes obtaining the user ID. Processes in the target system have several identities, but the key ones are the *real UID* (which identifies the user running the process) and the *effective UID* (which identifies the privileges with which the process runs).[16] The effective UID of this program must have *root* privileges (see Exercise 4) regardless of who runs the process. Hence, it is useless for this purpose. Only the real UID identifies the user running the program. So, to obtain the user ID of the user running the program, we use

```
userid = getuid();
```

The time of day is obtained from the system and expressed in internal format. The internal representation can be given in seconds since a specific date and time (the *epoch*)[17] or in microseconds since that time. It is unlikely that times will need to be specified in microseconds in the access control file. For both simplicity of code and simplicity of the access control data,[18] the internal format of seconds will be used. So, to obtain the current time, we use

```
timeday = time(NULL);
```

Finally, we need to obtain the location. There is no simple method for obtaining this information, so we defer it until later by encapsulating it in a function. This also localizes any changes should we move this program to a different system (for example, the methods used on a Linux system may differ from those used on a FreeBSD system).

```
entry = getlocation();
```

Opening the access control file for reading is straightforward:

```
if ((fp = fopen(acfile, "r")) == NULL){
     logerror(errno, acfile);
     return(0);
}
```

---

[16] See Section 15.3, “Users.”

[17] On Linux and most other UNIX-like systems, the epoch is midnight on January 1, 1970 (UTC).

[18] See Section 14.2.3, “Principle of Economy of Mechanism,” and Section 14.2.8, “Principle of Least Astonishment.”

Notice first the error checking, and the logging of information on an error. The variable `errno` is set to a code indicating the nature of the error. The variable `acfile` points to the access control file name. The processing of the access control records follows:

```
do {
    acrec = getnextacrec(fp);
    if (acrec != NULL)
        status = match(acrec, rname, cmd, user, timeday,
                        entry);
} while (acrec == NULL || status == 1);
```

Here, we read in the record—assuming that any records remain—and check the record to see if it allows permission. This looping continues until either some record indicates that permission is to be given or all records are checked. The exact internal record format is not yet specified; hence, the use of functions. The routine concludes by closing the access control file and returning status:

```
(void) fclose(fp);
return(status);
```

31.4.3 Functions

Three functions remain: the function for obtaining location, the function for getting an access control record, and the function for checking the access control record against the information of the current process. Each raises security issues.

31.4.3.1 Obtaining Location

UNIX and Linux systems write the user's account name, the name of the terminal on which the login takes place, the time of login, and the name of the remote host (if any) to the *utmp* file. Any process may read this file. As each new process runs, it may have an associated terminal. To determine the *utmp* record associated with the process, a routine may obtain the associated terminal name, open the *utmp* file, and scan through the record to find the one with the corresponding terminal name. That record contains the name of the host from which the user is working.

This approach, although clumsy, works on most UNIX and Linux systems. It suffers from two problems related to security.

1. If any process can alter the *utmp* file, its contents cannot be trusted. Several security holes have occurred because any process could alter the *utmp* file [2254].
2. A process may have no associated terminal. Such a detached process must be mapped into the corresponding *utmp* record through other means.

> However, if the *utmp* record contains only the information described above, this is not possible because the user may be logged into multiple terminals. The issue does not arise if the process has an associated terminal, because only one user at a time may be logged into a terminal.

In the first case, we make a design decision that if the data in the *utmp* file cannot be trusted because any process can alter that file, we return a meaningless location. Then, unless the location specifier of the record allows access from any location, the record will not match the current process information and will not grant access. A similar approach works if the process does not have an associated terminal.

The outline of this routine is

```
hostname getlocation()
    status ← false
    myterm ← name of terminal
    obtain access control list for utmp
    if any user other than root can alter it then
        return "*nowhere*"
    open utmp
    repeat
        term ← obtain next entry from utmp; otherwise EOF
        if term ≠ EOF and myterm = term then
                status ← true
    until term = EOF or status = true
    if host field of utmp entry = empty
        host = "localhost"
    else
        host = host field of utmp entry
    close utmp
    return host
```

We omit the implementation due to space limitations.

31.4.3.2 The Access Control Record

The format of the records in the access control file affects both the reading of the file and the comparison with the process information, so we design it here.

Our approach is to consider the match routine first. Four items must be checked: the user name, the location, the time, and the command. Consider these items separately.

The user name is represented as an integer. Thus, the internal format of the user field of the access control record must contain either integers or names that the match routine can convert to integers. If a match occurs before all user names have been checked, then the program needs to convert no more names to integers. So, we adopt the strategy of representing the user field as a string read directly from the file. The match routine will parse the line and will use lazy evaluation to check whether or not the user ID is listed.

A similar strategy can be applied to the location and the set of commands in the record.

The time is somewhat different, because in the previous two cases, the process user ID and the location had to match one of the record entries exactly. However, the time does not have to do so. Time in the access control record is (almost always) a range. For example, the entry "May 30" means any time on the date of May 30. The day begins at midnight and ends at midnight, 24 hours later. So, the range would be from May 30 at midnight to May 31 at midnight, or in internal time (for example) between 1527638400 and 1527724800. In those rare cases in which a user may assume a role only at a precise second, the range can be treated as having the same beginning and ending points. Given this view of time as ranges, checking that the current time falls into an acceptable range suggests having the match routine parse the times and checking whether or not the internal system time falls in each range as it is constructed.

This means that the routine for reading the record may simply load the record as a sequence of strings and let the match routine do the interpretation. This yields the following structure:

```
record
    role rname
    string userlist
    string location
    string timeofday
    string commands[]
    integer numcommands
end record;
```

The *commands* field is an array of strings, each command and argument being one string, and *numcommands* containing the number of commands.

Given this information, the function used to read the access control records, and the function used to match them with the current process information, are not hard to write, but error handling does deserve some mention.

31.4.3.3 Error Handling in the Reading and Matching Routines

Assume that there is a syntax error in the access control file. Perhaps a record specifies a time incorrectly (for example, "Thurxday"), or a record divider is garbled. How should the routines handle this?

The first observation is that they cannot ignore the error. To do so violates basic principles of security (specifically, the principle of least astonishment[19]). It also defeats the purpose of the program, because access will be denied to users who need it.[20] So, the program must produce an indication of error. If it is printed, then the user will see it and should notify the system administrator maintaining the access control file. Should the user forget, the administrator will not know of the error. Hence, the error must be logged. Whether or not the user should be told why the error has occurred is another question. One school of thought holds that the more information users have, the more helpful they will be. Another school holds that information should be denied unless the user needs to know it, and in the case of an error in the access control file, the user only needs to know that access will be denied.

---

[19] See Section 14.2.8, "Principle of Least Astonishment."

Hence, the routines must log information about errors. The logged information must enable the system administrator to locate the error in the file. The error message should include the access control file name and line or record number. This suggests that both routines need access to that information. Hence, the record counts, line numbers, and file name must be shared. For reasons of modularity, this implies that these two routines should be in a submodule of the access checking routine. If they are placed in their own module, no other parts of the routine can access the line or record numbers (and none need to, given the design described here). If the module is placed under the access control routine, no external functions can read records from the access control file or check data against that file's contents.

31.4.4 Summary

This section has examined the development of a program for performing a security-critical function. Beginning with a requirements analysis, the design and parts of the implementation demonstrate the need for repeated analysis to ensure that the design meets the requirements and that design decisions are documented. From the point at which the derivation stopped, the implementation is simple.

We will now discuss some common security-related programming problems. Then we will discuss testing, installation, and maintenance.

31.5 Common Security-Related Programming Problems

Unfortunately, programmers are not perfect. They make mistakes. These errors can have disastrous consequences in programs that change the protection domains. Attackers who exploit these errors may acquire extra privileges (e.g., access to a system account such as *root* or *Administrator*). They may disrupt the normal functioning of the system by deleting or altering services over which they should have no control. They may simply be able to read files to which they should have no access.[21] So the problem of avoiding these errors, or security holes, is a necessary issue to ensure that the programs and system function as required.

---

[20] Note that a record with a syntax error will never grant access (see Exercise 6).

We present both management rules (installation, configuration, and maintenance) and programming rules together. Although there is some benefit in separating them, doing so creates an artificial distinction by implying that they can be considered separately. In fact, the limits on installation, configuration, and maintenance affect the implementation, just as the limits of implementation affect the installation, configuration, and maintenance procedures.

Researchers have developed several models for analyzing systems for these security holes.[22] These models provide a framework for characterizing the problems. The goal of the characterization guides the selection of the model. Because we are interested in technical modeling and not in the reason or time of introduction, many of the categories of the NRL model[23] are inappropriate for our needs. We also wish to analyze the multiple components of vulnerabilities rather than force each vulnerability into a particular point of view, as Aslam's model[24] does. So either the PA model[25] or the RISOS model[26] is appropriate. We have chosen the PA model for our analysis.

We examine each of the categories and subcategories separately. We consider first the general rules that we can draw from the vulnerability class, and then we focus on applying those rules to the program under discussion.

31.5.1 Improper Choice of Initial Protection Domain

Flaws involving improper choice of initial protection domain arise from incorrect setting of permissions or privileges. There are three objects for which permissions need to be set properly: the file containing the program, the access control file, and the memory space of the process. We will consider them separately.

31.5.1.1 Process Privileges

The principle of least privilege[27] dictates that no process have more privileges than it needs to complete its task, but the process must have enough privileges to complete its task successfully.

Ideally, one set of privileges should meet both criteria. In practice, different portions of the process will need different sets of privileges. For example, a process may need special privileges to access a resource (such as a log file) at the beginning and end of its task, but may not need those privileges at other times. The process structure and initial protection domain should reflect this.

---

[21] See Chapter 24, "Vulnerability Analysis."

[22] See Section 24.4, "Frameworks."

[23] See Section 24.4.3, "The NRL Taxonomy."

[24] See Section 24.4.4, "Aslam's Model."

[25] See Section 24.4.2, "Protection Analysis Model."

[26] See Section 24.4.1, "The RISOS Study."

[27] See Section 14.2.1, "Principle of Least Privilege."

> **Implementation Rule 31.1.** Structure the process so that all sections requiring extra privileges are modules. The modules should be as small as possible and should perform only those tasks that require those privileges.

The basis for this rule lies in the reference monitor.[28] The reference monitor is verifiable, complete (it is always invoked to access the resource it protects), and tamperproof (it cannot be compromised). Here, the modules are kept small and simple (verifiable), access to the privileged resource requires the process to invoke these modules (complete), and the use of separate modules with well-defined interfaces minimizes the chances of other parts of the program corrupting the module (tamperproof).

> **Management Rule 31.1.** Check that the process privileges are set properly.

Insufficient privileges could cause a denial of service. Excessive privileges could enable an attacker to exploit vulnerabilities in the program. To avoid these problems, the privileges of the process, and the times at which the process has these privileges, must be chosen and managed carefully.

One of the requirements of this program is availability (Requirements 31.1 and 31.4). On Linux and UNIX systems, the program must change the effective identity of the user from the user's account to the role account. This requires special (setuid) privileges of either the role account or the superuser.[29] The principle of least privilege[30] says that the former is better than the latter, but if one of the role accounts is *root*, then having multiple copies of the program with limited privileges is irrelevant, because the program with privileges to access the *root* role account is the logical target of attack. After all, if one can compromise a less privileged account through this program, the same attack will probably work against the *root* account. Because the Drib plans to control access to *root* in some cases, the program requires setuid to *root* privileges.

If the program does not have root privileges initially, the UNIX protection model does not allow the process to acquire them; the permissions on the program file corresponding to the program must be changed. The process must log enough information for the system administrator to identify the problem,[31] and should notify users of the problem so that the users can notify the system administrator. An alternative is to develop a server that will periodically check the permissions on the program file and reset them if needed, or a server that the program can notify should it have insufficient privileges. The designers felt that the benefits of these servers were not sufficient to warrant their development. In particular, they were concerned that the system administrators investigate any unexpected change in file permissions, and an automated server that changed the permissions back would provide insufficient incentive for an analysis of the problem.

---

[28] See Section 20.1.2.2, "Building Security In or Adding Security Later." Programs implemented following this rule are *not* reference monitors.

[29] See Section 15.3, "Users."

[30] See Section 14.2.1, "Principle of Least Privilege."

[31] See Section 25.3, "Designing an Auditing System."

As a result, the developers required that the program acquire *root* permission at start-up. The access control module is executed. Within that module, the privileges are reset to the user's once the log file and access control file have been opened.[32] Superuser privileges are needed only once more—to change the privileges to those of the role account should access be granted. This routine, also in a separate module, supplies the granularity required to provide the needed functionality while minimizing the time spent executing with *root* privileges.

31.5.1.2 Access Control File Permissions

Biba's models[33] emphasize that the integrity of the process relies on both the integrity of the program and the integrity of the access control file. The former requires that the program be properly protected so that only authorized personnel can alter it. The system managers must determine who the "authorized personnel" are. Among the considerations here are the principle of separation of duty[34] and the principle of least privilege.[35]

Verifying the integrity of the access control file is critical, because that file controls the access to role accounts. Some external mechanism, such as a file integrity checking tool, can provide some degree of assurance that the file has not changed. However, these checks are usually periodic, and the file might change after the check. So the program itself should check the integrity of the file when the program is run.

> **Management Rule 31.2.** The program that is executed to create the process, and all associated control files, must be protected from unauthorized use and modification. Any such modification must be detected.

In many cases, the process will rely on the settings of other files or on some other external resources. Whenever possible, the program should check these dependencies to ensure that they are valid. The dependencies must be documented so that installers and maintainers will understand what else must be maintained in order to ensure that the program works correctly.

> **Implementation Rule 31.2.** Ensure that any assumptions in the program are validated. If this is not possible, document them for the installers and maintainers, so they know the assumptions that attackers will try to invalidate.

---

[32] Section 14.2.3, "Principle of Complete Mediation," provides detail on why this works.

[33] See Section 6.2, "The Biba Model."

[34] See Section 6.1, "Goals."

[35] See Section 14.2.1, "Principle of Least Privilege."

The permissions of the program, and its containing directory, are to be set so only *root* can alter or move the program. According to Requirement 31.2, only *root* can alter the access control file. Hence, the file must be owned by *root*, and only *root* can write to it. The program should check the ownership and permissions of this file, and the containing directories, to validate that only *root* can alter it.

EXAMPLE: The naive way to check that only *root* can write to the file is to check that the owner is *root* and that the file permissions allow only the owner to write to it. But consider the group permissions. If *root* is the only member of the group, then the group permissions may allow members of the group to write to the file. The problem is that checking group membership is more complicated than looking up the members of the group. A user may belong to a group without being listed as a member, because the GID of the user is assigned from the password file, and group membership lists are contained in a different file.[36] Either the password file and the group membership list must both be checked, or the program should simply report an error if anyone other than the user can write to the file. For simplicity,[37] the designers chose the second approach.

31.5.1.3 Memory Protection

As the program runs, it depends on the values of variables and other objects in memory. This includes the executable instructions themselves. Thus, protecting memory against unauthorized or unexpected alteration is critical.

Consider sharing memory. If two subjects can alter the contents of memory, then one could change data on which the second relies. Unless such sharing is required (for example, by concurrent processes), it poses a security problem because the modifying process can alter variables that control the action of the other process. Thus, each process should have a protected, unshared memory space.

If the memory is represented by an object that processes can alter, it should be protected so that only trusted processes can access it. Access here includes not only modification but also reading, because passwords reside in memory after they are types. Multiple abstractions are discussed in more detail in the next section.

Implementation Rule 31.3. Ensure that the program does not share objects in memory with any other program, and that other programs cannot access the memory of a privileged process.

---

[36] Specifically, if the group field of the password file entry for *matt* is 30, and the group file lists the members of group 30 as *root*, the user *matt* is still in group 30, but a query to the group file (the standard way to determine group membership) will show that only *root* is a member.

[37] See Section 14.2.3, "Principle of Economy of Mechanism."

Interaction with other processes cannot be eliminated. If the running process obtains input or data from other processes, then that interface provides a point through which other processes can reach the memory. The most common version of this attack is the buffer overflow.

Buffer overflows involve either altering of data or injecting of instructions that can be executed later. There are a wide variety of techniques for this [32, 706].[38] Several remedies exist. For example, if buffers reside in sections of memory that are not executable, injecting instructions will not work. Similarly, if some data is to remain unaltered, the data can be stored in read-only memory.

> **Management Rule 31.3.** Configure memory to enforce the principle of least privilege. If a section of memory is not to contain executable instructions, turn execute permission off for that section of memory. If the contents of a section of memory are not to be altered, make that section read-only.

These rules appear in three ways in our program. First, the implementers use the language constructs to flag unchanging data as constant (in the C programming language, this is the keyword *const*). This will cause compile-time errors if the variables are assigned to, or runtime errors if instructions try to alter those constants.

The other two ways involve program loading. The system's loader places data in three areas: the *data* (initialized data) segment, the *stack* (used for function calls and variables local to the functions), and the *heap* (used for dynamically allocated storage). A common attack is to trick a program into executing instructions injected into three areas. The vector of injection can be a buffer overflow,[39] for example. The characteristic under discussion does not stop such alteration, but it should prevent the data from being executed by making the segments or pages of all three areas nonexecutable. This suffices for the data and stack segments and follows Management Rule 31.3.

If the program uses dynamic loading to load functions at runtime, the functions that are loaded may change over the lifetime of the program. This means that the assumptions the programmers make may no longer be valid.[40] One solution to this problem is to compile the program in such a way that it does not use dynamic loading. This also also prevents the program from trying to load a module at runtime that may be missing. This could occur if a second process deleted the appropriate library. So disabling of dynamic loading also follows Implementation Rule 31.3.[41]

Finally, some UNIX-like systems (including the one on which this program is being developed) allow execution permission to be turned off for the stack. The boot file sets the kernel flag to enforce this.

---

[38] However, alternative techniques involving corrupting data, causing the flow of control to change improperly, do work. See Section 31.5.6, "Improper Validation."

[39] Buffer overflows can also alter data. See Section 31.5.3.1, "Memory," for an example.

[40] See Section 31.5.3.2, "Changes in File Contents."

[41] Other considerations contributed. See Section 31.5.4, "Improper Naming."

31.5.1.4 Trust in the System

This analysis overlooks several system components. For example, the program relies on the system authentication mechanisms to authenticate the user, and on the user information database to map users and roles into their corresponding UIDs (and, therefore, privileges). It also relies on the inability of ordinary users to alter the system clock. If any of this supporting infrastructure can be compromised, the program will not work correctly. The best that can be done is to identify these points of trust in the installation and operation documentation so that the system administrators are aware of the dependencies of the program on the system.

> **Management Rule 31.4.** Identify all system components on which the program depends. Check for errors whenever possible, and identify those components for which error checking will not work.

For this program, the implementers should identify the system databases and information on which the program depends, and should prepare a list of these dependencies. They should discuss these dependencies with system managers to determine if the program can check for errors. When this is not possible, or when the program cannot identify all errors, they should describe the possible consequences of the errors. This document should be distributed with the program so that system administrators can check their systems before installing the program.

31.5.2 Improper Isolation of Implementation Detail

The problem of improper isolation of implementation detail arises when an abstraction is improperly mapped into an implementation detail. Consider how abstractions are mapped into implementations. Typically, some function (such as a database query) occurs, or the abstraction corresponds to an object in the system. What happens if the function produces an error or fails in some other way, or if the object can be manipulated without reference to the abstraction?

The first rule is to catch errors and failures of the mappings. This requires an analysis of the functions and a knowledge of their implementation. The action to take on failure also requires thought. In general, if the cause cannot be determined, the program should fail by returning the relevant parts of the system to the states they were in when the program began.[42]

> **Implementation Rule 31.4.** The error status of every function must be checked. Do not try to recover unless the cause of the error, and its effects, do not affect any security considerations. The program should restore the state of the system to the state before the process began, and then terminate.

---

[42] See Section 14.2.2, "Principle of Fail-Safe Defaults."

The abstractions in this program are the notion of a user and a role, the access control information, and the creation of a process with the rights of the role. We will examine these abstractions separately.

31.5.2.1 Resource Exhaustion and User Identifiers

The notion of a user and a role is an abstraction because the program can work with role names and the operating system uses integers (UIDs). The question is how those user and role names are mapped to UIDs. Typically, this is done with a user information database that contains the requisite mapping, but the program must detect any failures of the query and respond appropriately.

EXAMPLE: A mail server allowed users to forward mail by creating a forwarding file [2225]. The forwarding file could specify files to which the mail should be appended. In this case, the mail server would deliver the letter with the privileges of the owner of the forwarding file (represented on the system as an integer UID). In some cases, the mail server would queue the message for later delivery. When it did so, it would write the name (not the UID) of the user into a control file. The system queried a database, supplying the UID, and obtaining the corresponding name. If the query failed, the mail server used a default name specified by the system administrator.

Attackers discovered how to make the queries fail. As a result, the user was set to a default user, usually a system-level user (such as *daemon*). This enabled the attackers to have the mail server append mail to any file to which the default user could write. They used this to implant Trojan horses into system programs. These Trojan horses gave them extra privileges, compromising the system.

The designers and implementers decided to have the program fail if, for any reason, the query failed. This application of the principle of fail-safe defaults[43] ensured that in case of error, the users would not get access to the role account.

31.5.2.2 Validating the Access Control Entries

The access control information implements the access control policy (an abstraction). The expression of the access control information is therefore the result of mapping an abstraction to an implementation. The question is whether or not the given access control information correctly implements the policy. Answering this question requires someone to examine the implementation expression of the policy.

The programmers developed a second program that used the same routines as the role-assuming program to analyze the access control entries. This program prints the access control information in an easily readable format. It allows the system managers to check that the access control information is correct. A specific procedure requires that this information be checked periodically, and always after the file or the program is altered.

---

[43] See Section 14.2.2, "Principle of Fail-Safe Defaults."

31.5.2.3 Restricting the Protection Domain of the Role Process

Creating a role process is the third abstraction. There are two approaches. Under UNIX-like systems, the program can spawn a second, *child*, process. It can also simply start up a second program in such a way that the parent process is replaced by the new process. This technique, called *overlaying*, is intrinsically simpler than creating a child process and exiting. It allows the process to replace its own protection domain with the (possibly) more limited one corresponding to the role. The programmers elected to use this method. The new process inherits the protection domain of the original one. Before the overlaying, the original process must reset its protection domain to that of the role. The programmers do so by closing all files that the original process opened, and changing its privileges to those of the role.

EXAMPLE: The effective UIDs and GIDs[44] control privileges. Hence, the programmers reset the effective GID first, and then the effective UID (if resetting were done in the opposite order, the change to GIDs would fail because such changes require *root* privileges). However, if the UNIX-like system supports saved UIDs, an authorized user may be able to acquire *root* privileges even if the role account is not *root*. The problem is that resetting the effective UID sets the saved UID to the previous UID—namely, *root*. A process may then reacquire the rights of its saved UID. To avoid this problem, the programmers used the *setuid* system call to reset *all* of the real, effective, and saved UIDs to the UID of the role. Thus, all traces of the *root* UID are eliminated and the user cannot reacquire those privileges.

Similarly, UNIX-like systems check access permissions only when the file is opened. If a *root* process opens a privileged file and then the process drops *root* privileges, it can still read from (or write to) the file.

The components of the protection domain that the process must reset before the overlay are the open files (except for standard input, output, and error), which must be closed, the signal handlers, which must be reset to their default values, and any user-specific information, which must be cleared.

31.5.3 Improper Change

This category describes data and instructions that change over time. The danger is that the changed values may be inconsistent with the previous values. The previous values dictate the flow of control of the process. The changed values cause the program to take incorrect or nonsecure actions on that path of control.

The data and instructions can reside in shared memory, in nonshared memory, or on disk. The last includes file attribute information such as ownership and access control list.

---

[44] See Section 15.3, "Users."

31.5.3.1 Memory

First comes the data in shared memory. Any process that can access shared memory can manipulate data in that memory. Unless all processes that can access the shared memory implement a concurrent protocol for managing changes, one process can change data on which a second process relies. As stated above, this could cause the second process to violate the security policy.

EXAMPLE: Two processes share memory. One process reads authentication data and writes it into the shared memory space. The second process performs the authentication, and writes a boolean *true* back into the shared memory space if the authentication succeeds, and *false* if it fails. Unless the two processes use concurrent constructs to synchronize their reading and writing, the first process may read the result before the second process has completed the computation for the current data. This could allow access when it should be denied, or vice versa.

> **Implementation Rule 31.5.** If a process interacts with other processes, the interactions should be synchronized. In particular, all possible sequences of interactions must be known and, for all such interactions, the process must enforce the required security policy.

A variant of this situation is the asynchronous exception handler. If the handler alters variables and then returns to the previous point in the program, the changes in the variables could cause problems similar to the problem of concurrent processes. For this reason, if the exception handler alters any variables on which other portions of the code depend, the programmer must understand the possible effects of such changes. This is just like the earlier situation in which a concurrent process changes another's variables in a shared memory space.

> **Implementation Rule 31.6.** Asynchronous exception handlers should not alter any variables except those that are local to the exception handling module. An exception handler should block all other exceptions when begun, and should not release the block until the handler completes execution, unless the handler has been designed to handle exceptions within itself (or calls an uninvoked exception handler).

A second approach applies whether the memory is shared or not. A user feeds bogus information to the program, and the program accepts it. The bogus data overflows its buffer, changing other data, or inserting instructions that can be executed later.

EXAMPLE: The buffer overflow attack on *fingerd* described in Section 24.4.5.2 illustrates this approach. The return address is pushed onto the stack when the input routine is called. That address is not expected to change between its being pushed onto the stack and its being popped from the stack, but the buffer overflow changes it. When the input function returns, the address popped from the stack is that of the input buffer. Execution resumes at that point, and the input instructions are used.

This suggests one way to detect such transformations (the *stack guard approach*) [469]. Immediately after the return address is pushed onto the stack, push a random number onto the stack (the *canary*). Assume that the input overflows the buffer on the stack and alters the return address on the stack. If the canary is n bits long and has been chosen randomly, the probability of the attacker not changing that cookie is 2^{-n}. When the input procedure returns, the canary is popped and compared with the value that was pushed onto the stack. If the two differ, there has been an overflow.[45]

In terms of trust, the return address (a trusted datum) can be affected by untrusted data (from the input). This lowers the trustworthiness of the return address to that of input data. One need not supply instructions to breach security.

EXAMPLE: One (possibly apocryphal) version of a UNIX login program allocated two adjacent arrays. The first held the user's cleartext password and was 80 characters long, and the second held the password hash[46] and was 13 characters long. The program's logic loaded the password hash into the second array as soon as the user's name was determined. It then read the user's cleartext password and stored it in the first array. If the contents of the first array hashed to the contents of the second array, the user was authenticated. An attacker simply selected a random password (for example, "password") and generated a valid hash for it (here, "12CsGd8FRcMSM"). The attacker then identified herself as *root*. When asked for a password, the attacker entered "password", typed 72 spaces, and then typed "12CsGd8FRcMSM". The system hashed "password", got "12CsGd8FRcMSM", and logged the user in as *root*.

A technique in which canaries protect data, not only the return address, would work, but raises many implementation problems (see Exercise 7).

> **Implementation Rule 31.7.** Whenever possible, data that the process trusts and data that it receives from untrusted sources (such as input) should be kept in separate areas of memory. If data from a trusted source is overwritten with data from an untrusted source, a memory error will occur.

In more formal terms, the principle of least common mechanism[47] indicates that memory should not be shared in this way.

---

[45] If the goal is to alter data on the stack other than the return address, the canary will not be altered. This technique will not detect the change. (See Exercise 7.)

[46] See Section 13.2, "Passwords."

[47] See Section 14.2.7, "Principle of Least Common Mechanism."

These rules apply to our program in several ways. First, the program does not interact with any other program except through exception handling.[48] So Implementation Rule 31.5 does not apply. Exception handling consists of calling a procedure that disables further exception handling, logs the exception, and immediately terminates the program.

Illicit alteration of data in memory is the second potential problem. If the user-supplied data is read into memory that overlaps with other program data, it could erase or alter that data. To satisfy Implementation Rule 31.7, the programmers did not reuse variables into which users could input data. They also ensured that each access to a buffer did not overlap with other buffers.

The problem of buffer overflow is solved by checking all array and pointer references within the code. Any reference that is out of bounds causes the program to fail after logging an error message to help the programmers track down the error.

31.5.3.2 Changes in File Contents

File contents may change improperly. In most cases, this means that the file permissions are set incorrectly or that multiple processes are accessing the file, which is similar to the problem of concurrent processes accessing shared memory. Management Rule 31.2 and Implementation Rule 31.5 cover these two cases.

A nonobvious corollary is to be careful of dynamic loading. Dynamic load libraries are not part of this program's executable. They are loaded, as needed, when the program runs. Suppose one of the libraries is changed, and the change causes a side effect. The program may cease to function or, even worse, work incorrectly.

If the dynamic load modules cannot be altered, then this concern is minimal, but if they can be upgraded or otherwise altered, it is important. Because one of the reasons for using dynamic load libraries is to allow upgrades without having to recompile programs that depend on the library, security-related programs using dynamic load libraries are at risk.

> **Implementation Rule 31.8.** Do not use components that may change between the time the program is created and the time it is run.

This is another reason that the developers decided not to use dynamic loading.

31.5.3.3 Race Conditions in File Accesses

A race condition in this context is the *time-of-check-to-time-of-use* problem. As with memory accesses, the file being used is changed after validation but before access.[49] To thwart it, either the file must be protected so that no untrusted user can alter it, or the process must validate the file and use it indivisibly. The former requires appropriate settings of permission, so Management Rule 31.2 applies. Section 31.5.7, "Improper Indivisibility," discusses the latter.

---

[48] If the access control information or the authentication information came from servers, then there would be interaction with other programs (the servers). The method of communication would need to be considered, as discussed above.

This program validates that the owner and access control permissions for the access control file are correct (the check). It then opens the file (the use). If an attacker can change the file after the validation but before the opening, so that the file checked is not the file opened, then the attacker can have the program obtain access control information from a file other than the legitimate access control file. Presumably, the attacker would supply a set of access control entries allowing unauthorized accesses.

EXAMPLE: The UNIX operating system allows programs to refer to files in two ways: by name and by file descriptor.[50] Once a file descriptor is bound to a file, the referent of the descriptor does not change. Each access through the file descriptor always refers to the bound file (until the descriptor is closed). However, the kernel reprocesses the file name at each reference, so two references to the same file name may refer to two *different* files. An attacker who is able to alter the file system in such a way that this occurs is exploiting a race condition. So any checks made to the file corresponding to the first use of the name may not apply to the file corresponding to the second use of the name. This can result in a process making unwarranted assumptions about the trustworthiness of the file and the data it contains.

In the *xterm* example[51] the program can be fixed by opening the file and then using the file descriptor (handle) to obtain the owner and access permissions.[52] Those permissions belong to the opened file, because they were obtained using the file descriptor. The validation is now ensured to be that of the access control file.

The program does exactly this. It opens the access control file and uses the file descriptor, which references the file attribute information directly to obtain the owner, group, and access control permissions. Those permissions are checked. If they are correct, the program uses the file descriptor to read the file. Otherwise, the file is closed and the program reports a failure.

31.5.4 Improper Naming

Improper naming refers to an ambiguity in identifying an object. Most commonly, two different objects have the same name. The programmer intends the name to refer to one of the objects, but an attacker manipulates the environment and the process so that the name refers to a different object. Avoiding this flaw requires that every object be unambiguously identified. This is both a management concern and an implementation concern.

---

[49] Section 24.3.1, "Two Security Flaws," discusses this problem in detail.

[50] See Section 15.2, "Files and Objects."

[51] See Section 24.3.1, "Two Security Flaws."

[52] The system call used would be *fstat*.

Objects must be uniquely identifiable or completely interchangeable. Managing these objects means identifying those that are interchangeable and those that are not. The former objects need a controller (or set of controllers) that, when given a name, selects one of the objects. The latter objects need unique names. The managers of the objects must supply those names.

> **Management Rule 31.5.** Unique objects require unique names. Interchangeable objects may share a name.

A name is interpreted within a context. At the implementation level, the process must force its own context into the interpretation, to ensure that the object referred to is the intended object. The context includes information about the character sets, process and file hierarchies, network domains, and any accessible variables such as the search path.

EXAMPLE: Stage 3 in Section 24.2.9 discussed an attack in which a privileged program called *loadmodule* executed a second program named *ld.so*. The attack exploited *loadmodule*'s failure to specify the context in which *ld.so* was named. *Loadmodule* used the context of the user invoking the program. Normally, this caused the correct *ld.so* to be invoked. In the example, the attacker changed the context so that another version of *ld.so* was executed. This version had a Trojan horse that would grant privileged access. When the attacker executed *loadmodule*, the Trojan horse was triggered and maximum privileges were acquired.

> **Implementation Rule 31.9.** The process must ensure that the context in which an object is named identifies the correct object.

This program uses names for external objects in four places: the name of the access control file, the names of the users and roles, the names of the hosts, and the name of the command interpreter (the *shell*) that the program uses to execute commands in the role account.

The two file names (access control file and command interpreter) must identify specific files. Absolute path names specify the location of the object with respect to a distinguished directory called / or the "root directory." However, a privileged process can redefine / to be any directory.[53] This program does not do so. Furthermore, if the root directory is anything other than the root directory of the system, a trusted process has executed it. No untrusted user could have done so. Thus, as long as absolute path names are specified, the files are unambiguously named.

---

[53] Specifically, the system call *chroot* resets / to mean the named directory. All absolute path names are interpreted with respect to that directory. Only the superuser, *root*, may execute this system call.

The name provided may be interpreted in light of other aspects of the environment. For example, differences in the encoding of characters can transform file names. Whether characters are made up of 16 bits, 8 bits, or 7 bits can change the interpretation, and therefore the referent, of a file name. Other environment variables can change the interpretation of the path name. This program simply creates a new, known, safe environment for execution of the commands.[54]

This has two advantages over sanitization of the existing context. First, it avoids having the program analyze the environment in detail. The meaning of each aspect of the environment need not be analyzed and examined. The environment is simply replaced. Second, it allows the system to evolve without compromising the security of the program. For example, if a new environment variable is assigned a meaning that affects how programs are executed, the variable will not affect how this program executes its commands because that variable will not appear in the command's environment. So this program closes all file descriptors, resets signal handlers, and passes a new set of environment variables for the command.

These actions satisfy Implementation Rule 31.9.

The developers assumed that the system was properly maintained, so that the names of the users and roles would map into the correct UIDs. (Section 31.5.2.1 discusses this.) This applies to Management Rule 31.5.

The host names are the final set of names. These may be specified by names or IP addresses. If the former, they must be fully qualified domain names to avoid ambiguity. To see this, suppose an access control entry allows user *matt* to access the role *wheel* when logging in from the system *amelia*. Does this mean the system named *amelia* in the local domain, or any system named *amelia* from any domain? Either interpretation is valid. The former is more reasonable,[55] and applying this interpretation resolves the ambiguity. (The program implicitly maps names to fully qualified domain names using the former interpretation. Thus, *amelia* in the access control entry would match a host named *amelia* in the local domain, and not a host named *amelia* in another domain.) This implements Implementation Rule 31.9.[56]

As a side note, if the local network is mismanaged or compromised, the name *amelia* may refer to a system other than the one intended. For example, the real host *amelia* may crash or go offline. An attacker can then reset the address of his host to correspond to *amelia*. This program will not detect the impersonation.

31.5.5 Improper Deallocation or Deletion

Failing to delete sensitive information raises the possibility of another process seeing that data at a later time. In particular, cryptographic keywords, passwords, and other authentication information should be discarded once they have been used. Similarly, once a process has finished with a resource, that resource should be deallocated. This allows other processes to use that resource, inhibiting denial of service attacks.

---

[54] The principle of fail-safe defaults (see Section 14.2.2) supports this approach.

[55] According to the principle of least privilege (see Section 14.2.1).

[56] As discussed in Section 15.6.1, "Host Identity," host names can be spoofed. For reasons discussed in the preceding chapters, the Drib management and security officers are not concerned with this threat on the Drib's internal network.

A consequence of not deleting sensitive information is that dumps of memory, which may occur if the program receives an exception or crashes for some other reason, contain the sensitive data. If the process fails to release sensitive resources before spawning unprivileged subprocesses, those unprivileged subprocesses may have access to the resource.

> **Implementation Rule 31.10.** When the process finishes using a sensitive object (one that contains confidential information or one that should not be altered), the object should be erased, then deallocated or deleted. Any resources not needed should also be released.

Our program uses three pieces of sensitive information. The first is the cleartext password, which authenticates the user. The password is hashed, and the hash is compared with the stored hash. Once the hash of the entered password has been computed, the process must delete the cleartext password. So it overwrites the array holding the password with random bytes.

The second piece of sensitive information is the access control information. Suppose an attacker wanted to gain access to a role account. The access control entries would tell the attacker which users could access that account using this program. To prevent the attacker from gaining this information, the developers decided to keep the contents of the access control file confidential. The program accesses this file using a file descriptor. File descriptors remain open when a new program overlays a process. Hence, the program closes the file descriptor corresponding to the access control file once the request has been validated (or has failed to be validated).

The third piece of sensitive information is the log file. The program alters this file. If an unprivileged program such as one run by this program were to inherit the file descriptor, it could flood the log. Were the log to fill up, the program could no longer log failures. So the program also closes the log file before spawning the role's command.

31.5.6 Improper Validation

The problem of improper validation arises when data is not checked for consistency and correctness. Ideally, a process would validate the data against the more abstract policies to ensure correctness. In practice, the process can check correctness only by looking for error codes (indicating failure of functions and procedures) or by looking for patently incorrect values (such as negative numbers when positive ones are required).

As the program is designed, the developers should determine what conditions must hold at each interface and each block of code. They should then validate that these conditions hold.

What follows is a set of validations that are commonly overlooked. Each program requires its own analysis, and other types of validation may be critical to the correct, secure functioning of the program, so this list is by no means complete.

31.5.6.1 Bounds Checking

Errors of validation often occur when data is supposed to lie within bounds. For example, a buffer may contain entries numbered from 0 to 99. If the index used to access the buffer elements takes on a value less than 0 or greater than 99, it is an invalid operand because it accesses a nonexistent entry. The variable used to access the element may not be an integer (for example, it may be a set element or pointer), but in any case it must reference an existing element.

> **Implementation Rule 31.11.** Ensure that all array references access existing elements of the array. If a function that manipulates arrays cannot ensure that only valid elements are referenced, do not use that function. Find one that does, write a new version, or create a wrapper.

In this example program, all loops involving arrays compare the value of the variable referencing the array against the indexes (or addresses) of both the first and last elements of the array. The loop terminates if the variable's value is outside those two values. This covers all loops within the program, but it does not cover the loops in the library functions.

For loops in the library functions, bounds checking requires an analysis of the functions used to manipulate arrays. The most common type of array for which library functions are used is the character string, which is a sequence of characters (bytes) terminating with a 0 byte. Because the length of the string is not encoded as part of the string, functions cannot determine the size of the array containing the string. They simply operate on all bytes until a 0 byte is found.

EXAMPLE: The program sometimes must copy character strings (defined in C as arrays of character data terminating with a byte containing 0). The canonical function for copying strings does no bounds checking. This function, `strcpy(x, y)`, copies the string from the array *y* to the array *x*, even if the string is too long for *x*. A different function, `strncpy(x, y, n)`, copies at most *n* characters from array *y* to array *x*. However, unlike *strcpy*, *strncpy* may not copy the terminating 0 byte.[57] The program must take two actions when *strncpy* is called. First, it must insert a 0 byte at the end of the *x* array. This ensures that the contents of *x* meet the definition of a string in C. Second, the process must check that both *x* and *y* are arrays of characters, and that *n* is a positive integer.

The programmers use only those functions that bound the sizes of arrays. In particular, the function *fgets* is used to read input, because it allows the programmer to specify the maximum number of characters to be read. (This solves the problem that plagued *fingerd*.[58])

---

[57] If the string in *y* is longer than *n* characters, *strncpy* will not add a 0 byte to the characters copied into *x*.

31.5.6.2 Type Checking

Failure to check types is another common validation problem. If a function parameter is an integer, but the actual argument passed is a floating point number, the function will interpret the bit pattern of the floating point number as an integer and will produce an incorrect result.

> **Implementation Rule 31.12.** Check the types of functions and parameters.

A good compiler and well-written code will handle this particular problem. All functions should be declared before they are used. Most programming languages allow the programmer to specify the number and types of arguments, as well as the type of the return value (if any). The compiler can then check the types of the declarations against the types of the actual arguments and return values.

> **Implementation Rule 31.13.** When compiling programs, ensure that the compiler reports inconsistencies in types. Investigate all such warnings and either fix the problem or document the warning and why it is spurious.

31.5.6.3 Error Checking

A third common problem involving improper validation is failure to check return values of functions. For example, suppose a program needs to determine ownership of a file. It calls a system function that returns a record containing information from the file attribute table. The program obtains the owner of the file from the appropriate field of the record. If the function fails, the information in the record is meaningless. So, if the function's return status is not checked, the program may act erroneously.

> **Implementation Rule 31.14.** Check all function and procedure executions for errors.

This program makes extensive use of system and library functions, as well as its own internal functions (such as the access control module). Every function returns a value, and the value is checked for an error before the results of the function are used. For example, the function that obtains the ownership and access permissions of the access control file would return meaningless information should the function fail. So the function's return value is checked first for an error; if no error has occurred, then the file attribute information is used.

---

[58] See Section 24.4.5.2, "The *fingerd* Buffer Overflow."

As another example, the program opens a log file. If the open fails, and the program tries to write to the (invalid) file descriptor obtained from the function that failed, the program will terminate as a result of an exception. Hence, the program checks the result of opening the log file.

31.5.6.4 Checking for Valid, Not Invalid, Data

Validation should apply the principle of fail-safe defaults.[59] This principle requires that valid values be known, and that all other values be rejected. Unfortunately, programmers often check for invalid data and assume that the rest is valid.

EXAMPLE: A *metacharacter* is a character that is interpreted as something other than itself. For example, to the UNIX shells, the character “?” is a metacharacter that represents all single character files. A vendor upgraded its version of the command interpreter for its UNIX system. The new command interpreter (shell) treated the character “ `” (back quote) as a delimiter for a command (and hence a metacharacter). The old shell treated the back quote as an ordinary character. Included in the distribution was a program for executing commands on remote systems. The set of allowed commands was restricted. This program carefully checked that the command was allowed, and that it contained no metacharacters, before sending it to a shell on the remote system. Unfortunately, the program checked a list of metacharacters to be rejected, rather than checking a list of characters that were allowed in the commands. As a result, one could embed a disallowed command within a valid command request, because the list of metacharacters was not updated to include the back quote.

Implementation Rule 31.15. Check that a variable’s values are valid.

This program checks that the command to be executed matches one of the authorized commands. It does not have a set of commands that are to be denied. The program will detect an invalid command as one that is not listed in the set of authorized commands for that user accessing that role at the time and place allowed.

As discussed in Section 31.3.2.3, it is possible to allow all users *except some specific users* access to a role by an appropriate access control entry (using the keyword *not*). The developers debated whether having this ability was appropriate because its use could lead to violations of the principle of fail-safe defaults. On one key system, however, the only authorized users were system administrators and one or two trainees. The administrators wanted the ability to shut the trainees out of certain roles. So the developers added the keyword and recommended against its use except in that single specific situation.

---

[59] See Section 14.2.2, “Principle of Fail-Safe Defaults.”

> **Implementation Rule 31.16.** If a trade-off between security and other factors results in a mechanism or procedure that can weaken security, document the reasons for the decision, the possible effects, and the situations in which the compromise method should be used. This informs others of the trade-off and the attendant risks.

31.5.6.5 Checking Input

All data from untrusted sources must be checked. Users are untrusted sources. The checking done depends on the way the data is received: into an input buffer (bounds checking) or read in as an integer (checking the magnitude and sign of the input).

> **Implementation Rule 31.17.** Check all user input for both form and content. In particular, check integers for values that are too big or too small, and check character data for length and valid characters.

The program determines what to do on the basis of at least two pieces of data that the user provides: the role name and the command (which, if omitted, means unrestricted access).[60] Users must also authenticate themselves appropriately. The program must first validate that the supplied password is correct. It then checks the access control information to determine whether the user is allowed access to the role at that time and from that location.

The length of the input password must be no longer than the buffer in which it is placed. Similarly, the lines of the access control file must not overflow the buffer allocated for it. The contents of the lines of the access control file must make up a valid access control entry. This is most easily done by constraining the format of the contents of the file, as discussed in the next section.

An excellent example of the need to constrain user input comes from formatted print statements in C.

EXAMPLE: The *printf* function's first parameter is a character string that indicates how *printf* is to format output data. The following parameters contain the data. For example,

```
printf("%d %d\n", i, j);
```

prints the values of *i* and *j*. Some versions of this library function allow the user to store the number of characters printed at any point in the string. For example, if *i* contains 2, *j* contains 21, and *m* and *n* are integer variables,

```
printf("%d %d%n %d\n%n", i, j, &m, i, &n);
```

prints

```
2 21 2
```

---

[60] See Section 14.2.6, "Principle of Separation of Privilege."

and stores 4 in *m* and 7 in *n*, because four characters are printed before the first "%n" and seven before the second "%n" (the sequence "\n" is interpreted as a single character, the newline). Now, suppose the user is asked for a file name. This input is stored in the array *str*. The program then prints the file name with

```
printf(str);
```

If the user enters the file name "log%n", the function will overwrite some memory location with the integer 3. The exact location depends on the contents of the program stack, and with some experimentation it is possible to cause the program to change the return address stored on the stack. This leads to the buffer overflow attack described earlier.

31.5.6.6 Designing for Validation

Sometimes data cannot be validated completely. For example, in the C programming language, a programmer can test for a NULL pointer (meaning that the pointer does not hold the address of any object), but if the pointer is not NULL, checking the validity of the pointer may be very difficult (or impossible). Using a language with strong type checking is another example.

The consequence of the need for validation requires that data structures and functions be designed and implemented in such a way that they can be validated. For example, because C pointers cannot be properly validated, programmers should not pass pointers or use them in situations in which they must be validated. Methods of data hiding, type checking, and object-oriented programming often provide mechanisms for doing this.

> **Implementation Rule 31.18.** Create data structures and functions in such a way that they can be validated.

An example will show the level of detail necessary for validation. The entries in the access control file are designed to allow the program to detect obvious errors. Each access control entry consists of a block of information in the following format:

```
role name
    user comma-separated list of users
    location comma-separated list of locations
    time comma-separated list of times
    command program and arguments
    . . .
    command program and arguments
endrole
```

This defines each component of the entry. (The lines need not be in any particular order.) The syntax is well-defined, and the access control module in the program checks for syntax errors. The module also performs other checks, such as searching for invalid user names in the **user** field and requiring that the full path names of all commands be specified. Finally, note that the module computes the number of commands for the module's internal record. This eliminates a possible source of error—namely, that the user may miscount the number of commands.

In case of any error, the process logs the error, if possible, and terminates. It does not allow the user to access the role.

31.5.7 Improper Indivisibility

Improper indivisibility[61] arises when an operation is considered as one unit (indivisible) in the abstract but is implemented as two units (divisible). The race conditions discussed in Section 31.5.3.3 provide one example. The checking of the access control file attributes and the opening of that file are to be executed as one operation. Unfortunately, they may be implemented as two separate operations, and an attacker who can alter the file after the first but before the second operation can obtain access illicitly. Another example arises in exception handling. Often, program statements and system calls are considered as single units or operations when the implementation uses many operations. An exception divides those operations into two sets: the set before the exception, and the set after the exception. If the system calls or statements rely on data not changing during their execution, exception handlers must not alter the data.

Section 31.5.3 discusses handling of these situations when the operations cannot be made indivisible. Approaches to making them indivisible include disabling interrupts and having the kernel perform operations. The latter assumes that the operation is indivisible when performed by the kernel, which may be an incorrect assumption.

> **Implementation Rule 31.19.** If two operations must be performed sequentially without an intervening operation, use a mechanism to ensure that the two cannot be divided.

In UNIX systems, the problem of divisibility arises with root processes such as the program under consideration. UNIX-like systems do not enforce the principle of complete mediation.[62] For *root*, access permissions are not checked. Recall the *xterm* example in Section 24.3.1. A user needed to log information from the execution of *xterm*, and specified a log file. Before appending to that file, *xterm* needed to ensure that the real UID could write to the log file. This required an extra system call. As a result, operations that should have been indivisible (the access check followed by the opening of the file) were actually divisible. One way to make these operations indivisible on UNIX-like systems is to drop privileges to those of the real UID, then open the file. The access checking is done in the kernel as part of the open.

---

[61] This is often called "atomicity."

[62] See Section 14.2.4, "Principle of Complete Mediation."

Improper indivisibility arises in our program when the access control module validates and then opens the access control file. This should be a single operation, but because of the semantics of UNIX-like systems, it must be performed as two distinct operations. It is not possible to ensure the indivisibility of the two operations. However, it is possible to ensure that the target of the operations does not change, as discussed in Section 31.5.3, and this suffices for our purposes.

31.5.7.1 Improper Sequencing

Improper sequencing means that operations are performed in an incorrect order. For example, a process may create a lock file and then write to a log file. A second process may also write to the log file, and then check to see if the lock file exists. The first program uses the correct sequence of calls; the second does not (because that sequence allows multiple writers to access the log file simultaneously).

> **Implementation Rule 31.20.** Describe the legal sequences of operations on a resource or object. Check that all possible sequences of the program(s) involved match one (or more) legal sequences.

In our program, the sequence of operations in the design shown in Section 31.3.1.2 follows a proper order. The user is first authenticated. Then the program uses the access control information to determine if the requested access is valid. If it is, the appropriate command is executed using a new, safe environment.

A second sequence of operations occurs when privileges to the role are dropped. First, group privileges are changed to those of the role. Then all user identification numbers are changed to those of the role. A common error is to switch the user identification numbers first, followed by the change in group privileges. Because changing group privileges requires *root* privileges, the change will fail. Hence, the programmers used the stated ordering.

31.5.8 Improper Choice of Operand or Operation

Preventing errors of choosing the wrong operand or operation requires that the algorithms be thought through carefully (to ensure that they are appropriate). At the implementation level, this requires that operands be of an appropriate type and value, and that operations be selected to perform the desired functions. The difference between this type of error and improper validation lies in the program. Improper implementation refers to a validation failure. The operands may be appropriate, but no checking is done. In this category, even though the operands may have been checked, they may still be inappropriate.

EXAMPLE: The UNIX program *su* allows a user to substitute another user's identity, obtaining the second user's privileges. According to an apocryphal story, one version of this program granted the user *root* privileges if the user information database did not exist (see Exercise 10 in Chapter 14). If the program could not open the user information database file, it assumed that the database did not exist. This was an inappropriate choice of operation because one could block access to the file even when the database existed.

Assurance techniques[63] help detect these problems. The programmer documents the purpose of each function and then checks (or, preferably, others check) that the algorithms in the function work properly and that the code correctly implements the algorithms.

> **Management Rule 31.6.** Use software engineering and assurance techniques (such as documentation, design reviews, and code reviews) to ensure that operations and operands are appropriate.

Within our program, many operands and operations control the granting (and denying) of access, the changing to the role, and the execution of the command. We first focus on the access part of the program, and afterwards we consider two other issues.

First, a user is granted access only when an access control entry matches all characteristics of the current session. The relevant characteristics are the role name, the user's UID, the role's name (or UID), the location, the time, and the command. We begin by checking that if the characteristics match, the access control module returns *true* (allowing access). We also check that the caller grants access when the module returns true and denies access when the module returns *false*.

Next, we consider the user's UID. That object is of type *uid_t*. If the interface to the system database returns an object of a different type, conversion becomes an issue. Specifically, many interfaces treat the UID as an integer. The difference between the types *int* and *uid_t* may cause problems. On the systems involved, *uid_t* is an unsigned integer. Since we are comparing signed and unsigned integers, C simply converts the signed integers to unsigned integers, and the comparison succeeds. Hence, the choice of operation (comparison here) is proper.

Checking location requires the program to derive the user's location, as discussed above, and pass it to the validator. The validator takes a string and determines whether it matches the pattern in the location field of the access control entry. If the string matches, the module should continue; otherwise, it should terminate and return false.

Unlike the location, a variable of type *time_t* contains the current time. The time checking portion of the module processes the string representing the allowed times and determines if the current time falls in the range of allowed times. Checking time is different than checking location because legal times are ranges, except in one specific situation: when an allowed time is specified to the exact second. A specification of an exact time is useless, because the program may not obtain the time at the exact second required. This would lead to a denial of service, violating Requirement 31.4. Also, allowing exact times leads to ambiguity.

---

[63] See Chapter 20, "Building Systems with Assurance."

EXAMPLE: The system administrator specifies that user *matt* is allowed access to the role *mail* at 9 a.m. on Tuesdays. Should this be interpreted as *exactly* 9 a.m. (that is, 9:00:00 a.m.) or as *sometime during* the 9 a.m. hour (that is, from 9:00:00 to 9:59:59 a.m.)? The latter interprets the specification as a range rather than an exact time, so the access control module uses that interpretation.

The use of signal handlers provides a second situation in which an improper choice of operation could occur. A signal indicates either an error in the program or a request from the user to terminate, so a signal should cause the program to terminate. If the program continues to run, and then grants the user access to the role account, either the program has continued in the face of an error or it has overridden the user's attempt to terminate the program.

31.5.9 Summary

This type of top-down analysis differs from the more usual approach of taking a checklist of common vulnerabilities and using it to examine code. There is a place for each of these approaches. The top-down approach presented here is a design approach, and should be applied at each level of design and implementation. It emphasizes documentation, analysis, and understanding of the program, its interfaces, and the environment in which it executes. A security analysis document should describe the analysis and the reasons for each security-related decision. This document will help other analysts examine the program and, more importantly, will provide future developers and maintainers of the program with insight into potential problems they may encounter in porting the program to a different environment, adding new features, or changing existing features.

Once the appropriate phase of the program has been completed, the developers should use a checklist to validate that the design or implementation has no common errors. Given the complexity of security design and implementation, such checklists provide valuable confirmation that the developers have taken common security problems into account.

Appendix H lists the implementation and management rules in a convenient form.

31.6 Testing, Maintenance, and Operation

Testing provides an informal validation of the design and implementation of the program. The goal of testing is to show that the program meets the stated requirements. When design and implementation are driven by the requirements, as in the method used to create the program under discussion, testing is likely to uncover only minor problems, but if the developers do not have well-articulated requirements, or if the requirements are changed during development, testing may uncover major problems, requiring changes up to a complete redesign and reimplementation of a program. The worst mistake managers and developers can make is to take a program that does not meet the security requirements and add features to it to meet those requirements. The problem is that the basic design does not meet the security requirements. Adding security features will not ameliorate this fundamental flaw.

Once the program has been written and tested, it must be installed. The installation procedure must ensure that when a user starts the process, the environment in which the process is created matches the assumptions embodied in the design. This constrains the configuration of the program parameters as well as the manner in which the system is configured to protect the program. Finally, the installers must enable trusted users to modify and upgrade the program and the configuration files and parameters.

31.6.1 Testing

The results of testing a program are most useful if the tests are conducted in the environment in which the program will be used (the production environment). So, the first step in testing a program is to construct an environment that matches the production environment. This requires the testers to know the intended production environment. If there are a range of environments, the testers must test the programs in all of them. Often there is overlap between the environments, so this task is not so daunting as it might appear.

The production environment should correspond to the environment for which the program was developed. A symptom of discrepancies between the two environments is repeated failures resulting from erroneous assumptions. This indicates that the developers have implicitly embedded information from the development environment that is inconsistent with the testing environment. This discrepancy must be reconciled.

The testing process begins with the requirements. Are they appropriate? Do they solve the problem? This analysis may be moot (if the task is to write a program meeting the given requirements), but if the task is phrased in terms of a problem to be solved, the problem drives the requirements. Because the requirements drive the design of the program, the requirements must be validated before designing begins.

As many of the software life cycle models indicate, this step may be revisited many times during the development of the program. Requirements may prove to be impossible to meet, or may produce problems that cannot be solved without changing the requirements. If the requirements are changed, they must be reanalyzed and verified to solve the problem.

Then comes the design. Section 31.4 discusses the stepwise refinement of the program. The decomposition of the program into modules allows us to test the program as it is being implemented. Then, once it has been completed, the testing of the entire program should demonstrate that the program meets its requirements in the given environment.

The general philosophy of testing is to execute all possible paths of control and compare the results with the expected results. In practice, the paths of control are too numerous to test exhaustively. Instead, the paths are analyzed and ordered. Test data is generated for each path, and the testers compare the results obtained from the actual data with the expected results. This continues until as many paths as possible have been tested.

For security testing, the testers must test not only the most commonly used paths but also the *least commonly used* paths.[64] The latter often create security problems that attackers can exploit. Because they are relatively unused, traditional testing places them at a lower priority than that of other paths. Hence, they are not as well scrutinized, and vulnerabilities are missed.

The ordering of the paths relies on the requirements. Those paths that perform multiple security checks are more critical than those that perform single (or no) security checks because they introduce interfaces that affect security requirements. The other paths affect security, of course, but there are no interfaces.

First, we examine a module that calls no other module. Then we examine the program as a composition of modules. We conclude by testing the installation, configuration, and use instructions.

31.6.1.1 Testing the Module

The module may invoke one or more functions. The functions return results to the caller, either directly (through return values or parameter lists) or indirectly (by manipulation of the environment). The goal of this testing is to ensure that the module exhibits correct behavior regardless of what the functions returns.

The first step is to define "correct behavior." During the design of the program, the refinement process led to the specification of the module and the module's interface. This specification defines "correct behavior," and testing will require us to check that the specification holds.

We begin by listing all interfaces to the module. We will then use this list to execute four different types of tests. The types of test are as follows:

1. *Normal data tests*. These tests provide unexceptional data. The data should be chosen to exercise as many paths of control through the module as possible.
2. *Boundary data tests*. These tests provide data that tests any limits to the interfaces. For example, if the module expects a string of up to 256 characters to be passed in, these tests invoke the module and pass in arrays of 255, 256, and 257 characters. Longer strings should also be used in an effort to overflow internal buffers. The testers can examine the source code to determine what to try. Limits here do not apply simply to arrays or strings. In the program under discussion, the lowest allowed UID is 0, for *root*. A good test would be to try a UID of -1 to see what happens. The module should report an error.

---

[64] See Section 20.3.3.1, "Security Testing."

EXAMPLE: One UNIX system had UIDs of 16 bits. The system used a file server that would not allow a client's *root* user to access any files. Instead, it remapped root's UID to the public UID of −2. Because that UID was not assigned to any user, the remapped root could access only those files that were available to all users. The limit problem arose because one user, named Mike, had the UID 65534. Because 65534 = −2 in two's complement 16-bit arithmetic, the remote root user could access all of Mike's files—even those that were not publicly available.

3. *Exception tests*. These tests determine how the program handles interrupts and traps. For example, many systems allow the user to send a signal that causes the program to trap to a signal handler, or to take a default action such as dumping the contents of memory to a core file. These tests determine if the module leaves the system in a nonsecure state—for example, by leaving sensitive information in the memory dump. They also analyze what the process does if ordinary actions (such as writing to a file) fail.

EXAMPLE: An FTP server ran on a system that kept its authentication information confidential. An attacker found that she could cause the system to crash by sending an unexpected sequence of commands, causing multiple signals to be generated before the first signal could be handled. The crash resulted in a core dump. Because the server would be restarted automatically, the attacker simply connected again and downloaded the core dump. From that dump, she extracted the authentication information and used a dictionary attack[65] to obtain the passwords of several users.

4. *Random data tests*. These tests supply inputs generated at random and observe how the module reacts. They should not corrupt the state of the system. If the module fails, it should restore the system to a safe state.[66]

EXAMPLE: In a study of UNIX utilities [1345], approximately 30% crashed when given random inputs. In one case, an unprivileged program caused the system to crash. In 1995, a retest showed some improvement, but still "significant rates of failure" [1346, p. 1]. Other tested systems fared little better [705, 1344].

Throughout the testing, the testers should keep track of the paths taken. This allows them to determine how complete the testing is. Because these tests are highly informal, the assurance they provide is not as convincing as the techniques discussed in Chapter 20. However, it is more than random tests, or no tests, would provide.

---

[65] See Section 13.4, "Attacking Passwords."

[66] See Section 14.2.2, "Principle of Fail-Safe Defaults."

31.6.2 Testing Composed Modules

Now consider a module that calls other modules. Each of the invoked modules has a specification describing its actions. So, in addition to the tests discussed in the preceding section, one other type of test should be performed.

5. *Error handling tests*. These tests assume that the called modules violate their specifications in some way. The goal of these tests is to determine how robust the caller is. If it fails gracefully, and restores the system to a safe state, then the module passes the test. Otherwise, it fails and must be rewritten.

EXAMPLE: Assume that a security-related program, running with *root* privileges, logs all network connections to a UNIX system. It also sends mail to the network administrator with the name of the connecting host on the subject line. To do this, it executes a command such as

```
mail -s hostname netadmin
```

where *hostname* is the name of the connecting host. This module obtains *hostname* from a different module that is passed the connecting host's IP address and uses the Domain Name Service to find the corresponding host name. A serious problem arose because the DNS did not verify that *hostname* was composed of legal characters. The effects were discovered when one attacker changed the name of his host to

```
hi nobody; rm -rf *; true
```

causing the security-related program to delete critical files. Had the calling module expected failure, and checked for it, the error would have been caught before any damage was done.

31.6.3 Testing the Program

Once the testers have assembled the program and its documentation, the final phase of testing begins. The testers have someone follow the installation and configuration instructions. This person should not be a member of the testing team, because the testing team has been working with the program and is familiar with it. The goal of this test is to determine if the installation and configuration instructions are correct and easy to understand. The principle of least astonishment[67] requires that the tool be as easy to install and use as possible. Because most installers and users will not have experience with the program, the testers need to evaluate how they will understand the documentation and whether or not they can install the program correctly by following the instructions. An incorrectly installed security tool does not provide security; it may well detract from it. Worse, it gives people a false sense of security.

---

[67] See Section 14.2.8, "Principle of Least Astonishment."

31.7 Distribution

Once the program has been completed, it must be distributed. Distribution involves placing the program in a repository where it cannot be altered except by authorized people, and from which it can be retrieved and sent to the intended recipients. This requires a policy for distribution. Specific factors to be considered are as follows.

1. *Who can use the program*? If the program is licensed to a specific organization, or to a specific host, then each copy of the program that is distributed must be tied to that organization or host so it cannot be redistributed or pirated. This is an originator controlled policy.[68] One approach is to provide the licensee with a secret key and encipher the software with the same key. This prevents redistribution without the licensee's consent, unless the attacker breaks the cryptosystem or steals the licensee's key.[69]
2. *How can the integrity of the master copy be protected*? If an attacker can alter the master copy, from which distribution copies are made, then the attacker can compromise all who use the program.

EXAMPLE: The program *tcp_wrappers* provides host-level access control for network servers. It is one of the most widely used programs in the UNIX community. In 1996, attackers broke into the site from which that program could be obtained [2238]. They altered the program to allow all connections to succeed. More than 50 groups obtained the program before the break-in was detected.

Part of the problem is credibility. If an attacker can pose as the vendor, then all who obtain the program from the attacker will be vulnerable to attack. This tactic undermines trust in the program and can be surprisingly hard to counter. It is analogous to generating a cryptographic checksum for a program infected with a computer virus.[70] When an uninfected program is obtained, the integrity checker complains because the checksum is wrong. In our example, when the real vendor contacts the duped customer, the customer usually reacts with disbelief, or is unwilling to concede that his system has been compromised.

---

[68] See Section 8.3, "Originator Controlled Access Control."

[69] See Section 14.2.5, "Principle of Open Design."

[70] See Section 23.9.1, "Scanning Defenses."

3. *How can the availability of the program be ensured*? If the program is sent through a physical medium, such as a read-only DVD, availability is equivalent to the availability of mail or messenger services between the vendor and the buyer. If the program is distributed through electronic means, however, the distributor must take precautions to ensure that the distribution site is available. Denial of service attacks such as SYN flooding may hamper the availability.

Like a program, the distribution is controlled by a policy. All considerations that affect a security policy affect the distribution policy as well.

31.8 Summary

This chapter discussed informal techniques for writing programs that enforce security policies. The process began with a requirements analysis and continued with a threat analysis to show that the requirements countered the threats. The design process came next, and it fed back into the requirements to clarify an ambiguity. Once the high-level design was accepted, we used a stepwise refinement process to break the design down into modules and a caller. The categories of flaws in the program analysis vulnerability helped find potential implementation problems. Finally, issues of testing and distribution ensured that the program did what was required.

31.9 Research Issues

The first research issue has to do with analysis of code. How can one analyze programs to discover security flaws? This differs from the sort of analysis that is performed in the development of high-assurance systems, because the program and system are already in place. The goal is to determine what, and where, the problems are. Some researchers are developing analysis tools for specific problems such as buffer overflows and race conditions. Others are using flow analysis tools to study the program for a wide variety of vulnerabilities.

Related to this issue is the development of languages that are safer with respect to security. For example, some languages automatically create an exception if a reference is made beyond the bounds of an array. How much overhead does this add? Can the language use special-purpose hardware to minimize the impact of checking the references? What else should a language constrain, and how should it do so?

31.10 Further Reading

Robust programming—the art of writing programs that work correctly and handle errors gracefully—is a topic of great interest, often in the guise of "secure programming." Kernighan and Plauger's book [1039] describes the principles and ideas underlying good programming style. Kernighan and Pike [1040] also discuss style and other elements of good programming. Stavely's book [1819] combines formalisms with informal steps. Maguire's book [1234] is much more informal, and is a collection of tips on how to write robust programs. Martin [1257] focuses on robust practices for agile programming, while McConnell [1277] discusses robust programming in the general context of software construction.

Howard and LeBlanc [926] discuss secure coding, emphasizing the Windows and .NET environment. Howard, LeBlanc, and Viega's book [927] describes 24 serious but common software flaws and how programmers can avoid them.

Much focus is on the C and C++ programming languages, because of their wide use, lack of type-safe features, and ability to manipulate memory directly. Seacord [1704] and Viega and Messier [1935] discuss ways to make programs in these languages more robust and secure. Sutter and Alexandrescu [1843] present a set of coding standards for C++. Similarly, developing robust, secure web applications is critical, and several books [119, 1241, 1393, 1734] discuss how to do so.

Graff and van Wyk [804] provide a general overview of principles and practice, and much sound advice. Viega and McGraw's book [1932] is also general, with many examples focusing on UNIX and Linux systems. Its design principles give good advice. McGraw [1287] expands on these in a later book. Garfinkel, Schwartz, and Spafford [747] has a wonderful chapter on trust, which is must reading for anyone interested in security-related programming. Wheeler [2000] also provides valuable information and insight.

31.11 Exercises

1. Consider the two interpretations of a time field that specifies "1 a.m." One interpretation says that this means exactly 1:00 a.m. and no other time. The other says that this means any time during the 1 a.m. hour.
   a. How would you express the time of "exactly 1 a.m." in the second interpretation?
   b. How would you express "any time during the 1 a.m. hour" in the first interpretation?
   c. Which is more powerful? If they are equally powerful, which do you think is least astonishing? Why?

2. Verify that the modified version of Requirement 31.4 shown as Requirement 31.6 on page 1105 counters the appropriate threats.
3. Assume the alternative interpretation of Requirement 31.4 given in Section 31.3.1.2, so that access only is controlled by location and time, and that commands are restricted by role and user. This means that if a user is authorized to run a command, she can run it from any location he is authorized to use. How would you change the way information is stored in the access control file described in Section 31.3.2.2?
4. Currently, the program described in this chapter is to have setuid-to-*root* privileges. Someone observed that it could be equally well-implemented as a server, in which case the program would authenticate the user, connect to the server, send the command and role, and then let the server execute the command.
    a. What are the advantages of using the server approach rather than the single program approach?
    b. If the server responds only to clients on the local machine, using interprocess communication mechanisms on the local system, which approach would you use? Why?
    c. If the server were listening for commands from the network, would that change your answer to the previous question? Why or why not?
    d. If the client sent the password to the server, and the server authenticated, would your answers to any of the three previous parts change? Why or why not?
5. The little languages presented in Section 31.3.2.3 have ambiguous semantics. For example, in the location language, does "not host1 or host2" mean "neither at host1 nor at host2" or "at host2 or not at host1"?
    a. Rewrite the BNF of the location language to make the semantics reflect the second meaning (i.e., the precedence of "not" is lower than that of "or"). Are the semantics unambiguous now? Why or why not?
    b. Rewrite the BNF of the time language to make the semantics reflect the second meaning (i.e., the precedence of "not" is higher than that of "or"). Are the semantics unambiguous now? Why or why not?
6. Suppose an access control record is malformed (for example, it has a syntax error). Show that the access control module would deny access.
7. The canary for StackGuard simply detects overflow that might change the return address. This exercise asks you to extend the notion of a canary to detection of buffer overflow.
    a. Assume that the canary is placed directly after the array, and that after every array, access is checked to see if it has changed. Would this detect a buffer overflow? If so, why do you think this is not suitable for use in

practice? If not, describe an attack that could change a number beyond the buffer without affecting the canary.

b. Now suppose that the canary was placed directly after the buffer but—like the canary for StackGuard—was only checked just before a function return. How effective do you think this method would be?

Index

Symbols

- |-* symbol, Take-Grant Protection Model, 33
- |- symbol, Take-Grant Protection Model, 33
- †-property, Basic Security Theorem controversy, 164–166
- *-property, Bell-LaPadula
  - Basic Security Theorem and, 143, 145, 152–155
  - Basic Security Theorem controversy, 164–166, 167
  - and Chinese Wall Models, 235
  - instantiation, 147
  - limits of capabilities, 522–523
  - Lipner's integrity matrix model, 180
  - Multics system, 159–161
  - rules of transformation, 155–157

Numbers

- 2-Step Verification protocol, Google, 446–447
- 64-bits. *See* Data Encryption Standard (DES)
- 128-bits. *See* Advanced Encryption Standard (AES)

A

- *A posteriori* design
  - auditing to detect known violations of policy, 895–897
  - auditing to detect violations of known policy, 893–895
  - *a priori* design vs., 900
- *A posteriori* testing
  - penetration testing as form of, 773, 844
  - techniques in, 16
- AAFID. *See* Autonomous Agents for Intrusion Detection (AAFID)
- Absolute path names, improper naming, 1130
- Abstract data type managers, capability systems, 522
- Abstract machines, HDM, 705–707
- Abstraction
  - application log advantages, 891–893
  - improper isolation of implementation detail, 1123–1124
  - level of weakness, CWE, 868
  - library OSs and, 586
  - representing attacks, 960–964
- Academic computer security policy example
  - electronic communications policy, 127–129
  - full description of. *See* Electronic communications policy, UCD
  - implementation at UC Davis, 130–131
  - overview of, 126–127
  - user advisories, 129–130
- Acceptable use policy, UCD
  - incorporating into allowable use policy, 1241–1246
  - overview of, 130–131, 1207–1212
- Acceptable vs. legal practices, 19–20
- Access control
  - affecting function of server, 579–582
  - break-the-glass policies overriding, 249–250
  - Clinical Information Systems security policy on, 237–238
  - DMZ WWW server vs. development system, 1047
  - electronic communications policy at UCD, 1218
  - file permissions, 1120–1121
  - improper choice of operand/operation and, 1140
  - improper deallocation/deletion of information and, 1132
  - obtaining record of, 1115–1116
  - preserving confidentiality via, 4
  - schemes, 95–97
  - shared password problem in, 1100–1101
  - types of, 117–118
  - unauthorized access to role accounts, 1102
  - using identity for, 472
  - validating entries, 1124
  - via capabilities. *See* Capabilities
  - wrappers, 977, 1046
- Access control lists (ACLs)
  - abbreviations of, 508–511
  - capabilities vs., 519, 523–524
  - Cisco dynamic, 527–528
  - creating/maintaining, 511–514
  - on DMZ WWW server, 1056
  - NTFS and, 515–517
  - outer firewall configuration, 1014–1015
  - overview of, 507–508
  - PACL vs., 532
  - revocation of rights and, 514–515
- Access control matrix
  - architectural security, 651–657
  - copy right and, 42
  - determining system safety, 52–56
  - formal model of Bell-LaPadula Model, 151, 153
  - model. *See* Access control matrix model
  - own right and, 42–43
  - Principle of Attenuation of Privilege, 43–44
  - protection state, 31–32
  - protection state transitions, 37–41
  - review, 44–47
  - security policies changing, 268–270
  - unwinding theorem, 266–268
- Access control matrix model
  - ATAM vs. TAM, 99–101
  - Boolean expression evaluation in, 35–36
  - comparing HRU, SPM and, 82
  - history and, 36–37
  - malleability of, 61
  - as protection system, 32–34
  - TAM model as expansion of, 92–94

Access control mechanisms
  ACLs. *See* access control lists (ACLs)
  capabilities, 518–526
  locks and keys, 526–531
  overview of, 507
  Propagated Access Control List (PACL), 533–534
  review, 535–537
  ring-based access control, 531–533
Access control module
  defined, 1104
  design, framework, 1104–1105
  design, roles and commands, 1106–1110
  first-level refinement, 1111–1112
  functions, 1114–1117
  review, 1117
  second-level refinement, 1112–1114
Access points, wireless networks, 1023
Access Restriction Facility (ARF) program, 35–36
Access, user security
  leaving system unattended, 1079
  login procedure, 1076–1079
  passwords, 1074–1076
Access without consent, electronic communications policy, 129, 1221–1222, 1232–1233
Accessibility
  electronic communications policy at UCD f, 1219
  password wallet disadvantages, 425
  using cloud remotely for, 1024
Accountability
  auditing for, 174, 472
  identities for, 472
Accuracy (classification rate), intrusion detection methods, 925
ACK packet
  availability, and SYN flooding, 215
  in pulsing DoS attack, 221–222
  SYN flooding countermeasures, 217
ACLs. *See* Access control lists (ACLs)
Actions
  considering effects of, 989
  DIDS, 950–951
Activation transition, resource allocation system, 211–212
Active side channel attacks, 280
Active wiretapping, 7
Acyclic attenuating schemes, 77–81, 87–88
Acyclic creates rule, 73–74
Adaptive directors, altering rules, 946
Adaptive intrusion detection models, 920
Adaptive timeouts, availability in flooding attacks, 220
Address space layout randomization (ASLR), 974–975
`AddRoundKey` transformation, AES, 304, 1199, 1203–1205
Adjacent pairs of specification, 677–680
Adleman, Len, 781
Administrative accounts
  shared password problem, 1100–1103
  user configuration for development system, 1050–1053
Administrative assurance, defined, 634
Adore-ng rootkit, 778
Adorned names, listing all SLDs in MLD, 148–149
Advanced Encryption Standard (AES)
  analysis of, 304–305
  background, 1196–1197
  basic transformations, 1197–1199
  as block cipher, 370
  ciphers that were finalists to, 303
  decryption, 1200–1201
  encryption, 1199
  equivalent inverse cipher, 1203–1205
  modes, 305
  overview of, 303, 1196
  replacing DES, 302
  review, 1205
  round key generation, 1201–1203
  strong mixing function of, 342
  structure, 303–304
Adware, 797–799
AEAD. *See* Authenticated encryption with associated data (AEAD)
Aegis kernel, isolation via library OS, 585
AES. *See* Advanced Encryption Standard (AES)
Agents
  AAFID autonomous, 952–953
  intrusion detection architecture, 942–945
  NSM network, 950
Aggregation principle, 238
Aggressive Chinese Wall Model, 233–234
Agile software development, 641–644
Aging, password, 434–438
AH. *See* authentication header (AH)
AI. *See* artificial intelligence (AI)
Aldus FreeHand, MacMag Peace virus and, 782
Alert protocol, TLS, 399
Algorithms
  determining system safety, 51–56
  generating random numbers, 341–342
  main DES, 1191–1194
  system security questions, 49–51
Allowable use, electronic communications policy
  overview of, 127–128, 1216–1220
  updated version, 1241–1246
  user advisories, 1235
Alteration, and threats, 7
AM security level. *See* Audit Manager (AM) security level
Amplification
  attacks, 221–222
  as increasing privileges, 521
Anagramming, attacking transposition cipher, 292
Analysis engine
  DIDS centralized, 950
  intrusion detection, 945–946
Analysis phase, digital forensics, 993–994
Analysis procedure, Protection Analysis (PA) model, 854–856
Analyzer, auditing system, 883
And-access, cryptographic locks and keys, 527
And, joining conditions with, 41
Anderson's Formula, attacking passwords, 426–427
Android cell phones
  Geinimi Trojan horse and, 776–777
  loading libraries for process confinement, 593–594
  privacy information flow issue, 568–570
Animal game, 779, 984
Annotated programs, formally verified products, 722–723
Anomaly detection
  clustering, 926–928
  defined, 920
  distance to neighbor, 930–931
  incident prevention via, 972
  intrusion detection, 972

  machine learning, 924–925
  Markov models, 922–924
  misuse detection vs., 941–942
  neural nets, 928–929
  other methods, 932
  overview of, 920–921
  self-organizing maps, 928–930
  statistical methods, 921–922
  support vector machine (SVM), 931–932
  threshold metrics, 921
Anon.penet.fi, Swedish anonymizer, 491
Anonymity
  electronic communications policy, 1218
  erosion of privacy/need for, 482
  on web. *See* Web, anonymity on
Anonymizers
  email, 491–494
  hiding origins of connections, 490–491
Anonymizing sanitizers, auditing, 889–891
Anonymous Diffie-Hellman, 394
Anonymous (persona) certificates, 482
Anti-forensics, 994–996
Anti-replay
  AH protocol, 408
  IPsec architecture, 405
Antivirus scanners, 808–809
APA tool. *See* Automated penetration analysis (APA) tool
AppAudit, 570
Append right, access control matrix, 33
Appendices in this book
  academic security policy. *See* Academic computer security policy example
  electronic communications policy at UCD, 1227–1233
  encryption standards. *See* Encryption standards
  entropy and uncertainty, 1163–1169
  Extended Euclidean Algorithm, 1157–1161
  lattices, 1153–1155
  logic. *See* Symbolic logic
  overview, 1151
  virtual machines, 1171–1177
Apple iPhones, Pegasus spyware for, 799–800
Appleseed trust model, 198
Application data protocol, TLS, 400
Application level firewalls. *See* Proxy (or application level) firewalls
Application logs, auditing design, 891–893
Arc attacks, 848, 974–975
Architecture
  building systems with assurance, 651–657
  of capabilities vs. ACLs, 519
  intrusion detection, 942–948
  IPsec, 404–407
  TCSEC, 733
  waterfall life cycle model, 640
ARF, *See* Access Restriction Facility (ARF) program
ARHIVEUS-A ransomware, 801
Arrays, bounds checking and, 1133–1134
Artificial intelligence (AI), exploratory programming model, 644
ASCII characters
  cryptographic checksums and, 315–316
  PEM design and, 387–388
  UNIX passwords and, 417
Aslam's model, 859–860, 862–864
ASLR, *See* Address space layout randomization (ASLR)
Assertions, policy-based trust models, 192
Assignment statements, information flow, 551
Assumptions
  conception stage of life cycle, 636
  logging in forensics not controlled by, 988–989
  trust and, 11–12, 115–117
Assurance. *See* also Systems, building with assurance
  design and, 14–15
  evaluation of evidence for. *See* Evaluation of systems
  example of, 22–24
  formal evaluation methodology for, 728
  formal methods. *See* Formal methods
  implementation. *See* Implementation assurance
  improper choice of operand/operation and, 1140
  low-assurance programs. *See* Program security practicum
  network organization and, 1025–1026
  security policies and, 110
  specifications and, 14
  and trust, 12–13
Assurance, introduction to
  Agile software development, 641–644
  life cycle, 634–639
  need for, 629–631
  other models of software development, 644–645
  review, 645–648
  role of requirements in, 631–632
  throughout life cycle, 632–634
  and trust, 627–629
  waterfall life cycle model, 639–641
Assurance requirements
  CISR, 743
  Common Criteria, 752, 759
  Federal Criteria, 745
  ITSEC, 739
  TCSEC, 732–733
Astonishment. *See* Principle of least astonishment
ATAM. *See* Augmented Typed Access Matrix Model (ATAM)
Attack and response
  anti-forensics, 994–996
  attack definitions, 959–960
  attack graphs, 969–971
  attack trees, 961–964
  digital forensics. *See* Digital forensics
  intrusion response. *See* Intrusion response
  representing attacks, 960–961
  requires/provides model, 965–969
  review, 996–1001
Attack graphs, 969–971
Attack phase, GISTA, 836
Attack trees
  developing, 961–964
  requires/provides model, 965–969
  as subset of attack graphs, 969–971
Attacker, remote shell (*rsh*) attack, 966–967
Attacks
  anticipating, 1027–1028
  on cryptosystem, 290–291
  defined, 987
  on DNS, 487
  further reading, 25
  password, 426–434
  protecting system after. *See* Intrusion response

  risk analysis of probability of, 17–18
  on systems failing to meet principles, 917–918
  threats vs., 7
  transposition cipher, 292
  Vigenére cipher, 294–299
Attenuating create-rule, 74
Attenuation of privilege. *See* Principle of attenuation of privilege
Audit browsing, 908–910
Audit logger, LAFS, 906
Audit logs, IDIOT monitoring, 934
Audit Manager (AM) security level, Lipner, 178–180, 182
Audit UID, 474
Auditing
  commercial requirements, 174
  definition of, 879–880
  electronic communications policy, 1227
  file systems, 900–907
  firewalls for, 573
  intrusion detection as automated, 942
  mechanisms for, 897–900
  review, 910–915
  security violations, 879
  TCSEC functional requirements, 732
Auditing, designing system
  anatomy of, 881–884
  application/system logging, 891–893
  implementation/, 886–887
  log sanitization, 888–891
  overview of, 884–886
  a posteriori design, 893–897
  syntactic issues, 887–888
Augmented Typed Access Matrix Model (ATAM), 98–99
Aurasium, constraints for Android apps, 593–594
Authenticated encryption with associated data (AEAD), 377–381, 392
Authentication
  accessing to attack passwords, 426–434
  basics of, 415–416
  biometrics as, 441–445
  challenge-response, 438–441
  defined, 415
  dynamic vs. static naming and, 486–487
  in electronic communications policy, 1226, 1240
  graphical passwords, 425–426
  improper deallocation or deletion of information, 1131–1132
  Kerberos protocol, 337–338
  location and, 445–446
  multifactor, 446–448
  one-time passwords, 436–438
  origin integrity as, 5
  password aging, 434–438
  password selection, 418–425
  passwords, 416–418
  review, 448–452
  symmetric key exchange and, 333–336
  system security practicum, 1053–1055
  using cookies for, 490
Authentication header (AH), IPsec, 403–408
Authentication path, Merkle's tree authentication, 345
Authentication policy, CA, 477
Authenticity of digital image, anti-forensics hindering, 994–996
Authority
  to change, electronic communications policy, 1233–1234
  key identifier extension, PKI certificates, 351
  principle of least, 458
Authorization
  electronic communications policy, 1226
  integrity policies and, 111
  policy specifications in Ponder, 119–121
Authorization scheme, TAM, 93
Authorized (secure) states, security policy, 109–113
Authorized transfer of rights, system safety, 50
Authorizing (certificate producing) participants, SOG-IS, 762
Autokey cipher, 373–374
Automated penetration analysis (APA) tool, Gupta and Gligor, 872–873
Automation
  classifying verification technologies via, 700
  intrusion detection process, 918
  legacy of Protection Analysis, 856
  polymorphic viruses for instruction, 788
  of security test suites, 689
  sophisticated system attacks, 918
  specification languages, 703
Autonomous Agents for Intrusion Detection (AAFID), 952–953
Availability
  as basic to computer security, 6
  configurations for system security, 1044–1045
  countering threats with, 7
  DoS attacks attempting to block, 6
  ensuring program, 1147
  nature of security policies, 111
  policy development practicum, 1010
Availability policies
  denial of service models, 203–204
  denial of service models, constraint-based, 204–210
  denial of service models, state-based, 210–215
  goals of, 201–202
  handling deadlock, 202–203
  network flooding example, 215–221
  other flooding attacks, 221–222
  review, 222–225
Avoidance, deadlock, 203

B

BABEL. *See* Mixmaster remailer
Backdoor.IRC.Aladinz bot, 794
Backoff techniques, on-line dictionary attacks, 430–431
Backups
  to cloud, 1025
  development system users, 1051–1052
  electronic communications policy for, 1227, 1240–1241
Bacterium, 796, 803
Bandwidth
  as property of covert channels, 595–596
  SYN flooding consuming, 216, 1026
Banker's Algorithm, deadlock avoidance, 203
Banners, adware installation via, 798
Basic blocks, 554–556
Basic constraints extension, X.509 PKI certificates, 351–352

Basic Security Module (BSM)
    nonsecure systems and, 899–900
    using grammar, 888
    Visual Audit Browser tool kit, 909–910
Basic Security Theorem
    formal Bell-LaPadula Model, 152–155
    McLean's †-property, 164–166
    McLean's System Z, 166–168
    preliminary version, 143, 145
Bayes signatures, worm detection, 810
Behavior, reputation-based trust models, 194–196
Behavioral analysis, malware detection, 810–811
Belief types, trust in technological world, 190–191
Bell-LaPadula Model
    Android two-level security model, 568
    Biba's strict integrity model, 177–178
    composition of two models, 256–258
    configuring outer firewall, 1014–1015
    controversy over, 164–168
    declassification principles, 163–164
    designing auditing for, 884–885
    emulating Chinese Wall Model, 234–236
    formal model, 151–158
    formal specifications, 702–705
    influencing TCSEC approach, 730
    informal description of, 142–146
    lattice-based information policy, 539–540
    limits of capabilities, 522–523
    Lipner's use of, 178–180
    military-style classifications of, 141
    MLS implementing SRI model of, 707
    nonlattice information flow, 542–543
    restricting flow of information, 183
    review, 169–172
    role of tranquility in, 164
    separating policy from mechanism, 256
    as subset of Clinical Information Systems Security Policy, 239
    Trusted Solaris example, 146–151
Bell V22 Osprey helicopter crashes, 631
Bellare-Rogaway protocol, symmetric key exchange, 336
Berkeley packet filter (BPF), malware defense, 818–819
Bernstein conditions, 34
Best matching unit (BMU), self-organizing maps, 929–930
Biba model
    Clark-Wilson model vs., 188–189
    lattice-based information policy in, 539
    Lipner's integrity matrix model vs., 182–183
    overview of, 175–178
    scanning as malware defense, 808
Biconditional commands, protection state transitions, 41
Biconditional monotonic protection systems, 55–56
Biometrics
    authentication and, 441–442
    combinations, 445
    eyes, 443–444
    faces, 444
    fingerprints, 442–443
    generating cryptographic keys, 342–343
    keystroke dynamics, 444–445
    voices, 443
Bionic libc, Aurasium, 594
Bit-oriented ciphers
    AES. *See* Advanced Encryption Standard (AES)
    DES, 300–302
    one-time pad, 371
Bitcoins, CryptoLocker ransomware, 801
Black box (functional) testing, 688–689
Bledsoe theorem prover, Gypsy, 712–713
Block ciphers
    CCM mode using AEAD for, 377–379
    multiple encryption, 375–377
    overview of, 374–375
    stream ciphers vs., 370
Blocks
    misordered ciphtertext message and, 368
    TLS record protocol, 396
Blowfish, modern symmetric cipher, 303
BMU. *See* best matching unit (BMU)
Boolean expressions, access control by evaluating, 35–36
BOOLEAN type, SPECIAL specification, 703
Boot sector infectors, 782–783, 786
Botmaster, 793
Bots and botnets, 793–796
Boundary controller, IDIP, 978
Boundary data tests, 1143–1144
Bounding set of privileges, processes, 524
Bounds checking, improper validation, 1133–1134
Boyer-Moore theorem prover, 707, 709–710
BPF. *See* Berkeley packet filter (BPF)
Brain (or Pakistani) virus, IBM PC, 782, 783
Branch instruction, Data Mark Machine, 564
Branching time logic systems, 1186
Breach, security, 110, 112
Break-the-glass policies, 249–250
Bridge relays, Tor, 499
Bridges, Take-Grant Protection Model, 60–61
Bro, misuse intrusion detection, 937–938
Browser plug-ins, allowing adware, 798
Browsing, audit, 908–910
BSM. *See* Basic Security Module (BSM)
Buffer overflow attacks
    memory protection and, 1122
    restricting access via type checking, 528, 847–848
Bugs, maintenance releases/hot fixes for, 695–696
Burroughs B5700 system, penetration study, 839–40
Businesses, key escrow system for, 354–355
"By law" or "by right" (*de jure*) rules, 57–61

C

C (CONFIDENTIAL) security clearance, Bell-LaPadula Model, 142–146
C-List. *See* capability list (C-List)
CA certificate, X.509 PKI, 350–351
Cache Kernel, isolation via library OS, 585–586
Cache poisoning attacks, 487, 488
Caching of information, restricting, 460–461
Caesar (shift) cipher, 289–291, 294
Call bracket, ring-based access control, 531–532
Can-create function, SPM, 73–74, 82–85
Canadian Trusted Computer Product Evaluation Criteria (CTCPEC), 737–738
Capabilities
    access control lists vs., 523–524
    copying and amplifying, 520–521

  in JIGSAW language, 967–969
  limits of, 522–523
  mechanisms protecting, 519–520
  overview of, 518–519
  privileges, 524–526
  requires/provides model, 965–966
  revocation of rights, 522
Capability list (C-List), 518, 522–523
Capability Maturity Levels, SSE-CMM, 767
Capability mode, Capsicum, 589
Capacitative technique, fingerprint biometrics, 442
Capacity, covert channel, 611–616
Capsicum
  file descriptor capabilities in, 526
  sandboxing single application via, 589–590
CAPSL. *See* Common Authentication Protocol Specification Language (CAPSL)
CAPTCHAs, thwarting on-line dictionary attacks, 431
CAs, *See* Certificate authorities (CAs)
Categories
  commercial vs. military environments, 174
  easy to guess passwords, 421–422
  Lipner's integrity matrix model, 179–182
Categories, Bell La-Padula Model
  adding to security classification, 143–146
  Chinese Wall Models and, 234–236
  formal model of, 151–158
  principle of tranquility, 161–163
  Trusted Solaris, 146–151
Category entry, CWE, 867
CAVP. *See* Cryptographic Algorithm Validation Program (CAVP)
CBC mode. *See* Cipher block chaining (CBC) mode
CC. *See* Common Criteria (CC)
CC Evaluation Methodology (CEM), 750–751, 761, 764–765
CCDB. *See* Common Criteria Development Board (CCDB)
CCEVS. *See* Common Criteria Evaluation and Validation Scheme (CCEVS)
CCM. *See* Counter with CBC-MAC (CCM) mode
CCMB. *See* Common Criteria Management Board (CCMB)
CCRA. See Common Criteria Recognition Arrangement (CCRA)
CCUF. *See* Common Criteria Users Forum (CCUF)
CCured program, compiling, 592
CDIs. *See* Constrained data items (CDIs)
CDs. *See* Company datasets
Cells, Tor onion router, 497–499
CEM, *See* CC Evaluation Methodology (CEM)
Centralized botnets, 793
Centralized security enforcement, architecture, 651–652
Cert_chain, TLS handshake protocol, 397–398
Certificate authorities (CAs)
  assurance of trust, 481–484
  authentication policy, 477
  certificate conflicts, 479–481
  controlling issuing of certificates, 476–478
  cross-certified, 347–348
  defined, 347
  extensions supported by, 350–351
  issuance policy, 477
  meaning of identity, 481–484
  X.509 certificate signature chains, 347–348
  X.509 PKI certificates, 351
Certificate policy extension, X.509 PKI, 351–352
Certificate producing (authorizing) participants, SOG-IS, 762
Certificate revocation list, X.509 PKI, 359
Certificate signature chains, 346–350
Certificates
  binding cryptographic keys to identifiers, 476
  binding identity to cryptographic key, 344
  conflicts, 479–481
  expired vs. revoked, 358–359
  Merkle's tree authentication scheme, 344–345
  naming and. *See* Naming and certificates
  PKI, 350–353
  policy-based trust models, 191
  TLS handshake protocol, 398
Certification
  Clark-Wilson integrity model rules, 184–186
  ITSEC, 738
Certified licensed evaluation facilities (CLEFs), ITSEC, 741
Cert_type, TLS handshake protocol, 398
CFB. *See* Cipher feedback (CFB) mode
Chain entry, CWE, 867
Chain key, instant messaging, 390–392
Challenge-response authentication, 438–441
Challenger space shuttle disaster, 1986, 630
Change authorization, 686–687
Change cipher spec protocol, TLS, 399
Change, improper program, 1125–1129
Channels, OSSTMM, 834, 835
Character frequencies, table of, 293
Characters, and monitors, 1086
Checking input, improper validation, 1136–1137
Checksums
  cryptographic, 315–318
  key escrow system, and Clipper chip, 356
  malware defense using, 817
  Merkle's tree authentication scheme, 344–345
  TLS setup phase, 394
Chinese Wall Model
  aggressive, 233–234
  auditing design for, 885–886
  Bell-LaPadula and, 234–236
  Clark-Wilson and, 236
  formal model, 230–233
  informal description, 228–230
  overview of, 227
CHMK instruction, privilege/virtual machines, 1172–1173
Chosen plaintext attack, 291
Chroot system, UNIX, 1056–1057
CIAC. *See* Computer Incident Advisory Capability (CIAC)
CIF. *See* Common Internal Form (CIF)
Cipher block chaining (CBC) mode, 302, 375
Cipher feedback (CFB) mode, 302, 374
Cipher techniques
  authenticated encryption, 377–381
  block ciphers, 370, 374–377
  example protocols, 384
  instant messaging, 389–393
  network layer security (IPsec), 402–410
  networks and cryptography, 381–384
  overview of, 367
  problems, 367–370
  review, 410–414
  secure electronic mail, 384–389

  stream ciphers, 370–374
  transport layer security. *See* Transport layer (TLS and SSL) security
Cipher_list, TLS handshake protocol, 397
Ciphertext
  perfect secrecy and, 1168–1169
  security problems with messages, 367–370
  self-synchronous stream ciphers, 373–374
  substitution ciphers, 292–294
  transposition ciphers, 291
Ciphertext only attack, 290
Circular wait, deadlocks from, 202
Cisco routers, dynamic access control lists, 527–528
CISR. *See* Commercial International Security Requirements (CISR)
Clark-Wilson integrity model
  Chinese Wall Models and, 236
  Clinical Information Systems Security Policy, 239
  comparing to other models, 188
  comparing to requirements, 187–188
  implementing under UNIX, 186–187
  the model, 184–186
  overview of, 183–184
Classes
  CC assurance requirements, 759
  CC security functional requirements, 756–759
  OSSTMM, 834–835
  policy development with data, 1007–1008
  policy development with user, 1008–1010
  TCSEC evaluation, 733–734, 736
  of threats, 7
Classification
  Aslam's model, 859–860
  confidentiality, 142–146
  flaws in Protection Analysis model, 849–851
  flaws in RISOS study, 849–851
  Gupta and Gligor's penetration analysis theory, 868–873
  Lipner's integrity, 181–182
  NRL taxonomy for vulnerabilities, 857–859
  principle of tranquility, 161–163
  Trusted Solaris, 146–151
  verification technologies, 700
  vulnerability frameworks for, 845–848
Classification rate (accuracy), intrusion detection methods, 925
Clearance, Trusted Solaris, 146–151
CLEFs. *See* certified licensed evaluation facilities (CLEFs)
Client, confinement problem, 579–582
Clinical Information Systems security policy, 236–239
Clipper Chip, and key escrow system, 355–357
Clock synchronization, Kerberos, 338
Cloud
  defined, 1024
  network security practicum for, 1024–1025
Clustering, anomaly detection and, 926–928
CMVP. *See* Cryptographic Module Validation Program (CMVP)
CNSS. *See* Committee on National Security Systems (CNSS)
Cocks, Clifford, 309
Code Red I computer worm, 791
Code Red II computer worm, 792
Code review (walkthroughs), implementation, 687–688
Code standards, implementation, 686
Codebook mode, statistical regularities, 369
The Codebreakers (Kahn), 325
Coding faults, Aslam's model, 859
Cohen, Fred, 781
COI classes. *See* Conflict of interest (COI) classes
Collaboration mission, CSIRT, 986
Collaborative Protection Profiles (cPP), 751
Collisions, off-line dictionary attacks, 429
Colored Petri Automaton (CPA), IDIOT system, 933–934
Combinations
  of biometrics, 445
  of malware, 803
Combining sources of information, agents, 943–944
Command and control (C&C) servers/motherships, 793–795
Command line, program security design, 1104–1105
Commands
  comparing HRU and SPM, 82
  designing access to roles and, 1106–1110
  protection state transitions, 38–41
  refinement to access control module, 1112–1114
  Tor, 497–499
Comment resolution, in review process, 684
Commercial integrity policies, 173–174
Commercial International Security Requirements (CISR), 742–744
Commercial off-the-shelf (COTS) components, 696–697, 730
Commercial security policies, 114
Committee on National Security Systems (CNSS), 729–730
Common Authentication Protocol Specification Language (CAPSL), 720–721
Common Criteria (CC)
  assurance requirements, 759
  defined, 727
  evaluation assurance levels, 759–761
  evaluation process, 761–762
  functional requirements, 756–759
  future of, 764–765
  impacts, 763–764
  informal arguments, 680–681
  methodology, 751–756
  overview of, 749–751
  requirements, 756
  SOG-IS, 762–763
  standards replaced by, 629
Common Criteria Development Board (CCDB), 764–765
Common Criteria Evaluation and Validation Scheme (CCEVS), 750
Common Criteria Management Board (CCMB), 764–765
Common Criteria Recognition Arrangement (CCRA), 749–751, 764–765
Common Criteria Users Forum (CCUF), 763
Common Internal Form (CIF), HDM, 706–707
Common Vulnerabilities and Exposures (CVE) database, 864–866
Common Weaknesses and Exposures (CWE) database, 866–868
Communications path, TCSEC, 732
Company datasets (CDs)
  Aggressive Chinese Wall Model, 233–234
  Bell-LaPadula and Chinese Wall Models, 234–236
  Chinese Wall Model audit design, 885–886
  Chinese Wall Model formal model, 230–233
  Chinese Wall Model informal description, 228–230
Compartmented Mode Workstation, auditing, 898

Compatibility, PEM design principles, 386
Compiling, process confinement via, 592–593
Complete mediation. *See* Principle of complete mediation
Completion, review process, 685
Complexity of programs, implementation phase, 15
Components
  auditing system, 881–884
  defined, 663
  example, 663–664
  Extended Components Definition, CC, 752
  external functional specification, 666
  internal design description, 669–672
Composed modules, testing, 1145
Composition
  deducibly secure systems, 273–274
  deterministic noninterference-secure systems, 270–271
  generalized noninterference systems, 275–277
  policy. *See* Noninterference, and policy composition
  of restrictive systems, 279
Compositional security analysis instance, 96–97
Compound element composite entry, CWE, 867
Compound sentences, composing, 1179
Compound statements, information flow, 551–553, 559–560
Compression_list, TLS handshake protocol, 397
Compromise Remote Users/Sites, ISSAF, 833–834
Computer forensics. *See* Digital forensics
Computer Incident Advisory Capability (CIAC), ransomware and, 800–801
Computer security incident response team (CSIRT), 985–987
Computer security, overview
  assumptions and trust, 11–12
  assurance, 12–16
  basic components, 3–6
  example, 22–24
  human issues, 20–22
  operational issues, 16–20
  policy and mechanism, 9–11
  review, 24–28
  threats, 6–9
Computer-supported collaborative working confidentiality policies, 170
Computer viruses
  concealment, 785–790
  defined, 780
  infection vectors, 782–785
  overview of, 780–782
  summary, 790
  theory of, 803–807
COMSEC class, OSSTMM, 834
Concealment, of computer viruses, 785–790
Conception stage, life cycle process, 635–636
Concurrency, 209–210, 558–561
Condition set consistency prover, automated penetration analysis, 873
Condition validation errors, Aslam's model, 859
Conditional and joint probability, 1163–1165
Conditional commands, TAM, 93
Conditional entropy, 1167–1168
Conditional instruction, Data Mark Machine, 563
Conditional statements, information flow, 552–553
Conditional transitivity of trust, 189–190
Conficker botnet, neutralizing, 570–571
Confidence, security assurance as, 628
CONFIDENTIAL (C) security clearance, Bell-LaPadula Model, 142–146
Confidentiality
  as basic to computer security, 4–5
  Chinese Wall Model and, 227
  Clinical Information Systems policy, 236–239
  consistency check, policy development, 1010–1011
  constraints to control flow of information, 566–567
  countering threats, 7
  data moving from internal network to Internet, 1011–1012
  electronic communications policy, 128, 1220–1225
  military security policy, 113–114
  nature of security policies, 110–111
  security threats as breaches of, 650
  TCSEC emphasis on, 730–731
Confidentiality policies
  Bell-LaPadula Model. *See* Bell-LaPadula Model
  Bell-LaPadula Model, controversy, 164–168
  defined, 111, 115
  developing via mapping, 658–660
  goals of, 141–142
  impact on logs, 888–889
  information flow policy within, 539–540
  Multics system example, 158–161
  principle of tranquility, 161–164
  review, 169–172
  security/precision in, 131–134
  trust and, 114
Configuration assistant, LAFS, 906
Configuration errors, Aslam's model, 859
Configuration management, 686–687, 732
Configuration of system, deployment stage of life cycle, 638
Confinement
  analyzing suspected malware, 810
  flow model, 543–544
  internal address issues, 1014
  problem of, 579–582
  review, 619–623
  via covert channels. *See* Covert channels
  via isolation. *See* Isolation
Confinement principle, 238, 239
Conflict of interest (COI) classes
  Aggressive Chinese Wall Model, 233–234
  Bell-LaPadula and Chinese Wall Models, 235–236
  Chinese Wall Model audit design, 885–886
  Chinese Wall Model formal model, 230–233
  Chinese Wall Model informal description, 228–230
Conflict resolution, review process, 684
Conflicts, certificate, 479–481
Conformance claims, 751–752
Conjunction, propositional logic, 1179–1180
Conjunction signatures, worm detection, 810
Connection ID, Network Security Monitor, 948–949
Connections
  anonymizers hiding origins of, 490
  TLS, 393–394, 397
Connectives
  compound sentences, 1179
  natural deduction in propositional logic, 1180–1181
Conservativity principle, declassification policy, 163
Consistency check, policy development, 1010–1011
Consistent state (or consistent), integrity of system data, 183

Consistent static analysis, state-based audits, 894
Conspiracy, Take-Grant Protection Model, 66–68
Constrained data items (CDIs), Wilson integrity model, 184–188
Constraint-based denial of service model
  finite waiting time policy, 207–208
  overview of, 204–205
  service specification, 208–210
  user agreement, 205–207
Constraints
  affecting penetration study, 827–828
  audit analysis of NFSv2, 902, 904–905
  designing auditing system, 884–885
  dynamic analysis for information flow, 568–570
  information flow integrity, 566–567
  low-level policy languages as, 125–126
  policies begin as, 127
  $RBAC_2$, 247–248
Consuming participants, SOG-IS agreement, 762
Containers, providing isolation via, 584–585
Containment, as malware defense
  information flow metrics, 812–813
  reducing rights, 813–816
  sandboxing, 816–817
Containment phase, intrusion handling, 975–977
Content delivery servers, 794–795
Content, digital rights management, 242
Content Scrambling System (CSS), DVDs, 461–462
Context, for meaning in forensics, 989–990
Contradiction, in propositional logic, 1180
Control, architectural security and, 651–657
Control rights (RC), SPM, 69, 71–72
Control Tree Logic (CTL), 716–720, 1186–1188
Controlled access protection, TCSEC, 733
Controlled environment, process isolation
  library operating systems, 585–586
  overview of, 582–583
  sandboxes, 586–590
  virtual machines, 583–585
Controversy, Bell-LaPadula Model, 164–168
Cookies
  right to privacy and, 501
  and state, 488–490
Copy flags
  access control matrix, 4
  associated with capabilities, 520
  Schematic Protection Model tickets, 69
Copy right, 42–43
Copyable tickets, SPM, 70–71
Copying of capabilities, 520–523
Copying strings, 1133
Copyright, 241–244, 1209
Corporate computer system, penetration of, 840–841
Corporate data (CpD) class, policy development, 1007–1009, 1011
Corporation executives class, policy development, 1008–1010
Correctness-preserving transformations, 644–645
Correspondence between schemes, simulation in models, 89–90
Cost-benefit analysis, operations, 16–17
COTS components. *See* commercial off-the-shelf (COTS) components
Counter, amplifying capabilities, 521
Counter (CTR) mode, AES, 305
Counter method
  cipher feedback mode, 374
  synchronous stream ciphers, 373
Counter with CBC-MAC (CCM) mode, AEAD, 377–379
Counterattacking, forms/consequences of, 983–984
Countermeasures, SYN flooding, 216
Courtesy, electronic communications policy, 1235
Covert channels
  adware installed via, 797–799
  analysis of noisy capacity, 614–616
  capacity/noninterference and, 611–613
  confinement problem and, 594–596
  defined, 580
  infinite loops and, 558
  measuring capacity, 613
  mitigation of, 616–619
  review, 619–623
  side channels vs., 581
  spyware installed via, 799–800
  types of, 581–582
  virtual machines for, 584
Covert channels, detecting
  via covert flow trees, 602–610
  via information flow analysis, 601–602
  via noninteference, 596–598
  via shared resource matrix, 598–600
Covert flow trees
  completed, 607, 609
  constructing two lists, 607, 610
  nodes of, 603
  overview of, 602
  stages of building, 605–608
  using at any point in SDLC, 610
Covert purposes, of Trojan horses, 776, 781
Covert storage channels
  analyzing, 598–599
  for covert flow trees, 605–606
  defined, 594
Covert timing channels
  analyzing, 600
  mitigating covert channels, 617–618
  overview of, 594
CP commands, IBM VM/370, 1175–1176
CPA. *See* Colored Petri Automaton (CPA)
CpD class. *See* corporate data (CpD) class
CPP. *See* Collaborative Protection Profiles (CPP)
Create-rule, SPM, 73–74
Create rule, Take-Grant Protection Model, 57, 65
Create rules, ESPM, 83–88
Credentials, policy-based trust models, 191–194
Credit card company alerts, 947
Crimea virus, 789
Criteria creep, 736, 764
Cross-certificate, X.509 PKI, 351
Cross-certified CAs, 347–348
Cross-realm operation, Kerberosv5, 338
Cryptanalysis
  defined, 289
  differential, 301
  linear, 301
  overview of, 290–291
Cryptographic Algorithm Validation Program (CAVP), 749
Cryptographic checksums, 315–318

Cryptographic keys. *See* also Key management
  defined, 4
  encrypted viruses do not encrypt, 786–787
  improper deallocation or deletion of, 1131–1132
Cryptographic Module Validation Program (CMVP), 748, 749
Cryptographic modules, FIPS 140-2, 746–749
Cryptography
  cipher techniques. *See* Cipher techniques
  formal methods for analyzing protocols, 702
  key management. *See* Key management
  locks and keys, 527
  NPA protocol verification, 720–721
  offensively used in ransomware, 801
  preserving confidentiality via, 4
  protecting capabilities via, 519–520
  as secret writing, 289
Cryptography, basics
  checksums, 315–318
  cryptanalysis, 290–291
  digital signatures, 318–323
  overview of, 289–290
  public key. *See* Public key cryptography
  review, 324–329
  symmetric cryptosystems. *See* Symmetric cryptosystems
CryptoLocker ransomware, 801
Cryptosystems
  cryptanalysis as analysis of, 290–291
  defined, 289
  perfect secrecy and, 1168–1169
  principle of open design and, 461
  symmetric. *See* Symmetric cryptosystems
  timing attacks on, 280–282
  transformations, 290
CSIRT. *See* computer security incident response team (CSIRT)
CSS. *See* Content Scrambling System (CSS)
CTCPEC. *See* Canadian Trusted Computer Product Evaluation Criteria (CTCPEC)
CTL. *See* Control Tree Logic (CTL)
CTR mode. *See* Counter (CTR) mode
CuD class. *See* customer data (CuD) class
Cued-recall systems, graphical passwords, 426
Current rights, access control by history, 36–37
Customer data (CuD) class, 1007–1009, 1011
Customer service, fielded product life stage, 638
Customs, operational controls and, 19–20
CVE database. *See* Common Vulnerabilities and Exposures (CVE) database
CWE database. *See* Common Weaknesses and Exposures (CVE) database
Cyber-physical systems, information flow in, 575
Cypherpunk remailers, 492–493

D

D (Development) category, Lipner, 179–180
D-WARD, defense against DoDS attacks, 217–218
DAC. *See* Discretionary access control (DAC)
Dalvik executables (DEX) bytecode, Android, 568–569
DARPA off-line intrusion detection evaluations, 925
Data
  checking for valid, 1135–1136
  checking input from untrusted sources, 1136–1137
  computer viruses infecting, 782, 786
  creating structures that can be validated, 1137–1138
  improper change over time, 1125–1129
  macro viruses infecting file, 785
  malware defenses, 811–812
Data classes
  configuring internal network, 1021–1022
  consistency check, 1010–1011
  policy development practicum, 1007–1008
Data descriptions, external functional specification, 666
Data encipherment key (DEK), PEM, 386–387
Data Encryption Standard (DES)
  AES replacing, 302
  analysis of, 301–302
  EDE mode, 376
  main algorithm, 1191–1194
  and modes, 302
  overview of, 299–300, 1191
  retirement of, 302
  review exercises, 1205
  round key generation, 1195
  structure, 300
  three-key Triple DES mode, 377
  two-key Triple DES mode, 376
Data integrity. *See* also Integrity; Integrity policies
  cryptography for, 290
  defined, 5
  DNSSEC providing, 488
  nature of security policies, 110–111
Data Mark Machine, Fenton's, 562–566
Data networks class, OSSTMM, 834
Data recovery component, key escrow systems, 355
Data segments, ring-based access control, 531–532
Database security, access control matrix model, 44
DDEP class. *See* development data for existing products (DDEP) class
DDFP class. *See* development data for future products (DDFP) class
DDoS. *See* distributed denial of service attack (DDoS)
De jure ("by law" or "by right") rules, 57–61
Deactivation transition, resource allocation system, 211–212
Deadlock, availability and, 202–203
Deallocation, improper deletion of, 1131–1132
Deception, as class of threat, 7–8
Deception Tool Kit (DTK), 976
Declarations
  information flow and, 549–550
  PVS language and, 714
  SMV program, 717–718
Declassification, 162–164
Decoy servers, honeypots as, 976
Decryption, AES transformations, 1200–1201, 1203–1205
Decryption key, encrypted viruses do not encrypt, 786–787
Defender's dilemma, intrusion detection using, 973
Defenses, malware. *See* Malware defenses
Definitions, electronic communications policy
  allowable use, 1242
  general provisions, 1214–1215
  overview of, 1227–1230
  summary of, 129
DEK. *See* Data encipherment key (DEK)
Delay, as form of usurpation, 8–9
Delegation, 7–8, 119–120

Deletion
  file, 1082–1084
  HRU model allowing, 82
  improper deallocation or, 1131–1132
Demand operations, SPM, 72–75
Demilitarized zone (DMZ)
  anticipating attacks within, 1027
  configuring inner firewall, 1016–1017
  configuring internal network, 1022–1023
  configuring outer firewall, 1014–1015
  defined, 1011
  firewalls between internal network and, 573
  firewalls between Internet and, 573
  network organization, servers in, 1017
  science, 946
Demonstrable conformance, CC methodology, 752
Denial of receipt, as deception, 8
Denial of service (DoS)
  attempting to block availability, 6
  definition of, 204
  disallowing deadlocks, 202–203
  as form of usurpation, 8–9
  protection base, 213–215
  resources/services unavailable in, 201–202
  security threats asf, 650–651
  when resource or service is not available, 201–202
Denial of service models
  availability policies and, 203–204
  constraint-based, 204–210
  further reading on inhibiting attacks, 223–224
  state-based, 210–215
Denial of service protection base (DPB), resource allocation, 213–215
Deployment stage, life cycle process, 637–638
Derivable state, safety analysis of SPM, 75–77, 79
Derived rules, natural deduction, 1181–1182
Derived Test Requirements (DTR) for FIPS PUB 140–2, 748
Derived Test Requirements for FIPS PUB 140–2, 748
Descriptor (handle), access control information, 1107
Design
  access to roles/commands, 1106–1110
  auditing system. *See* Auditing, designing system
  implementation and, 15–16
  implementation in HDM, 707
  privacy-enhanced electronic mail, 386
  program security framework, 1104–1105
  testing, 1142–1143
  for validation, 1137–1138
Design assurance, system/software
  design documents contents, 665
  design principles, 662–664
  external interfaces, 666–673
  internal design, 673–675
  need for, 630
  overview of, 14–15
  security functions, 665–666
  TCSEC requirements, 733
  techniques for, 662–664
  throughout life cycle, 633–634
Design principles
  principle of complete mediation, 460–461
  principle of economy of mechanism, 459–460
  principle of fail-safe defaults, 458–459
  principle of least astonishment, 464–465
  principle of least authority, 458
  principle of least common mechanism, 463–464
  principle of least privilege, 457–458
  principle of open design, 461–462
  principle of separation of privilege, 463
  psychological acceptability, 465–466
  review, 466–469
  supporting assurance, 664
  underlying ideas, 455–457
DESIGNATOR type, SPECIAL specification, 703
Destroy object rule, HRU model, 82
Destroy subject rule, HRU model, 82
Detection mechanisms
  deadlock, 203
  as goal of security, 10
  and integrity, 5
Detection (true positive) rate, intrusion detection, 925
Deterministic noninterference
  access control matrix, 266–268
  composition of secure systems, 270–271
  overview of, 259–263
  security policies changing over time, 268–270
  unwinding theorem, 263–265
Deterministic packet selection, IP header marking, 981
Developers class, policy development, 1008–1010
Development (D) category, Lipner, 179–180
Development data for existing products (DDEP) class, 1007–1009, 1011
Development data for future products (DDFP) class, 1007–1009, 1011
Development (ID) entities category, Lipner, 181–182
Development system
  attacking to test security, 1047
  authentication, 1054–1055
  file configuration, 1063–1066
  network configuration, 1045–1047
  policies, 1037–1041
  process configuration, 1054–1055
  retrospective on system security, 1067–1068
  user configuration, 1052–1053
Device, digital rights management, 242
Devices, user security
  monitors and characters, 1086
  monitors and Windows systems, 1086–1087
  smart terminals, 1085–1086
  writable devices, 1084–1085
DEX bytecode. *See* Dalvik executables (DEX) bytecode
Dictionaries, PACL for, 532
Dictionary attacks
  challenge response and, 439–440
  on-line, 430–432
  password guessing in, 427
  salting to thwart, 429–430
  withstanding off-line, 428–430
DIDS. *See* Distributed Intrusion Detection System (DIDS)
Differential cryptanalysis, 301, 305
Diffie-Hellman ciphers
  instant messaging with ECDH, 390–392
  public key exchange, 339–341
  as TLS interchange ciphers, 394
Diffusion, in cryptosystems, 290
Digital Equipment Corporation, virtual machines, 583–584

Digital forensics
  anti-forensics, 994–996
  defined, 987
  overview of, 987
  practice of, 990–994
  principles, 987–990
  review, 996–1001
Digital rights management (DRM), 241–244, 778–779
Digital signatures
  defined, 318
  El Gamal, 321–323
  overview of, 318–319
  public key signatures, 319–323
  RSA, 319–321
  secret key signatures, 319
Dijkstra's Banker's Algorithm, deadlock avoidance, 203
Direct recognition goal, covert flow tree, 605–606
Direct trust, 190, 195
Director
  AAFID, 953
  DIDS, 950
  intrusion detection system, 942
  intrusion detection system architecture, 945–946
Directories, in Trusted Solaris, 148–151
Disabling, thwarting on-line dictionary attacks, 431
Disclosure
  as class of threat, 7
  commercial vs. military integrity policies, 174
  confidentiality policies prevent unauthorized, 141
  military security policy constraints on, 114
Disconnection, thwarting on-line dictionary attacks, 431
Discovery phase, GISTA, 836
Discrete logarithm problem, Diffie-Hellman, 339–341
Discrete logarithm problem, El Gamal, 307–309
Discretionary access control (DAC)
  Bell-LaPadula Model using, 142, 153
  built-in security vs. adding later in UNIX, 656–657
  security policy, 117–118
  security policy changing over time, 268–270
  TCSEC functional requirements, 731
  Trusted Solaris, 147–148
Discretionary protection, TCSEC, 733
Disjunction, propositional logic, 1179
Disposition, electronic communications policy, 1227, 1241
Disruption, as class of threat, 7
Distance to neighbor, anomaly detection, 930–931
Distinguished Names, 476, 479–480, 481–482
Distinguished rights, 33, 56
Distributed denial of service attack (DDoS), 215, 796
Distributed Intrusion Detection System (DIDS), 949–952
Distributed security enforcement, 651–652
Distribution
  deployment stage of life cycle process, 637
  of program, 1146–1147
Diversity, intrusion detection using, 972–973
Divisibility, improper indivisibility, 1138–1139
DMZ. *See* Demilitarized zone (DMZ)
DMZ DNS server
  network configurations, 1044–1045
  network infrastructure, 1013
  network organization practicum, 1020
DMZ log server, 1020–1021, 1027
DMZ mail server, 1013, 1015–1018
DMZ WWW server
  basic security policy of, 1036–1037
  configuring outer firewall, 1015
  devnet developer security policy vs., 1041
  file configuration, 1061–1063, 1065–1066
  network configuration, 1042–1045, 1047
  network infrastructure, 1014, 1018–1019
  policy configuration, 1036–1037
  process configuration, 1053–1105
  retrospective on system security, 1066–1067
  system security and, 1035–1036
  system security authentication, 1053–1105
  user configuration, 1048–1050, 1052–1053
DNS. *See* Domain Name Service (DNS)
DNSKEY RR. *See* public key resource record (DNSKEY RR)
DNSSEC DNS. *See* Domain Name System Security Extensions (DNSSEC DNS)
Docker, isolation features of, 585
Documentation
  additional FIPS 140-2, 748
  penetration testing usefulness from, 829
  security trade-offs/attendant risks, 1136
  and specification, 675–677
  TCSEC assurance for product, 733
Documentation, design contents
  design document specification, 673–675
  external functional specification, 666–668
  internal design description, 668–673
  overview of, 664
  security functions summary specification, 665–666
Documented or known (overt) purpose, Trojan horses, 776, 781
Domain flux botnet, 795–796
Domain Name Service (DNS)
  amplification attack, 221
  associating host names with IP addresses, 485–487
  security extensions for integrity, 487–488
  security issues, 487
Domain Name System Security Extensions (DNSSEC DNS), 487–488
Domain-type enforcement language (DTEL), 121–125, 529
Domain value, cookies, 489
Domains
  DIDS, 950–951
  DTEL associating subjects with, 122–125
  in Schematic Protection Model, 69
DoS. *See* Denial of service (DoS)
Double flux botnet, 795
Downgraded directory, Trusted Solaris, 147
DPB. *See* denial of service protection base (DPB)
Drawbridge library OS, process isolation, 586
Drive-by download, defined, 798
DRM. *See* digital rights management (DRM)
DroidDisintegrator, 569–570
Ds-property, Bell-LaPadula
  Basic Security Theorem, 153, 155, 165–167
  Multics system, 159–161
  rules of transformation, 155, 157
DT. *See* direct trust (DT)
DTE, configuring sandboxes via, 587
DTEL. *See* domain-type enforcement language (DTEL)
DTK. *See* Deception Tool Kit (DTK)
DTR. *See* Derived Test Requirements (DTR)

Dual mapping, nontransitive information flow, 546–547
Duff, Tom, 782
Dynamic access control lists, Cisco routers, 527–528
Dynamic debuggers, sandboxes via, 587
Dynamic identifiers, 485–487
Dynamic information flow analysis tool, TaintDroid, 568–570
Dynamic loading, 1122, 1128
Dynamic mechanisms, information flow, 562–566
Dynamic-Typed Access Matrix Model, 102

E

EALs. *See* Evaluation Assurance Levels (EALs)
Earlybird worm detector, 809–810
Eavesdropping (snooping), 7
ECB. *See* Electronic codebook (ECB) mode
ECDH. *See* elliptic curve Diffie-Hellman (ECDH)
Economy of mechanism. *See* Principle of economy of mechanism
Economy of mechanism principle, 459–460
EDE mode. *See* Encrypt-Decrypt-Encrypt (EDE) mode
Edge adding operations, in models, 88–89
EES. *See* Escrowed Encryption Standard (EES)
Effective set (ES) of privileges, Trusted Solaris, 525
Effective set of privileges, processes, 524–525
Effective UID, 474
Efficiency, thwarting off-line dictionary attacks, 428–429
EFTA. *See* European Free Trade Association (EFTA)
EHDM. *See* Enhanced Hierarchical Development Methodology (EHDM)
EKE. *See* encrypted key exchange (EKE)
El Gamal cryptosystem, 307–309, 315
El Gamal digital signatures, 321–323
Electronic codebook (ECB) mode, 302, 369
Electronic communications
  automated electronic mail processing, 1092–1093
  failure to check certificates, 1093–1094
  sending unexpected content, 1094
  user security and, 1092
Electronic communications policy, UCD
  acceptable use policy. *See* Acceptable use policy, UCD
  allowable use, 1216–1220, 1241–1246
  Appendix A, Definitions, 1227–1230
  Appendix B, References, 1230–1232
  Appendix C, Access Without Consent, 1232–1233
  general provisions, 1213–1215
  introduction, 1212–1213
  overview of? 127–129
  posting and authority to change, 1233–1234
  privacy and confidentiality, 1220–1225
  retention and disposition, 1227
  security, 1225–1227
  user advisories, 1234–1241
Electronic mail
  basic design, 386–387
  design principles, 385–386
  instant messaging and, 389–393
  other considerations, 387–388
  PEM and OpenPGP, 388–389
  protocols (PEM and OpenPGP), 385
  state of typical network service, 384–385
Electronic mail anonymizers
  Cypherpunk remailers, 492–493
  Mixmaster remailers, 493–494
  pseudo-anonymous remailers, 491–494
Electronic voting systems, physical isolation of, 583
Electronmagnetic radiation emissions, side channel attacks, 282
Elimination rules, natural deduction, 1180–1181
Elliptic curve ciphers
  El Gamal digital signature using, 322–323
  instant messaging with ECDH, 390–392
  public key cryptography, 312–315
  TLS interchange cipher in Diffie-Hellman, 394
Elliptic curve Diffie-Hellman (ECDH), instant messaging, 390–392
Ellis, James, 306
Emergent faults, Aslam's model, 859
Employees class, policy development, 1008–1010
Encapsulating security payload (ESP), IPsec
  architecture, 405–407
  message security, 403–404
  overview of, 408–410
Encoding of characters, improper naming and, 1131
Encrypt-Decrypt-Encrypt (EDE) mode, 302, 376
Encrypted key exchange (EKE), defeating off-line dictionary attacks, 440–441
Encrypted viruses, 786–787
Encryption
  AES transformations for, 1199–1201
  authenticated cipher, 377–381
  block ciphers using multiple, 375–377
  electronic communications policy at UCD, 1226–1227
  networks and cryptographic protocols, 382–384
  order of AES transformations for, 1203–1205
Encryption standards
  AES, 1196–1205
  DES, 1191–1195
  review, 1205
End entity certificate, X.509 PKI, 350
End-to-end protocols, networks/cryptography, 381–384
Endorsements, electronic communications policy, 1218
Enforcement
  of acceptable use policy, 1209
  Clinical Information Systems Security Policy, 238
  rules, Clark-Wilson integrity model, 184–186
Engineering, manufacturing stage of life cycle, 637
Enhanced Hierarchical Development Methodology (EHDM), 705, 710–711, 713
Entity name, tickets in Schematic Protection Model, 69
Entropy
  password strength and, 432–433
  as uncertainty, 540–541, 1163
Entropy and uncertainty
  conditional and joint probability, 1163–1165
  conditional entropy, 1167–1168
  joint entropy, 1166–1167
  overview of, 1165–1166
  perfect secrecy, 1168–1169
Entropy-based analysis, information flow, 540–541
Enumerating Further, ISSAF, 833–834
Environment
  CAPSL specification for, 721
  emergent faults in Aslam's model, 859
  risk analysis as function of, 18

Ephemeral Diffie-Hellman, 394
Equifax breach of 2017, 638–639
Equivalent Inverse Cipher, AES, 304, 1203–1205
Eradication phase, intrusion handling, 977–980
EROS. *See* Extremely Reliable Operating System (EROS)
Error handling
  Data Mark Machine and, 565
  improper validation, 1134–1135
  in reading/matching routines, 1116–1117
  in second-level refinement to access control module, 1114
  testing composed modules, 1145
ES privileges. *See* effective set (ES) privileges
Escrowed Encryption Standard (EES), and Clipper Chip, 355–357
ESP. *See* Encapsulating security payload (ESP)
ESPM. *See* Extended Schematic Protection Model (ESPM)
EU. *See* European Union (EU)
European Free Trade Association (EFTA), SOG-IS agreement, 762–764
Evaluation Assurance Levels (EALs), CC, 750–751, 759–763
Evaluation classes, TCSEC, 730, 733–734, 735–737
Evaluation levels, ITSEC, 738, 740–741
Evaluation of systems
  CISR 1991, 742–744
  Common Criteria (CC), 749–765
  Federal Criteria (FC), 744–745
  FIPS 140, 746–748
  goals of formal evaluation, 727–730
  international efforts and ITSEC, 737–742
  other commercial efforts, 744
  overview of, 727
  review, 768–771
  SOG-IS agreement, 762–763
  SSE-CMM model, 765–768
  TCSEC, 730–737
Evaluation process
  CC, 761–762
  ITSEC, 741
  TCSEC, 734–737
Event engine, Bro, 937
Exact conformance, CC, 752
Exception handling, improper indivisibility in, 1138
Exceptions
  causing information flow problems, 557–558
  electronic communications policy, 128
  testing module, 1145
Exchange. *See* Key exchange
Executable files, macro viruses can infect, 785
Executable infectors
  computer viruses as, 783–784
  multipartite viruses as, 783–784
  stealth virus as, 786
  TSR viruses as, 786
  Zmist computer virus, 789
Execute right, access control matrix, 33
Execution phase, computer viruses, 781, 817
Execution phase, computer worms, 791
Execution trace of subject, 939
Existence
  of data, preserving confidentiality, 4
  as property of covert channels, 595–596
Existential security analysis instance, 96–98
Exokernel, isolation via library OS, 585–586
Expansion table, main DES algorithm, 1193
Expansive packet marking, IP header, 981
Expert system, 932–938, 950–952
Expires field, cookies, 489
Explicit flows of information
  checking flow requirements, 562
  entropy-based analysis and, 540–541
  goto statements and, 554–555
  infinite loops and, 558
Exploitable logic error class of flaw, RISOS study, 851
Exploitable vulnerabilities, defined, 825–826
Exploratory programming model, software, 644
Exponential backoff, thwarting on-line dictionary attacks, 430–431
Expressions, functional programming via mathematical, 721
Expressiveness
  ATAM vs. TAM, 101
  ESPM vs. SPM, 90–92
  state-matching reductions and, 98–99
Extended components definition
  CC protection profiles, 753
  ST, 752
Extended Euclidean Algorithm, 1157–1161
Extended Schematic Protection Model (ESPM)
  multiple parenting, 83–88
  security properties of HRU vs., 94–101
  simulation/expressiveness, 88–92
  Typed Access Matrix Model similar to, 92–94
Extended scheme, salting, 429–430
Extensible markup language (XML), security policies, 137–138
Extension_list, TLS handshake protocol, 397
Extensions, certificate, 347, 351–352
External events, triggering logic bombs, 797
External functions
  design document specification, 666–668
  design documentation, 665
  requirements tracing/informal correspondence, 677–680
Extreme Programming (XP), Agile, 643
Extremely Reliable Operating System (EROS), 519, 536
Eyes, biometric authentication, 443–444

F

Face recognition, biometric authentication, 444
Fail-safe defaults. *See* Principle of fail-safe defaults
Failed attacks, dealing with, 1027–1028
Failure symbol node, covert flow trees, 603
Fairness constraints, SMV program, 717–718
Fairness policy, 207–208, 216
FairPlay digital rights management, Apple iTunes store, 242–243
False alarm rate (false positive rate), intrusion detection, 925
False identity, 1211, 1218
Fast flux botnet, 795
Father Christmas worm, 791, 803
Fault trees, covert flow trees, 602–610
FC. *See* Federal Criteria (FC)
FEAL, modern symmetric cipher, 302
Feature descriptor, biometrics, 343
Federal Criteria (FC), 743–745
Federal Information Processing Standard (FIPS) Publication 140-2, 727, 746–749

Feedback
  in Kanban, 643
  noninterference-secure systems and, 276–277
Fenton's Data Mark Machine, 562–565
FER. *See* final evaluation report (FER)
Fielded product life stage, life cycle process, 638–639
Fielding the system, waterfall life cycle model, 641
File descriptors, Capsicum, 589
File system, auditing
  comparing NFSv2 and LAFS, 907
  LAFS, 905–907
  NFSv2, 900–905
  overview of, 900
File Transfer Protocol (FTP)
  access control matrix model and, 33–34
  Class FTP, CC security requirements, 758
  network configuration for development system, 1046
  on systems other than development systems, 1045–1046
Files
  improper changes in contents of, 1128
  improper naming of, 1129–1131
  permissions for access control, 1120–1121
  race conditions in, 1128–1129
Files, user security
  deletion, 1082–1084
  group access, 1081–1082
  identifying by assigning names, 472–473
  improper change in contents of, 1128
  overview of, 1080–1084
  permissions on creation, 1081
  system security practicum, 1061–1066
Filter function
  firewalls, 570–573
  specification-based intrusion detection, 939
  SPM, 69–72, 86–87
Filters, AAFID, 953
Final evaluation report (FER), TCSEC evaluation process, 735
Finger protocol, UNIX security flaw, 847–848
Finger veins, biometrics authentication, 443
Fingerd flaw
  buffer overflow, 862–864
  comparison and analysis, 860
  as condition validation error in Aslam's model, 859
  UNIX, 847–848
Fingerprints, biometric authentication, 442–443
Finite-state machine, security policies, 109–113
Finite waiting time policy
  constraint-based denial of service model, 207–208
  SYN flooding analysis and, 216
FIPS Publication 140-2. *See* Federal Information Processing Standard (FIPS) Publication 140-2
Firewalls
  anticipating attacks, 1027
  blocking attacks via, 979
  configuring inner, 1016–1017
  configuring internal network, 1021
  configuring outer, 1014–1015
  as information flow controls, 570–573
  network configurations for systems, 1042–1045
  network organization and, 1012–1014
First-level refinement, access control module, 1111–1112
Flame worm, 795
Flaw classes
  Aslam's model, 859–860
  NRL taxonomy, 857–859
  Protection Analysis (PA), 852–854
  RISOS study, 849–851
Flaw detection module, automated penetration analysis tool, 873
Flaw elimination, Flaw Hypothesis Methodology, 830, 832–833
Flaw generalization, Flaw Hypothesis Methodology, 830, 832, 843–844
Flaw hypothesis, Flaw Hypothesis Methodology
  Burroughs B5700, 839–840
  corporate computer system, 840–841
  ISSAF version, 833
  Michigan Terminal System, 838–839
  OSSTMM version, 835
  overview of, 830
  in PTES, 837
  UNIX system, 842
  using, 830–831
  Windows system, 844
Flaw Hypothesis Methodology
  flaw elimination, 832–833
  flaw generalization, 832
  flaw testing, 831–832
  goal of vulnerability analysis, 845–846
  information gathering and flaw hypothesis, 830–831
  penetration testing methodology springs from, 829
  problems with, 845
  steps of, 830
Flaw Hypothesis Methodology versions
  GISTA, 835–836
  ISSAF, 833–834
  OSSTMM, 834–835
  PTES, 836–837
Flaw testing, Flaw Hypothesis Methodology
  Burroughs B5700, 840
  corporate computer system, 840–841
  ISSAF version, 833
  Michigan Terminal System, 838–839
  OSSTMM version, 835
  overview of, 830
  in PTES, 837
  UNIX system, 842–843, 844
  using, 831–832
Flooding attacks
  availability and, 215–221, 223
  other types of, 221–222
  using IDIP to handle, 979
Flow-based model of penetration analysis, Gupta and Gligor, 869–872
Flow function, safety analysis of SPM, 75–76
Follow-up phase, intrusion handling, 980–985
Forensics. *See* Digital forensics
Formal evaluation methodology, 728
Formal languages, 676–677
Formal methods
  current verification systems, 713–721
  early formal verification techniques, 705–713
  formal specifications, 702–705
  formal verification techniques, 699–702
  functional programming languages, 721–723
  overview of, 699

  proving programs are correct, 695
  review, 723–726
Formal model
  Bell-LaPadula Model, 151–158
  Chinese Wall Model, 230–233
Formal proof mechanisms, 681–682
Formal security evaluation, 727–730
Formal specifications
  defined, 702
  documentation and, 676–677
  Gypsy, 711–712
  justifying design meets requirements, 681–682
  NPA Temporal Requirements Language (NPATRL), 720–721
  overview of, 702–705
  Prototype Verification System (PVS), 713–715
  SPECIAL. *See* SPECIAL formal specification language
  Symbolic Model Verifier (SMV), 716–718
Formal transformation model, software development, 644–645
Formal verification
  formal specification as part of, 703
  overview of, 699–702
  penetration testing vs., 826
  of products, 722–723
  proving absence of vulnerabilities, 826
Formal verification, current techniques
  Naval Research Laboratory (NRL) Protocol Analyzer (NPA), 720–721
  overview of, 713
  Prototype Verification System (PVS), 713–716
  Symbolic Model Verifier (SMV), 716–720
Formal verification, early techniques
  Boyer-Moore theorem prover, 709–710
  Enhanced HDM, 710–711
  Gypsy Verification Environment (GVE), 711–713
  Hierarchical Development Methodology (HDM), 705–708
  overview of, 705
Formulas
  compound sentences as, 1179
  connectives of propositional logic, 1179
  reaching proof using truth tables, 1182–1183
  well-formed, 1182
Forward search
  countermeasures to, 312
  overview of, 332
  precomputing possible messages, 367–368
  preventing with session keys, 332
Foundational results
  basic results, 51–56
  comparing expressive power of models, 81–94
  comparing security properties of models, 94–101
  general question, 49–51
  overview of, 49
  review, 101–105
  Schematic Protection Model (SPM), 68–81
  Take-Grant Protection Model. *See* Take-Grant Protection Model
Framework, design for program security, 1104–1105
Frameworks, vulnerability
  Aslam's model, 859–860
  comparison and analysis of, 860–864
  NRL taxonomy, 857–859
  Protection Analysis (PA) model, 851–856
  review, 864
  RISOS study, 849–851
  structure determined by goals of, 849
  vulnerability classification, 845–848
FreeBSD system
  availability during flooding attacks, 219–220
  implementing Biba's strict integrity model, 178
  long passwords in v10, 417
  supporting audit ID in v10.3, 1049
FTP. *See* File Transfer Protocol (TFP)
Full specification verification, 700
Function *f*, main DES algorithm, 1191–1192
Function flow generator, automated penetration analysis, 873
Functional (black box) testing, 688–689
Functional programming languages, 721
Functional requirements
  CC, 752, 756–759
  CISR, 743
  Federal Criteria, 745
  TCSEC, 731–732
  United Kingdom IT Security Evaluation and Certification Scheme Certification Body, 738
  waterfall life cycle model, 639
Functional specification, design documentation, 666–668
Functions
  access control module issues, 1114–1117
  documentation for high-level security, 665–666
  at heart of SPECIAL specification, 703–704
  role as group tying membership to, 475
  separation of, 174
  validation, 1137–1138
Fuzzy time, mitigating covert channels, 617

G

Galois Counter Mode (GCM), AEAD, 379–381
GCIR. *See* generalized conflict of interest relation (GCIR)
GCM. *See* Galois Counter Mode (GCM)
Geinimi, Android cell phones, 776–777
Gemsos system, 655
General provisions, electronic communications policy, 1213–1215
Generalized noninterference, policy, 274–277
Generation effect, user-created passwords, 421–425
Generation, key, 341–343
Genesis, NRL taxonomy flaws, 857
Get-read rule, 159–160, 165
GISTA. *See* Guide to Information Security Testing and Assessment (GISTA)
Give-access rule, Bell-LaPadula, 702–705
Give-read rule, Multics system, 160–161
Global identifier, on web, 486
Global object tables, 522
Global Positioning System (GPS), location authentication, 445–446
Goals
  attack trees and, 961–964
  of attackers, 959–960
  of attacks, 960–961
  of confidentiality policies, 141–142
  of covert flow tree, 605–606
  of formal evaluation, 727–730
  of intrusion detection systems, 918–919

- in network security policy practicum, 1006
- of penetration studies, 827–828
- of privacy-enhanced electronic mail, 386
- role of requirements in assurance, 631–632
- of security, 10–11

Good symbols node, covert flow trees, 603
Good_cert_authorities, TLS handshake protocol, 398
Google Chrome, sandboxing using Capsicum, 590
Google, two-factor authentication, 446–447
Goto statements, information flow and, 554–556
Government (military)
- Bell-LaPadula Model. *See* Bell-LaPadula Model
- early driver of computer security research, 729–730
- integrity policies, 174
- protection of citizen's privacy, 141

GPS. *See* Global Positioning System (GPS)
Grammar, specifying log content using, 887–888
Grant, digital rights management, 242
Grant policies, break-the-glass policy, 249
Grant rule, Take-Grant Protection Model
- formulating as instance of SPM, 71–72
- interpretation of, 61–63
- overview of, 56
- sharing of rights, 58–59
- theft, 62–66

Graph-based representation
- comparing security models, 88–92
- Take-Grant Protection Model. *See* Take-Grant Protection Model

Graphical interfaces
- intrusion detection systems, 946–947
- Network Security Monitor, 949

Graphical Intrusion Detection System (GrIDS), 946–947, 952
Graphical passwords, authentication, 425–426
Graphs, attack, 969–971
Greatest lower bound, lattices, 1154
GrIDS. *See* Graphical Intrusion Detection System (GrIDS)
Groups, 475–476, 1081–1082
GTbot, 793
Guessing entropy, passwords, 432–433
Guide to Information Security Testing and Assessment (GISTA), 835–836
Gupta and Gligor's penetration analysis theory, 868–873
GVE. *See* Gypsy Verification Environment (GVE)
Gypsy Verification Environment (GVE), 711–713

H

Halt instruction, Data Mark Machine, 564
Halting problem, 52, 54
Handle (descriptor), access control information, 1107
Handshake protocol, TLS, 397–399
Hardware-based virtual machines (HVMs), 584
Hardware-supported challenge-response procedures, 439
Harrison-Ruzzo-Ullman (HRU) Model
- basic results, 51–56
- as central to safety analysis, 82
- of computer security, 49
- ESPM security properties vs., 94–101
- general question, 49–51
- relationship between ESPM and, 87–88
- simulation and expressiveness of, 88–92
- SPM vs., 82
- Typed Access Matrix Model, 92–94

Hash functions
- key crunching with, 420
- Merkle's tree authentication, 344–345
- UNIX password mechanism, 417

Haskell functional programming language, 721
Haystack, anomaly detection, 922
HDM. *See* Hierarchical Development Methodology (HDM)
Heartbeat protocol extension, TLS, 399–400
Hierarchical Development Methodology (HDM)
- early formal verification via, 705–707
- Enhanced HDM (EHDM), 705, 710–711
- formal verification example, 701–702
- verification in, 707–708

Hierarchical domains, Ponder, 119
Hierarchies
- certificate-based key management, 477–478
- control scheme, large botnets, 793
- formal model of Bell-LaPadula Model, 151–158
- RBAC adding role, 247

Hierarchy consistency checker, HDM, 706
Hierarchy Specification Language (HSL), HDM, 705–706
High-level design, user interface, 1104–1105
High-level policy languages, 119–125
High-level test specifications (HLTS), PGWG, 693–695
High severity behavior, adware/madware, 798
Highland, Harold Joseph, 782
History
- access control by, 36–37
- safety analysis of SPM, 75, 77–81

HKDF. *See* HMAC-based key derivation function (HKDF)
HLTS. *See* High-level test specifications (HLTS)
HMAC-based key derivation function (HKDF), instant messaging, 390–392
HMAC-Based One-Time Password Algorithm (HOTP), 437–438
HMAC functions, 317–318
HMAC-SHA-1, one-time passwords, 436–438
HMAC_SHA256, instant messaging, 390–392
Hold and wait, deadlocks from, 202
Hold specification, Gypsy, 712
Homomorphic encryption schemes, 325
Honeynet Project, 976
Honeypot (honeyfile or honeydocument), 976–977
Host-based information gathering, agents, 942–943
Host monitoring, 949–952
Host names
- DNS associating IP address with, 485–486
- identity on web and, 484–485
- security issues with DNS, 487
- specifying, 1131

Hostname resource record (NSEC RR), DNSSEC, 488
Hosts, network, 381–383
Hot fixes, maintenance, 695–696
HOTP. *See* HMAC-Based One-Time Password Algorithm (HOTP)
HSL. *See* Hierarchy Specification Language (HSL)
Human class, OSSTMM, 834
Human factors
- of graphical passwords, 426
- implementing security controls, 20
- organizational problems, 20–21
- people problems, 21–22

  - principle of least astonishment, 464–465
  - principle of psychological acceptability, 465–466

HVMs. *See* Hardware-based virtual machines (HVMs)

Hybrid policies
  - break-the-glass policies, 249–250
  - Chinese Wall Model. *See* Chinese Wall Model
  - clinical information systems security policy, 236–239
  - ORGCON or ORCON access control, 239–244
  - overview of, 227
  - RBAC, 244–249
  - review, 250–253

HYDRA, amplifying capabilities for, 521

Hypertext display technique, audit browsing, 908

Hypervisor (virtual machine monitor), 1171–1172

Hypervisors, virtual machines and, 583–585

Hypotheses, Gupta and Gligor, 869

I

I&A. *See* Identification and authentication (I&A)

IBAC. *See* Identity-based access control (IBAC)

IBM, cryptographic locks and keys, 527

ICMP packets, Smurf attacks, 221

IDEA, modern symmetric cipher, 303

Ideas, conception stage of life cycle process, 635–636

Identification and authentication (I&A), TCSEC, 732

Identifiers, static or dynamic, 485–487

Identity
  - anonymity on web, 490–501
  - capabilities encapsulating object, 518
  - confirming. *See* Authentication
  - files and objects, 472–473
  - groups and roles, 475–476
  - intruders changing, 950
  - naming and certificates, 475–484
  - overview of, 471
  - review, 501–505
  - theft, 501
  - understanding, 471–472
  - user, 473–475
  - on web, 484–490

Identity-based access control (IBAC), security policy, 117–118

Identity pair IK, instant messaging, 390

IDES Intrusion Detection Expert System (IDES)

IDEVAL dataset, 925, 930–931

IDIOT. *See* Intrusion Detection In Our Time (IDIOT)

IDIP. *See* Intruder Detection and Isolation Protocol (IDIP)

IFD. *See* immediate forward dominator (IFD)

IG. *See* Security Requirements for Cryptographic Modules (IG)

IIS web servers, and Code Red I computer worm, 791–792

IKE protocol. *See* Internet Key Exchange (IKE) protocol

Immediate forward dominator (IFD), basic blocks, 555–556

Implementation
  - access control module, 1114–1117
  - Agile, 642–644
  - auditing system design and, 886–887
  - computer security and, 15–16
  - improper isolation of detail, 1123–1125
  - noncryptographic. *See* Noncryptographic implementation mechanisms
  - rules, 1247–1248
  - verification in HDM, 707–708
  - in waterfall life cycle model, 640

Implementation assurance
  - considerations, 685–686
  - defined, 634
  - implementation management, 686–687
  - justifying implementation meets design, 687–688
  - need for, 630
  - overview of, 15–16
  - security testing for, 688–689
  - security testing using PGWG, 689–695

Implementation Guidance for FIPS PUB 140-2, 748

Implementation-level constructs, DTEL, 121–125

Implication, connectives of propositional logic, 1179–1180

Implicit flows of information
  - Data Mark Machine studying, 562–565
  - defined, 541
  - dynamic mechanisms involving, 562
  - may occur in goto statements, 554–556

Implicit sharing of privileged/confidential data class of flaw, RISOS study, 850

Importing, file systems from another zone, 149–150

Improper change, program security, 1125–1129

Improper choice of operand or operation flaws, Protection Analysis (PA), 853–854, 1139–1141

Improper deallocation or deletion, program security, 1131–1132

Improper indivisibility, program security, 1138–1139

Improper naming, program security, 1129–1131

Improper protection flaws, Protection Analysis, 852–853

Improper synchronization flaws, Protection Analysis, 853–854

Improper validation flaws, 853, 1132–1138

Inadequate identification/authentication/authorization class of flaw, RISOS study, 850

Incident prevention, 971–975

Incident response groups, 985–987

Incomplete parameter validation class of flaw, RISOS study, 849–850

Inconsistent parameter validation class of flaw, RISOS study, 849–850

Inconsistent static analysis, state-based auditing, 894

Incremental development, Gypsy language, 711

Independence, PEM design principles, 386

Indirect trust, 190

Indirection, revoking rights in capability systems, 522

Indivisibility, improper, 1138–1139

Induction phase, OSSTMM modules, 835

Inductive verification techniques, 700–701, 703

Inetd daemon, development system, 1059–1060

Infection phase, computer viruses, 817

Infection vectors, computer viruses, 782–785

Inferred recognition goal, covert flow tree, 605–606

Inferred via-goal, covert flow tree, 606

Infinite loops, unexpected information flow, 558

Informal arguments, 680–681

Informal description, Chinese Wall Model, 228–230

Informal (representation) correspondence, 677–680

Information
  - aggregation in commercial integrity policies, 174
  - assurance, 628
  - extracting from data in digital forensics, 992
  - filtering, 120

  leakage, 111
  processing/presenting in digital forensics, 990
Information flow
  basics and background, 539–540
  Bell-LaPadula Model restricting, 183
  concurrency, 558–561
  configuring internal network, 1021
  configuring outer firewall, 1014–1015
  dynamic mechanisms, 562–566
  entropy-based analysis, 540–541
  examples of controls, 567–573
  high assurance, 655
  integrity mechanisms, 566–567
  metrics for malware containment, 812–813
  models and mechanisms for, 541–542
  nonlattice policies for, 542–548
  policies. *See* Confidentiality policies
  review, 573–577
  soundness, 561–562
  static mechanisms, 548–558
  uncovering covert channels in, 601–602
Information flow generator, penetration analysis, 873
Information gathering, Flaw Hypothesis Methodology
  Burroughs B5700, 839
  corporate computer system, 840
  ISSAF version, 833
  Michigan Terminal System, 837–839
  overview of, 830
  in PTES, 837
  UNIX system, 841–842
  using, 830–831
  Windows system, 844
Information Systems Security Assessment Framework (ISSAF), 833–834
Information Technology Security Evaluation Criteria (ITSEC)
  assurance requirements, 739
  CISR, 742
  evaluation levels, 740–741
  evaluation process, 741
  evaluation process limitations, 742
  impacts of, 741–742
  overview of, 738–739
  replaced by Common Criteria, 629
  requirements not found in TCSEC, 739–740
  suitability analysis, 660–662
Information transfer path, Biba Model, 175
Infrastructure as a service cloud, 1024–1025
Infrastructures
  analysis of network, 1013–1017
  key. *See* Key infrastructures
Inheritable set (IS) privileges, Trusted Solaris, 525
Inhibit anyPolicy extension, X.509 PKI certificates, 352
Initial message, instant messaging, 391–392
Initial product assessment report (IPAR), TCSEC evaluation process, 735
Initial protection domain, flaws
  access control file permissions, 1120–1121
  memory protection, 1121–1122
  overview of, 1118
  process privileges, 1118–1120
  trust in system, 1123
Initial state operations, comparing simulation in models, 88–89
Initialization vector, block ciphers, 375, 379
Input
  checking all user, 1136–1137
  parameters, 549–550
Inquest phase, OSSTMM modules, 835
Insecure (unauthorized) states, security policies, 109–113
Insertion phase, computer viruses, 780–781
Insiders
  problems from, 21
  threats to security from, 650–651
Installation of system, deployment stage, 638
Instant messaging
  supplanting some use of electronic mail, 389–393
  transition-based logging, 895
Instructions, improper change over time, 1125–1129
Integration
  as implementation management tool, 687
  of security at beginning, 653–657
  supporting assurance, 685–686
  in waterfall life cycle model, 640–641
Integrity
  as basic to computer security, 5–6
  Chinese Wall Model and, 227
  Clinical Information Systems security policy, 236–239
  in commercial security policies, 114
  constraints, Clark-Wilson integrity model, 184–186
  countering threats with, 7
  cryptography providing, 290
  of data moving from internal network, 1011–1012
  of data moving from Internet, 1011
  DNSSEC providing DNS, 488
  in electronic communications policy, 1226
  file configuration for DMZ WWW server and, 1063
  of information flow mechanisms, 566–567
  protecting master copy of program, 1146
  security policies and, 110–111
  security threats as disruptions of, 650
Integrity levels
  Biba's model, 175–178
  Clark-Wilson model vs. Biba model, 188
  Lipner's full model, 181–182
Integrity models
  Clark-Wilson. *See* Clark-Wilson integrity model
  Lipner. *See* Lipner's integrity matrix model
  SPM subsuming, 82
  trust models vs., 189
Integrity policies
  Biba model, 175–178
  Clark-Wilson integrity model, 183–189
  definition of, 111, 115
  goals of, 173–174
  information flow policy within, 539–540
  Lipner's integrity matrix model, 178–183
  penetration tests violate constraints in, 827–828
  review, 196–200
  security and precision in, 131, 133–135
  trust and, 114–115
  trust models, 189–196
Integrity Value Check (IVC), AH protocol, 407–408
Integrity verification procedures (IVPs), Clark-Wilson integrity model, 184–186
Intel architectures
  privilege and virtual machines, 1174–1175
  ring-based access control for Itanium, 533

Intellectual property, electronic communications policy, 1219–1220
Interaction phase, OSSTMM modules, 835
Interchange key
  Bellare-Rogaway protocol, 336
  PEM design, 386–387
  session key vs., 332
  TLS cryptography, 394
Interface
  for external functional specification, 667
  operations, 207–208
Interference
  electronic communications policy on, 1218–1219
  noninterference. *See* Noninterference, and policy composition
  system security and, 259
Intermediate systems, and network flooding, 216–218
Internal design
  access to roles/commands, 1107–1108
  design documentation, 665, 668–673
  requirements tracing/informal correspondence, 677–680
  specification, 673–675
Internal network
  concealing addresses of, 1013–1014
  configuring inner firewall, 1016–1017
  network organization practicum, 1021–1025
  using firewalls to protect, 573
Internal packet marking, IP header marking, 981
Internet
  anonymity on web. *See* Web, anonymity on
  identity on web. *See* Web, identity on
  isolating electronic voting systems from, 583
Internet Key Exchange (IKE) protocol, 361, 404
Internet of Things, and botnets, 796
Internet Policy Registration Authority (IPRA), 477–481
Internet worm
  as bacterium, 803
  incident response groups and, 985
  overview of, 790–791
  publicizing flaw in UNIX, 847–848
Interpretation, Take-Grant Protection Model, 61–63
Interprocess communication, DMZ WWW server, 1058
Intervention phase, OSSTMM modules, 835
Introduction rules, natural deduction, 1180–1181
Introduction section, CC protection profiles, 752
Intruder Detection and Isolation Protocol (IDIP), 978–979
Intrusion detection
  adding signatures of known attacks, 1028
  anticipating attacks, 1027
  architecture, 942–948
  autonomous agents via AAFID, 952–953
  basic, 918–920
  goals of, 918–919
  host/network monitoring with DIDS, 949–952
  incident prevention via, 971–975
  monitoring network traffic with NSM, 948–949
  organizing systems, 948–953
  principles, 917–918
  review, 954–957
Intrusion Detection Exchange Protocol (IDXP), 947–948
Intrusion Detection Expert System (IDES), 819, 921–922, 924
Intrusion Detection In Our Time (IDIOT), 933–934
Intrusion Detection Message Exchange Format (IDMEF), 947–948
Intrusion detection models
  as adaptive or static, 920
  anomaly modeling, 920–932
  misuse modeling, 932–938
  overview of, 920
  specification-based modeling, 938–941
  summary, 941–942
Intrusion handling
  containment phase, 975–977
  eradication phase, 977–980
  follow-up phase, 980–985
  incident response groups, 985–987
  intrusion response, 975
  phases of, 975–987
Intrusion prevention system, 948
Intrusion response
  containment phase, 975–977
  digital forensics. *See* Digital forensics
  eradication phase, 977–980
  follow-up phase, 980–985
  incident prevention, 971–975
  incident response groups, 985–987
  intrusion handling, 975
  review, 996–1001
`InvMixColumns` transformation, AES decryption, 304, 1201–1205
`InvShiftRows` transformation, AES decryption, 304, 1200–1205
`InvSubBytes` transformation, AES decryption, 304, 1200–1205
IO integrity classification. *See* Operational (IO) integrity classification
IP address hopping, network defense, 973–974
IP addresses
  concealing on internal networks, 1013
  DNS associating host names with, 485–486
  DNS security issues, 487
IP flux botnets, 795
IP header marking, 981
IPAR. *See* Initial product assessment report (IPAR)
IPRA. *See* Internet Policy Registration Authority (IPRA)
IPsec
  AH protocol, 407–408
  architecture, 404–407
  ESP protocol, 408–410
  network layer security via, 402–404
IRC channel, as C&C channel for bots, 793–794
Iris, eye biometrics, 443–444
IS privileges. *See* Inheritable set (IS) privileges
ISL. *See* System Low (ISL) integrity classification
Islands, Take-Grant Protection Model, 59–61
ISO/IEC standardization
  future of Common Criteria, 764–765
  impact of FIPS 140-2, 748
  SSE-CMM, 765
ISO. *See* International Standards Organization (ISO)
ISO/OSI model
  context for host naming, 484–485
  network layer security. *See* IPsec
  networks and cryptography, 381–384
  transport layer security. *See* Transport layer (TLS and SSL) security

Isolation
  confinement problem, 582
  controlled environment, 582–590
  of implementation detail, improper, 1123–1125
  library operating systems, 585–586
  program modification, 590–594
  review, 619–623
  sandboxes, 586–590
  virtual machines, 583–585
ISP integrity classification. *See* System Program (ISP) integrity classification
Israeli (Jerusalem) virus, 783–784
ISSAF. *See* Information Systems Security Assessment Framework (ISSAF)
Issuance policy, CA, 477
Issuer, digital rights management, 242
Iteration, in stages of water life cycle model, 641
Iterative statements, information flow, 553–554
ITSEC. *See* Information Technology Security Evaluation Criteria (ITSEC)
IVC. *See* Integrity Value Check (IVC)
IVPs. *See* integrity verification procedures (IVPs)

J

Jailbreaking, Pegasus spyware, 800
Jailing technique, 432, 971–972
Janus, user-level sandbox, 587–588
Java applets, blocking at firewalls, 572, 979
Java, as type-safe language, 592
Jerusalem (Israeli) virus, 783–784
JIGSAW language, 967
Joint and conditional probability, 1163–1165
Joint creation operation, ESPM, 83–88
Joint entropy, 1166–1167
Justification
  of security requirements, 660–662
  that design meets requirements. *See* Requirements, justifying that design meets

K

Kanban, Agile software development, 643
Kasiski attack method, Vigenére cipher, 294–299
KDDCUPS- 99 (or KDD-99) dataset
  analyzing with neural nets, 928
  analyzing with self-organizing maps, 930
  anomaly detection using distance to neighbor, 931
  intrusion detection evaluations, 925
Keccak hash function, as SHA-3, 317
KEDP. *See* Key Escrow Decrypt Processor (KEDP)
Kerberos
  further reading, 361
  key exchange, 337–338
  user identity, 474–475
Kerckhoff's Principle, security of cryptosystem, 290–291
Kernel function, anomaly detection with SVM, 931–932
Kernels
  audit analysis of NFSv2, 900–901
  building system with security, 654
  containers enforcing isolation, 584–585
  events in system logs, 892
  as formally verified products, 722–723
  hypervisor functioning as, 583–584
  identifying covert channels in source code, 601
  later rootkits altering parts of, 778
  library operating systems enforcing isolation, 585–586
  sandboxes restricting actions, 587–590
Key crunching, 420
Key Escrow Decrypt Processor (KEDP), and Clipper chip, 355–357
Key escrow system
  and Clipper chip, 355–357
  key escrow component, 355
  key storage using, 354–355
  Yaksha security system, 357
Key exchange
  Kerberos, 337–339
  overview of, 332–333
  public key cryptographic, 338–341
  symmetric cryptographic, 333–336
Key infrastructures
  Merkle's tree authentication scheme, 344–345
  overview of, 343–344
  PGP certificate signature chains, 348–350
  PKIs, 350–353
  X.509 certificate signature chains, 346–348
Key length, AES, 303
Key management
  hierarchical certificate-based, 477–478
  key exchange. *See* Key exchange
  key generation, 341–343
  key infrastructures. *See* Key infrastructures
  overview of, 331
  review, 359–365
  revocation, 358–359
  session and interchange keys, 332, 386–387
  storage, 353–358
Key usage extension, X.509 PKI certificates, 351–352
Keyed cryptographic checksums, 317
Keyless cryptographic checksums, 317
Keynote trust management system, 191–194, 198
Keys, PEM design, 386–387
Keystroke dynamics, biometric authentication, 444–445
Knark rootkit, 778
Knowledge-based subsystem, malware containment, 814–815
Known plaintext attack, 291
Konheim's model of single-character frequencies, substitution ciphers, 294

L

Labeled security protection, TCSEC, 734
Labeled zones (zones), directories in Trusted Solaris, 149–150
Labels
  security vs. integrity, 175
  TCSEC functional requirements, 732
  Trusted Solaris security classifications, 146–151
LAFS. *See* Logging and Auditing File System (LAFS)
Lagrange interpolating polynomials, secret sharing via, 530–531
Lampson, 580, 616
Land attack, auditing to detect known violations, 896–897
Lanes of work, in Kanban, 643
Languages
  DTEL, 121–125, 529
  formal specification. *See* Formal specifications
  HSL, 705–706
  programming. *See* Programming languages

Lattach command, LAFS, 906
Lattices
  Bell-LaPadula Model, 143–144, 235
  Bell-LaPadula Model information flow policy, 539
  composition of Bell-LaPadula models, 256–258
  embedding nonlattice policies into, 548
  mathematical nature of, 1153–1155
  as models of information flow policies, 541–542
  nonlattice information flow policies, 542–548
  self-organizing maps with neurons arranged in, 929–930
Law Enforcement Access Field (LEAF), 356
Laws
  electronic communications policy and, 1209, 1215
  key escrow system and, 354–357
  operational issues and, 19–20
Layers
  abstraction in representing attacks, 960–964
  architecture security mechanisms, 652–653
  ISO/OSI model, 381–382
  penetration study, 828–829
  penetration testing at all, 829
  simplifying design to support assurance, 662–664
LEAF. *See* Law Enforcement Access Field (LEAF)
Leakage of information, 580–582
Leaking of rights, determining system safety, 50–52
Least astonishment. *See* Principle of least astonishment
Least authority, principle of, 458
Least common mechanism. *See* Principle of least common mechanism
Leaving system unattended, user security and, 1079
Legal mechanisms, counterattacking and, 983–984
Legal practices, 19–20
Legal transitions, SPM, 75, 77, 79
Lemmata, Gypsy execution of, 712
Less than or equal to relation, lattices, 1153–1155
Levels
  of adware, 797–798
  Biba model integrity, 175–178
  security. *See* Security levels
LFC. *See* locality frame count (LFC)
LFSR method. *See* linear feedback shift register (LFSR) method
Libcapsicum library, Capsicum, 589
Libraries
  confinement constraints via loading, 593–594
  Prototype Verification System (PVS), 714
Library operating systems, isolation via, 585–586
Licenses
  digital rights management, 242
  unacceptable conduct for, 1209
Life cycle
  assurance throughout, 632–634
  of bots in botnet, 793
  building secure/trusted systems, 634–639
  PVS proof checker, 715–716
  waterfall life cycle model, 639–641
Linear cryptanalysis attack, 301, 305
Linear feedback shift register (LFSR) method, synchronous stream ciphers, 371–372
Linear time logic systems, 1186
Link predicates, SPM
  defined, 69
  multiple parenting in ESPM, 85–87
  overview of, 69–70
  putting it all together, 71–72
Link protocols, networks and cryptography, 381–384
Linux Rootkit IV, 777–779, 918–919
Linux system
  adore-ng rootkit on, 778
  Android based on, 568
  Crimea virus targeting, 790
  isolation features of Docker in, 585
Lipner's integrity matrix model
  comparing to Biba, 182–183
  full model, 181–182
  overview of, 178
  use of Bell-LaPadula Model, 178–180
LLTS. *See* low-level test specifications (LLTS)
Loading libraries, process confinement, 593–594
Loadmodule, penetration testing UNIX system, 842–843
Local identifier, on web, 486
Locality frame count (LFC), intrusion detection with system calls, 972
Locard's Exchange Principle, 987
Location
  authentication by, 445–446
  function for obtaining, 1114–1115
  improper choice of operand/operation, 1140
  NRL taxonomy, flaws by, 857
  second-level refinement to access control module, 1113
  unauthorized users accessing role accounts, 1102
Location signature sensor (LSS), location authentication, 445–446
Locks and keys, access control, 526–530
Locky ransomware, 801
Log files
  analysis phase in digital forensics, 993–994
  computer security and, 880
  definition of, 879
  director eliminates unnecessary records in, 945–946
  improper deallocation or deletion of, 1132
  network configurations for system security, 1045
  processes on development systems and, 1060
  transition-based, 895
Log sanitization, auditing system design, 888–891
Logging and Auditing File System (LAFS), 905–907
Logic. *See* Symbolic logic
Logic bombs, triggering on external event, 797
Logical Coprocessor Kernel (LOCK) system
  malware detection on data/information, 812
  sharing procedures and, 818
  type checking, 528–529
Login
  procedure, 1076–1079
  UID, 474
LOKI89, modern symmetric cipher, 302
LOKI91, modern symmetric cipher, 302
LOKI97, modern symmetric cipher, 303
LOOKUP request, audit analysis of NFSv2, 901, 903–904
Loops
  information flow using semaphores, 560
  iterative statements/information flow, 553
  unexpected information flow from infinite, 558
Lotus 1-2-3, virus, 782
Low-level policy languages, 125–126
Low-level test specifications (LLTS), using PGWG, 693–695
Low severity behavior, adware, 797

Low-water-mark policy, Biba Model, 176–177
Lower bound, lattices, 1154–1155
LSS. *See* location signature sensor (LSS)
LUCIFER algorithm, modified as DES, 300

M

MAC. *See* Mandatory access control (MAC)
Machine learning, anomaly detection, 924–925
MacMag Peace virus, 782
Macro viruses, 785
Macro worms, 791
Madware, 798
Maintenance
  assurance during system, 695–696
  fielded product life stage, 638
  waterfall life cycle model, 641
Malware
  adware, 797–799
  bots and botnets, 793–796
  combinations, 803
  computer viruses, 780–790
  computer worms, 790–792
  defined, 775–776
  introduction, 775–776
  logic bombs, 797
  phishing, 802–803
  rabbits and bacteria, 796
  ransomware, 800–801
  review, 820–824
  spyware, 799–800
  theory of computer viruses, 803–807
  Trojan horses, 776–780
Malware defenses
  containment, 812–817
  data and instructions, 811–812
  limiting sharing, 817–819
  notion of trust, 819–820
  scanning, 808–811
  specifications as restrictions, 817
  statistical analysis, 819
Man-in-the-middle attack
  integrity services countering, 7
  problems with SSL, 402
  public key exchange and, 339
Management, computer security incident response team, 986
Management rules
  list of, 1249
  security-related problems. *See* Program security practicum, common programming problems
Mandatory access control (MAC)
  Bell-LaPadula Model, 142, 153
  RBAC as form of, 245–246
  security policy, 118
  TCSEC requirements, 731–732
  Trusted Solaris, 146–151
  UNIX built-in security in vs. adding later and, 656–657
Manifesto for Agile Software Development, 642
Manipulation detection codes (MDCs), malware defense, 808
Manufacturing stage, life cycle process, 636–637
Mapping
  in compositional security analysis, 96–97
  of improper isolation of implementation detail, 1123–1125
  of security threats to security objectives, 651
  of specifications in HDM, 706–707
  strongly security-preserving, 97–99
  of system to existing model for policy definition, 658–659
  that preserves security properties, 95–99
  threats to requirements, 661–662
Markov models, anomaly detection, 922–924
Masquerading, 7–8
Master Comment List, review process, 684
Master secret
  instant messaging, 391–392
  SSLv3 vs. TLSv1.2, 400–401
  TLS, 393–395, 398–399
Mathematical induction, proof technique, 1183–1184
Mathematics
  Boyer-Moore theorem prover, 709–710
  cryptosystem attack using, 291
  El Gamal cryptosystem using, 307–309
  elliptic curve ciphers based on, 312–315
  Extended Euclidean Algorithm and, 1157–1161
  formal security models, 700
  formal specification in BLP security policy as, 702–705
  functional programming, 721
  public key systems based on, 306–307
  RSA cryptosystem, 309–312
  Symbol Model Verifier (SMV) based on, 716–720
  symbolic logic and. *See* Symbolic logic
  transposition ciphers and, 292
  underlying all verification techniques, 700
Matrix, Network Security Monitor, 949
Maximal state, SPM, 75–78, 81
McLean's †-property, Basic Security Theorem, 164–166
McLean's System Z, Bell-LaPadula Model, 166–168
MDCs. *See* manipulation detection codes (MDCs)
Mebroot, Torpig botnet, 794
Mechanisms. *See* Security mechanisms
Mediation. *See* Principle of complete mediation
Medical records, Clinical Information Systems security policy, 236–239
Medium severity behavior, adware, 797–798
Melissa virus, 785
Membership, in group, 475
Memory
  digital forensics and data stored in, 991–992
  improper choice of initial protection domain and, 1121–1122
  network flooding, TCP state and, 218–221
  protecting capabilities, 519
  SYN flooding consuming, 216
Merkle's tree authentication scheme, 344–345
Message integrity check (MIC), PEM, 387, 388
Message key, instant messaging, 390–392
Message transfer agent (MTA), network mail service, 384–385
Metamorphic viruses, 789
Methodologies
  Agile, 642–644
  at each layer of penetration studies, 829
  evidence of assurance/trustworthiness, 629
  Flaw Hypothesis Methodology, 830
  vulnerability analysis goal is developing, 845–846

Metrics
  anomaly detection using distance to neighbor, 930–931
  assurance, 646–647
  intrusion detection methods, 925
  recommendation system, 196
MIC-CLEAR mode, PEM, 388
MIC. *See* message integrity check (MIC)
Michigan Terminal System, penetration testing, 837–839
MieLog, audit browser, 910
Military, as early driver of computer security research, 729–730
Military (governmental) security policies, 113–115
Millen model, SYN flooding analysis, 216
Min-entropy, passwords, 432–433
Minimum Security Functionality Requirements for Multi-User Operating Systems (MISR), FC, 745
Mirai Internet-of-Things botnet, 793
Misordered blocks, ciphertext problems, 368
MISR. *See* Minimum Security Functionality Requirements for Multi-User Operating Systems (MISR)
Misuse detection
  anomaly detection vs., 941–942
  Bro, 937–938
  defined, 920
  IDIOT, 933–934
  overview of, 932–933
  specification-based detection vs., 942
  STAT, 934–937
  summary, 941
Mitigation
  of covert channels, 616–619
  security objectives for identified threats, 651
MITRE tool, 712, 865
`MixColumns` transformation, AES, 304, 1198, 1199, 1203–1205
Mixing functions, software pseudorandom number generators, 342
Mixmaster remailers, 493–494
MLD. *See* Multilevel Directory (MLD)
MLS tool. *See* Multilevel Security (MLS) tool
Mobile computing, wireless network practicum, 1023–1024
Model checking
  formal specification languages in, 703
  formal verification, 700
  overview of, 701
  processing specification to meet constraints, 681–682
Models
  of information flow, 541–542
  intrusion detection. *See* Intrusion detection models specific; Foundational results
  verification techniques based on, 700
Moderator, review process, 682–685
Modes
  AES, 305
  DES, 302
  IPsec, 403
Modification
  constructing covert flow tree, 605
  specifications, 675
  and threats, 7
Modular scheme, salting, 430
Modules
  analyzing OSSTMM channels, 834
  building specifications in EHDM, 710–711
  building specifications in HDM, 706
  defined, 663
  designing access to roles/commands, 1106–1110
  designing framework, 1104–1105
  developing systems designed in, 686
  FIPS 140 cryptographic, 748
  PVS language supporting, 714
  simplifying design to support assurance, 662–664
  SMV program, 717
  SPECIAL module specification, 703
  testing, 1143–1144
  testing composed, 1145
Modus tollens rules, natural deduction, 1181–1182
Monitoring
  network traffic by agents, 943
  network traffic with NSM, 948–949
Monitors
  AFFID system and, 953
  and characters, 1086
  Windows systems and, 1086–1087
Mono-operational protection systems, safety in, 51–52
Monoalphabetic cipher, Vigenére cipher, 294
Monoconditional commands, protection state transitions, 41
Monoconditional monotonic protection systems, safety in, 55–56
Monotonic protection systems, safety in, 55–56
Monotonic Typed Access Matrix (MTAM) Model, 92–94
Monotonicity of release principle, declassification, 163
Moving target defenses, intrusion detection, 973–974
MOVPSL instruction, privilege and virtual machines, 1172–1173
MRA. *See* Mutual Recognition Arrangement (MRA)
MTA. *See* message transfer agent (MTA)
MTAM Model. *See* Monotonic Typed Access Matrix (MTAM) Model
MtE tool kit. *See* Mutation Engine (MtE) tool kit
Multiconditional commands, HRU, 82
Multicreate command, HRU, 82
Multics system
  analysis procedure for PA, 855–856
  example model instantiation, 158–161
  ring-based access control, 531–533
  for secure applications, 655
Multifactor authentication, 446–448
Multilevel Directory (MLD), Trusted Solaris, 148–149
Multilevel Security (MLS) tool
  formal verification, 701–702
  HDM design verification package for, 707–708
  providing axions to theorem prover, 709–710
  SRI model embedded in, 707–708
Multilevel security models
  limit sharing for malware defense, 818
  shared resource matrix and covert channels, 598–599
  SPM subsuming, 82
Multipartite viruses, 783–784
Multiple encryption, block ciphers, 375–377
Multiple parenting
  comparing expressive power of models, 90–92
  comparing simulation in models, 88–90
  in ESPM, 83–88
Multistage attacks, 960, 965
Mutation Engine (MtE) tool kit, 788
Mutual exclusion, deadlocks from, 202
Mutual Recognition Arrangement (MRA), 749

N

N previous password, password aging, 435
N-stage nonlinear feedback shift register (NLFSR) method, 372–373
N-version programming, as malware defense, 817
Name constraints extension, X.509 PKI certificates, 352
Name (or key) value, cookies, 489
Naming
    assigning user login, 474
    file and objects, 472–473
    implementing logging criteria and, 886–887
    imported file systems in Solaris Trusted Extensions, 149–150
    improper, 1129–1131
    static and dynamic, 485–487
    user, 473–474
Naming and certificates
    conflicts, 479–481
    meaning of identity, 481–484
    overview of, 476–479
NAT. *See* Network Address Translation (NAT)
National Computer Security Center (NCSC), and TCSEC, 731, 736
National Scheme, CC evaluation methodology, 750
Natural deduction
    in predicate logic, 1185
    in propositional logic, 1180–1184
Naval Research Laboratory (NRL), Protocol Analyzer (NPA), 720–721
NCSC. *See* National Computer Security Center (NCSC)
Need for assurance, 629–631
Need to know rule, 143–144, 457–458
Needham-Schroeder protocol, 333–335, 337–339
Negation, connectives of propositional logic, 1179–1180
Network Address Translation (NAT), 1013
Network-based information gathering, agents, 942–943
Network File System (NFS), development system, 1051
Network File System version 2 (NFSv2) audit analysis of, 900–905 LAFS vs., 905–907
Network identification number (NID), DIDS, 950–951
Network Security Monitor (NSM), 949–952
Network security practicum
    analysis of infrastructure, 1013–1017
    anticipating attacks, 1027–1028
    availability, 1026
    cloud, 1024–1025
    DMZ DNS server, 1020
    DMZ log server, 1020–1021
    DMZ mail server, 1017–1018
    DMZ WWW server, 1018–1019
    general comment on assurance, 1025–1026
    internal network, 1021–1023
    introduction, 1005–1006
    overview of, 1011–1013
    policy development, 1006–1011
    review, 1028–1033
    wireless network, 1023–1024
Networks
    computer incident security response, 743
    and cryptography, 381–384
    layer security. *See* IPsec
    monitoring traffic in DIDS, 949–952
    monitoring traffic with NSM, 948–949
    system security practicum, 1042–1047
Neural nets, anomaly detection, 928–929
New nonargument (NNA) files, malware containment, 815–816
NewDES, modern symmetric cipher, 302
NFS. *See* Network File System (NFS)
NFSv2 (NFS version 2) protocol
    audit analysis of, 900–905
    LAFS vs., 905–907
NID. *See* network identification number (NID)
Nizza architecture, 591
NLFSR method. *See* n-stage nonlinear feedback shift register (NLFSR)
NNA files. *See* new nonargument (NNA) files
No preemption, deadlocks from, 202
Node creation operations, simulation in models, 88–89
Nodes, attack tree, 961–964
Noiseless covert channels, 595
Noisy covert channels, 595
Non-competition, electronic communications policy at UCD, 1217
Nonce
    Galois Counter Mode (GCM) of AEAD, 379
    Needham-Schroeder protocol, 334
Noncryptographic implementation mechanisms
    access control. *See* Access control mechanisms
    confinement. *See* Confinement principle
    design principles. *See* Design principles
    information flow. *See* Information flow
    representing identity. *See* Identity
Nondeductibility
    noninterference/policy composition and, 271–274
    and side channels, 280–282
Nonfunctional requirements, waterfall life cycle model, 639
Noninterference
    covert channels and, 596–598, 613–614
    policy model, 146–151
Noninterference, and policy composition
    deducibly secure systems, 273–274
    deterministic noninterference. *See* Deterministic noninterference
    generalized noninterference, 274–277
    nondeductibility, 271–273
    overview of, 255
    the problem, 255–258
    restrictiveness, 277–279
    review, 282–286
    side channels and deducibility, 280–282
Nonlattice information flow policies
    confinement flow model, 543–544
    nontransitive, 545–548
    overview of, 542–543
    transitive, 544–545
Nonmalicious computations, with botnets, 796
Nonrepudiation
    digital signatures providing, 318
    PEM design, 387
Nonsecure systems, auditing mechanisms for, 899–900
Nontransitive information flow policies, 545–548
Normal behavior (traces), anomaly detection, 923–924
Normal data tests, 1143
Notifier
    auditing system component, 883–884
    intrusion detection system architecture, 946–948
Notion of trust, malware defenses, 819–820

The Nozzle, 224
NP-complete problems, public key systems, 306–307
NPA. *See* NRL Protocol Analyzer (NPA)
NPA Temporal Requirements Language (NPATRL), 720–721
NPATRL. *See* NPA Temporal Requirements Language (NPATRL)
NRL. *See* Naval Research Laboratory (NRL)
NRL taxonomy, 857–859, 861–863
NSEC RR. *See* hostname resource record (NSEC RR)
NSM. *See* Network Security Monitor (NSM)
NSTISSP #11 policy, for IA and IA-enabled products, 728–729

O

Objects
  access control matrix model and, 32–37
  adding categories to security classifications for, 143–146
  Aggressive Chinese Wall Model and, 233–234
  basic results of determining system safety, 51–52
  Bell-LaPadula Model and, 142–143, 151–158
  Biba model for integrity policy, 175–178
  capabilities and, 518
  Chinese Wall Model, formal model, 230–233
  Chinese Wall Model, informal description, 228–230
  detecting covert channels via noninterference, 597–598
  DTEL associating types with, 122
  example model instantiation in Multics, 158–161
  identifying by assigning names, 472–473
  improper naming of, 1129–1131
  Lipner's security levels for, 180, 182
  principle of tranquility for security levels of, 161–163
  Propagated Access Control List (PACL), 533–534
  as protection types in SPM, 69
  Take-Grant Protection Model and. *See* Take-Grant Protection Model
  TCSEC reuse requirements, 731
  Trusted Solaris security classifications for, 146–151
Obligation, policy specifications in Ponder, 121
Oblivious transfer, cryptography, 358
Observation component, D-WARD, 218
OCaml functional programming language, 721
Occlusion principle, declassification, 163
OCSP. *See* Online Certificate Status Protocol (OCSP)
OFB mode. *See* output feedback (OFB) mode
Off-line dictionary attacks, 428–430, 440–441
Official Comment List, review process, 684–685
OFUNs, SPECIAL specification and, 703–704
On-line dictionary attacks, thwarting, 430–432
One-time pad
  meeting perfect secrecy requirement, 1168–1169
  as proven secure, 371
  simulating with LFSR method, 372
  as substitution cipher, 299
  weakness of, 299
One-time password authentication, 436–438
One-time pre-key pair OPK, instant messaging, 390–392
Onfusion, in cryptosystems, 290
Onion proxy, Tor, 497–499
Onion routing, anonymity on web, 494–498
Online Certificate Status Protocol (OCSP), X.509 PKI online revocation, 359
Online revocation system, X.509 PKI, 359
Open design. *See* Principle of open design
Open files, interception of requests to, 816
Open Source Security Testing Methodology Manual (OSSTMM), 834–835
OpenPGP, 348–350, 388–389
Operand or operation, improper choice of, 853–854, 1139–1141
Operation
  assurance during system, 695–696
  waterfall life cycle model, 641
Operation symbol node, covert flow trees, 603
Operational assurance
  defined, 634
  need for, 630
  overview of, 695–696
  TCSEC functional requirements, 732
Operational (IO) integrity classification, Lipner, 181–182
Operational issues
  computer security and, 16
  cost-benefit analysis, 16–17
  laws and customs, 19–20
  risk analysis, 17–18
Operations
  commercial integrity policies, 173–174
  improper choice of operand or, 853–854, 1139–1141
  improper sequencing of, 1139
Operator user account, development system backups, 1051–1052
Optical devices, fingerprint biometrics, 442
Or symbol node, covert flow trees, 603
ORCON. *See* Originator controlled access control (ORCON or ORGCON)
Organizational certificates, issuing, 478
Organizational problems
  operational controls and, 20–21
  security life cycle and, 22–24
Organizational security policies (OSPs), 752
ORGCON. *See* Originator controlled access control (ORCON or ORGCON)
Origin integrity
  as authentication. *See* Authentication
  cryptography providing, 290
  defined, 5
  DNSSEC providing, 488
  nature of security policies, 110–111
Originator controlled access control (ORCON or ORGCON)
  PACL implementation for, 533–534
  security policy, 118
Originator controlled access control (ORCON or ORGCON)
  digital rights management, 241–244
  further reading, 251
  overview of, 239–241
  PACL implementation for, 533–534
  security policy, 118
OSPs. *See* organizational security policies (OSPs)
OSSTMM. *See* Open Source Security Testing Methodology Manual (OSSTMM)
Otway-Rees protocol, 335–336, 369–370
Output
  conception stage of life cycle, 636
  feedback mode, 372
  manufacturing stage of life cycle, 637
  parameters, 549–550
Output feedback (OFB) mode, DES, 302

Outsiders
  problems from, 21
  threats to security from, 650
Outsiders class, policy development, 1008–1010
Overt (documented or known) purpose, Trojan horses, 776, 781
OVFUNs, 703–704, 708
Own right, access control matrix, 33, 42–43
Ownership, allowable use and, 1216–1217

P

PA model. *See* Protection Analysis (PA) model
Packets, IP header marking and, 981–983
PacketScore, 224
PACL. *See* Propagated Access Control List (PACL)
Padding Oracle On Downgraded Legacy Encryption (POODLE) attack, SSL, 401–402
Paging, virtual machines and, 1175–1176
Pakistani (or Brain) virus, IBM PC, 782, 783
PAM. *See* pluggable authentication modules (PAM)
Panorama, analyzing suspected malware, 810–811
Parameters
  passing information into/out of procedures via, 549–550
  validation flaws in RISOS study, 849–850
Parent component, internal design, 669–670
Parenting
  comparing expressive power of models, 91–92
  ESPM multiple, 83–88
Partial ordering, lattices, 1153–1154
Pascal programming language, Gypsy based on, 711
Pass algorithms, challenge-response authentication, 438–439
Passive side channel attack, deducibility, 280
Passive wiretapping, 7
Passively monitoring attack, containment in intrusion handling, 975–976
Passphrases
  used with passwords, 424–425
  voice recognition systems, 443
Passports, as certificates/assurance of trust, 483
Password space, defined, 416–417
Passwords
  aging, 434–438
  attacks on, 426–434
  authentication via, 416–418, 1053–1055
  challenge response authentication for, 438–441
  graphical, 425–426
  improper deallocation or deletion of, 1131–1132
  meters for password strength, 434
  in multifactor authentication, 446–448
  one-time, 436–438
  passphrases used with, 424–425
  password wallet or password manager, 425
  policy enforcing restrictions on new, 422–424
  principle of open design and, 461
  problem of sharing. *See* Program security practicum
  pronounceable, 420–421
  random, 418–420
  strength, 432–434
  user-created, 421–425
  user security and, 1074–1076
  writing down, 419–420
Patching of bugs, 638
Path value, cookies, 489
PATH variable, penetration testing in UNIX, 842–843
Paths, testing ordering of, 1143
Patients, Clinical Information Systems security policy, 236–239
Pattern-directed protection evaluation, Protection Analysis (PA) model, 851
Patterns of usage, audit reviews of, 880
PC CYBORG ransomware, 800–801
PC category. *See* Production Code (PC) category
PCAs. *See* policy certification authorities (PCAs)
PCC. *See* proof-carrying code (PCC)
PD category. *See* Production Data (PD) category
PD class. *See* public data (PD) class
Peer-to-peer botnets, 794
Peer-to-peer Overnet protocol, 794
PeerTrust recommendation system, 196
Pegasus spyware, iPhones, 799–800
PEM, *See* Privacy-enhanced Electronic Mail (PEM)
Penetration analysis theory, Gupta and Gligor, 868–873
Penetration studies
  conclusion, 845
  debate on validity of, 844–845
  Flaw Hypothesis Methodology, 830–833
  Flaw Hypothesis Methodology, versions, 833–837
  goals, 827–828
  layering of tests, 828–829
  methodology at each layer, 829
  overview of, 827
Penetration studies examples
  compromise of Burroughs system, 839–840
  penetration of corporate computer system, 840–841
  penetration of Michigan Terminal System, 837–839
  penetration of UNIX system, 841–843
  penetration of Windows system, 843–844
Penetration testing
  formal verification vs., 826
  using attack graphs to guide, 970–971
Penetration Testing Execution Standard (PTES), 836
People problems
  operational controls and, 21–22
  security life cycle and, 22–24
Perceptron neural network, anomaly detection, 928
Perfect secrecy, 1168–1169
Performance
  intrusion detection using system calls, 972
  paging, virtual machines, and loss of, 1176
Permissions
  access control file, 1120–1121
  on file creation, 1081
  security-related programming problems. *See* Program security practicum, common programming problems
Permitted set (PS) privileges, Trusted Solaris, 525
Permutation table, main DES algorithm, 1193
Personal conduct, acceptable use policy for, 1209–1212
Personal health information, Clinical Information Systems security policy, 236–239
Personal information, privacy/confidentiality policy, 1223
Personal use
  allowable use policy, 1219
  unacceptable conduct policy, 1212
Petri nets, IDIOT system, 933–934
PGP. *See* Pretty Good Privacy (PGP)

PGWG. *See* Process Action Team Guidance Working Group (PGWG)
Phases
  GISTA, 835–836
  OSSTMM modules, 835
  PTES, 836–837
Phenomes, voice recognition systems, 443
Phishing, impersonating legitimate entity, 802–803
Phonemes, pronounceable passwords based on, 420–421
Physical class, OSSTMM, 834
Physical resources, virtual machines and, 1175
PHYSSEC class, OSSTMM, 834
Pigeonhole principle, cryptographic checksums, 316
PKI. *See* public key infrastructure (PKI)
Plagiarism, as unacceptable conduct, 1209
Plaintext
  block ciphers and, 374–377
  ciphertext messages security issues, 367–370
  perfect secrecy and, 1168–1169
  self-synchronous stream ciphers, 373–374
  substitution ciphers changing characters in, 292–294
  transposition ciphers rearranging characters in, 291–292
Planning phase, GISTA, 835
Plans, manufacturing stage of life cycle, 637
Platform as a service cloud, 1024
PlayReady DRM, Microsoft, 243–244
Pluggable authentication modules (PAM), UNIX, 447–448, 1053, 1054
Pluggable transports, Tor, 499
Pointers, specifying confinement for compiler, 592
Policies. *See also* Noninterference, and policy composition
  assurance, throughout life cycle, 632–633
  availability policies. *See* Availability policies
  computer security incident response team, 986
  confidentiality policies. *See* Confidentiality policies
  high-level, 119–125
  how certificates encode, 477–478
  hybrid policies. *See* Hybrid policies
  information flow, 539–540
  integrity policies. *See* Integrity policies
  low-level, 125–126
  models of information flow, 541–542
  nonlattice information flow, 542–548
  overview of, 118
  password, 422–423
  policy-based trust models, 191–192
  program distribution, 1146–1147
  security. *See* Security policy
  shared password, 1100–1101
  system security practicum, 1036–1041
  user security practicum, 1072–1073
Policy-based trust models, 191–194
Policy certification authorities (PCAs), 478–479, 481–484
Policy checkers, 906, 942–943
Policy development practicum
  availability, 1010
  consistency check, 1010–1011
  data classes, 1007–1008
  overview of, 1006–1007
  user classes, 1008–1010
Policy models, 113, 632
Policy script interpreter, Bro, 937
Political activity, unacceptable conduct policy, 1211–1212
Polyalphabetic cipher, 294
Polymorphic viruses, 787–789
Ponder
  expressing trust relationships via, 194
  as high-level policy language, 119–121
POODLE attack. *See* Padding Oracle On Downgraded Legacy Encryption (POODLE) attack
Poset, partial ordering and, 1153
Postconditions, vulnerability analysis, 826
Postdevelopment verification technique, 700
Posting and authority to change, electronic communications policy, 1233–1234
Power, establishing clear chains of responsibility and, 20–21
PP-Configuration, CC, 753
PP-Module, CC, 753
PP. *See* protection profiles (PP)
PP Reference, CC, 753
Practice, of digital forensics, 990–994
Pre-key bundle, instant messaging, 391
Precedence rules, Control Tree Logic, 1186
Precision
  degree of overrestrictiveness, 134
  security mechanisms and, 12
  security policies and, 9–10, 131–136
Precomputing off-line dictionary attacks, 428
Precomputing possible messages, ciphertext problems, 367–368
Preconditions, vulnerability analysis, 826
Predevelopment verification technique, 700
Predicate logic (predicate calculus or first order logic), 1184–1185
Predicate subtypes, PVS language, 714
Preliminary technical review (PTR), TCSEC, 734
Preludes, as predefined theories in PVS, 714
Premaster secret, TLS handshake protocol, 398
Pretty Good Privacy (PGP)
  certificate revocation, 359
  certificate signature chains, 348–350
  certificates and assurance of trust, 483–484
  creation of, 385–386
  OpenPGP and PEM, 388–389
Prevention mechanisms
  deadlock, 202–203
  as goal of security, 10
  and integrity, 5
PRF function, TLS, 394–395
Primitive commands, 38–41
Primitive flow generator, penetration analysis tool, 872–873
Primitive inference rules, PVS proof checker, 715–716
Primitive operations
  basic results of, 51–52
  as mono-operational, 51
  TAM, 92–93
Primitive rights, determining system safety, 50, 52–56
Principals
  assigning rights to groups, 475
  authentication policy defines proof of identity for, 481–484
  certificates bind cryptographic keys to, 476–478
  in digital rights management, 242
  floating identifiers assigned to, 486
  host identity on web identifying, 485
  as unique entities specified by identity, 471
Principle, Locard's Exchange, 987
Principle of attenuation of privilege, 42–44, 74

Principle of complete mediation
  access to developer systems, 1040
  development system, 1060
  direct login to sysadmin account, 1050
  improper choice of initial protection domain, 1120
  inner firewall meeting, 1014
  overview of, 460
  restricting caching, 460–461
  systems not enforcing, 1056
  WWW server system in DMZ, 1066
Principle of conservativity, declassification, 163
Principle of economy of mechanism
  access control file permissions, 1121
  autonomous agents and, 953
  configuration of outer firewall, 1015
  development system, 1045–1046
  overview of, 459–460
  refinement of program security, 1113
Principle of fail-safe defaults
  design access to roles/commands, 1107
  overview of, 458–459
  principle of least astonishment vs., 1066
  storage of access control data, 1108–1109
  validation should apply principle of, 1135
Principle of least astonishment
  development system, 1045, 1047, 1063
  overview of, 464–465
  overwriting files, 1089
  principle of fail-safe defaults vs., 1066
  refinement and implementation, 1113, 1116–1117
  storage of access control data, 1108, 1110
  testing program, 1145–1146
Principle of least authority, 458
Principle of least common mechanism
  analysis of network infrastructure, 1014
  development system, 1064
  hiding information among modules, 1051
  memory programming problems, 1127
  network services, 1047
  overview of, 463–464
  users for system security, 1052
Principle of least privilege
  cell phones in violation of, 568
  configuring memory to enforce, 1122
  configuring outer firewalls, 1014–1015
  confinement enforcing, 581
  containment of internal addresses, 1014
  data classes in policy development, 1007–1008
  for improper choice of initial protection domain, 1118–1119
  for malware containment, 813–816
  overview of, 457–458
  processes run on development system, 1059–1060
  processes run on DMZ WWW server, 1055
  processes with fine-grained restrictions, 524
  Trusted Solaris privilege sets, 525–526
  user classes in policy development, 1008–1010
Principle of monotonicity of release, declassification, 163
Principle of occlusion, declassification, 163
Principle of open design
  data classes in policy development, 1007–1008
  development system, 1064
  distribution of program, 1146
  minimize secrets, 462
  overview of, 461–462
  user classes in policy development, 1008–1010
Principle of psychological acceptability, 465–466
Principle of semantic consistency, declassification, 163
Principle of separation of duty
  access control file permissions, 1120
  commercial integrity policies, 174, 183–184
  overview of, 463
  policy-based trust models, 193–194
  RBAC modeling, 246
Principle of separation of privilege
  configuring outer firewall, 1014
  data classes in policy development, 1007–1008
  overview of, 463
  user classes in policy development, 1008–1010
  via threshold scheme, 530
Principle of tranquility, security levels, 161–163
Principles
  access to medical records, 237–238
  composition policy, 257–258
  declassification, 163–164
  intrusion detection, 917–918
  record creation and information deletion, 238–239
Printf function, checking input from untrusted sources, 1136–1137
Privacy
  Android cell phones and, 568–570
  assurance of, 630
  Clinical Information Systems security policy, 236–239
  and confidentiality, 114, 1220–1225
  electronic communications policy and, 128–129, 1208–1209, 1220–1225
  expectations, 1235–1236
  limits, 1237–1239
  protections, 1236–1237
  right to anonymity, 500–501
Privacy Act of the United States, 114, 141
Privacy-enhanced Electronic Mail (PEM)
  basic design, 386–387
  certificate conflicts, 479–481
  certificates and assurance of trust, 482–483
  creation of, 385
  design principles, 385–386
  OpenPGP and, 388–389
Privacy Research Group, 385
Private functions, SPECIAL specification, 703–704
Private key, public key cryptography
  defined, 306
  infrastructures. *See* Key infrastructures
  public key cryptographic key exchange, 338–341
Privileges
  amplifying for capabilities, 521
  improper choice of initial protection domain and process, 1118–1120
  overriding restrictions on access via, 524–526
  principle of attenuation of privilege, 42–44, 74
  principle of fail-safe defaults, 458–459
  principle of least privilege, 457–458
  principle of separation of privilege, 463
  programming problems. *See* Program security practicum, common programming problems
  RBAC managing assignment of, 248
  virtual machines and, 583–584, 1172–1175
Proactive password checking, 422–424

Proactive password selection, 421
Probabilistic packet selection, IP headers, 981
Problem sources, in computer systems, 630
Procedure calls, information flow and, 556–557
Procedure segments, ring-based access control, 531–532
Procedures, electronic communications policy, 127
Process Action Team Guidance Working Group (PGWG), security testing, 689–695
Processes
  configuration of development system, 1059–1061
  configuration of DMZ WWW server, 1055–1061
  confinement problem, 580–582
  isolation of. *See* Isolation
  limitations of TCSEC, 736
  SSE-CMM analysis of existing, 765–768
  system security practicum, 1055–1061
  three sets of privileges in, 524
  trusted UNIX, 815
Processes, user security
  accidentally overwriting files, 1088–1089
  copying and moving files, 1087–1088
  encryption, cryptographic keys, and passwords, 1089–1090
  limiting privileges, 1091
  malicious logic, 1091–1092
  startup settings, 1090–1091
Processor status longword (PSL), virtual machines, 1172–1174
Product backlog, Scrum, 643
Product cipher, DES as, 300
Production Code (PC), Lipner, 179–180
Production Data (PD), Lipner, 179–180
Production, deployment stage of life cycle, 637
Production (IP) entities, Lipner, 181–182
Products
  formally verified, 722–723
  retirement of, 638
  TCSEC documentation requirements, 733
  tools for generation of, 687
Profile registry, Federal Criteria, 745
Program
  distribution, 1146–1147
  memory protection, 1122
  testing, 1145–1146
Program modification
  compiling, 592–593
  loading, 593–594
  overview of, 590
  rewriting, 590–591
  sandboxes using, 589–590
Program security practicum
  designing access to roles and commands, 1106–1110
  designing framework, 1104–1105
  distribution, 1146–1147
  overview of, 1099
  password management problem, 1099–1100
  refinement and implementation, 1111–1117
  requirements and policy, 1100–1103
  review, 1147–1150
  testing, maintenance, and operation, 1141–1146
Program security practicum, common programming problems
  improper change over time, 1125–1129
  improper choice of initial protection domain, 1118–1123
  improper choice of operand or operation, 1139–1141
  improper deallocation or deletion, 1131–1132
  improper indivisibility, 1138–1139
  improper isolation of implementation detail, 1123–1125
  improper naming, 1129–1131
  improper validation, 1132–1138
  overview, 1117–1118
Program statements
  assignment statements, 551
  certified information flow policy and, 548
  classification of, 550–551
  compound statements, 551–552
  conditional statements, 552–553
  goto statements, 554–556
  iterative statements, 553–554
  procedure calls, 556–557
Programming languages
  choice affects assurance of implementation, 685–686
  formal specification. *See* Formal specifications
  functional, 721
  information flow control in, 575
  type-safe, 592
Programming rules
  implementation, 1247–1248
  management, 1249
Pronounceable passwords, 420–421
Proof
  by contradiction, 1183
  of correctness, 15, 318
  logical systems used in formal proof technologies. *See* Symbolic logic
  PVS based on constructing/writing proofs, 713–716
  rules, 1180
  theory, SMV, 718–720
  using truth tables in natural deduction, 1182–1183
Proof-based verification techniques, vs. model-based, 700
Proof-carrying code (PCC), malware defense, 818–819
Proof checkers
  overview of, 681–682
  Prototype Verification System (PVS), 713, 715–716
Proof of concept, conception stage of life cycle, 636
Propagated Access Control List (PACL), 533–534
Propagating (or replicating) Trojan horse, 779–780
Propagation phase, computer worms, 791–792
Properties
  ATAM vs. TAM, 99–101
  cloud, 1024–1025
  comparing in access control models, 95–99
  mapping system to existing model for policy definition, 659–660
  nature of security policies, 111
  SMV proof theory for, 718–720
Property-based testing, detecting vulnerabilities, 826
Property specification verification, 700
Propositional logic (propositional calculus), 1179–1180
Protected memory, 519
Protection Analysis (PA) model
  analysis procedure, 854–856
  fingerd buffer overflow flaw, 862–863
  flaw classes, 852–854
  legacy, 856
  overview of, 851–852

  programming problems. *See* Program security practicum, common programming problems
  xterm log file flaw, 860–861
Protection domain, restricting for role processes, 1125
Protection flaws, improper, 852–853
Protection mechanisms, 132–135
Protection profiles (PP)
  CC, 751–753
  evaluating security target against, 754–756
  Federal Criteria, 744–745
Protection rings, 531
Protection state
  access control matrix model of, 32–37
  describing via access control matrix, 31–32
  determining if system is safe, 50–51
  results of determining system safety, 51–56
  Take-Grant Protection Model of. *See* Take-Grant Protection Model
  transitions, 37–41
  Typed Access Matrix Model (TAM), 92–94
Protection type, Schematic Protection Model, 69
Protocol Analyzer (NPA), NRL, 720–721
Protocol verifiers, 703
Protocols
  affect on security of cryptosystems, 367–370
  CAPSL specification, 721
  electronic mail (PEM and OpenPGP), 384–389
  formal methods for analyzing cryptographic, 702
  key management for instant messaging, 390–393
  networks and cryptographic, 381–384
  NPA cryptographic protocol verification, 720–721
  symmetric key exchange/authentication, 333–336
Prototype Verification System (PVS)
  as current verification system, 713
  experience with, 716
  proof checker, 715–716
  specification language, 713–715
Prototyping model, software development, 644
Proxy, multiple parenting in ESPM, 83–88
Proxy (or application level) firewalls
  blocking Java applets with, 978
  configuration of outer firewall, 1015
  overview of, 571–572
PS privileges. *See* permitted set (PS) privileges
Pseudo-anonymous (or pseudonymous) remailers, 491
Pseudocode, 1111–1114
Pseudonymizing sanitizers, auditing system design, 889–891
Pseudonyms, electronic communications policy at UCD, 1218
Pseudorandom numbers, key generation, 341–342
PSL. *See* processor status longword (PSL)
PTES. *See* Penetration Testing Execution Standard (PTES)
PTR. *See* preliminary technical review (PTR)
Public data (PD) class, policy development, 1007–1008
Public key cryptography
  El Gamal, 307–309
  elliptic curve ciphers, 312–315
  key exchange, 338–341
  overview of, 306–307
  RSA, 309–312
  TLS, 394
Public key digital signatures
  El Gamal, 321–322
  RSA, 318–321
Public key infrastructure (PKI), 350–353
Public key resource record (DNSKEY RR), DNSSEC, 488
Public records, electronic communication policy, 1224
Publication mission, CSIRT, 986
Pulsing denial-of-service attack, 221–222
Pumps, mitigation of covert channels via, 617–619
Purge function, noninterference security, 261–263
Purpose, electronic communications policy, 1213
PVS. *See* Prototype Verification System (PVS)

Q

Quality of service, availability policy ensuring, 201
Quasi-identifiers, auditing system design, 890

R

RA. *See* Registration authority (RA)
Rabbits, exhausting resources, 796
Race conditions
  detecting in file accesses, 817, 1128–1129
  improper indivisibility of, 1138
RACF, security enhancement package for IBM, 881
RAMP. *See* Ratings Maintenance Program (RAMP)
Random data tests, testing modules, 1145
Random numbers, key generation and, 341–342
Random (or pseudorandom) number generator, 419
Random (or pseudorandom) passwords, 436–438
Random password selection, 418–420
Random variable, 1163, 1165–1166
Randomness
  El Gamal cryptosystem, 309
  key generation and, 341–343
RANSOM-A ransomware, 801
Ransomware, 800–801
Rapid prototyping, Extreme Programming, 643
Rapid system iterations, exploratory programming model, 644
Rate-limiting component, D-WARD, 218
Rated product, TCSEC, 731
Rating Profile, SSE-CMM, 767–768
Ratings Maintenance Program (RAMP), 735, 737
Ratings, TCSEC trust management, 731
Rationale, CC, 752
Raw error patterns, Protection Analysis (PA), 854
RBAC. *See* role-based access control (RBAC)
RC. *See* control rights (RC)
Rdist UNIX program, 939–941
Re-call based systems, graphical passwords, 425
Read right, access control matrix, 33
Readable object set, detecting covert channels, 597–598
Real UID, 474
Reallocation transition, 211–212
Recognition-based systems, graphical passwords, 425–426
Recognition branch, covert flow tree, 605–606
Recognition goal, covert flow tree, 605–606
Recommendation systems, reputation-based trust models, 194–196
Recommender trust (RT), trust value semantics, 195
RECON guard, information flow control, 655
Record layer, TLS, 396
Recovery, 10–11, 1227
Red team attack. *See* Penetration studies
REDOC-II, modern symmetric cipher, 302

Reduced-round AES, 305
Reductio ad absurdum rules, natural deduction, 1181–1182
REFEREE trust model, 198
Reference monitor
    building system with security, 654
    defined, 654
    heavily influencing TCSEC approach, 730
Reference validation, 662
Reference validation mechanism (RVM), 654, 730
Refinement, access control module, 1111–1114
Reflector attacks, 221
Refrain, policy specifications in Ponder, 120–121
Registration authority (RA)
    CA delegates certificate requirements to, 477
    certificates and assurance of trust, 484
    certificates in X.509 PKI, 351
Registration, instant messaging, 391
Regular fixes, 695–696
Relational database browsing, audit browsing, 908
Relations, describing properties of, 1153–1155
Relationships, digital rights management, 242
Relay commands, Tor, 497–499
Reliability
    need for assurance of, 630
    paging, virtual machines and loss of, 1176
Religious activity, acceptable use policy, 1211–1212
Remote shell (*rsh*) connection spoofing attack
    JIGSAW representation of, 967–969
    overview of, 966–967
    using attack graphs, 970–971
Remove rule, Take-Grant Protection Model, 57
Repetitions
    attacks on Vigenére cipher and, 294–299
    iterative statements/information flow, 553
Replay attacks
    Kerberos clock synchronization and, 338
    symmetric key exchange vulnerability, 333–336
    on voice recognition systems, 443
Replay technique, audit browsing, 908–909
Reporting phase, GISTA, 836
Repository, program distribution, 1146
Representation
    correspondence, justifying design meets requirements, 677–680
    electronic communications policy at UCD, 1218
Repudiation of origin, as form of deception, 8
Reputation-based trust models, 194–196
Requirements
    access to role accounts, 1100–1103
    CISR, 743
    commercial integrity policies, 173–174, 187–188
    Common Criteria (CC), 752, 756
    conception stage of life cycle, 636
    design phase feeds back into, 1105
    Federal Criteria, 745
    FIPS 140, 746–747
    formal evaluation methodology, 728
    levels of abstract machines in HDM, 705–706
    mapping security functions to security, 666
    policy definition and specification of, 657–660
    role in assurance, 631–632
    TCSEC functional, 731–732
    testing begins with, 1142
    waterfall life cycle model, 639–640
Requirements, definition and analysis
    architecture, 651–657
    justifying requirements, 660–662
    policy definition/requirements specification, 657–660
    system assurance, 649
    threats and security objectives, 650–651
Requirements, justifying that design meets
    formal methods/proof techniques, 681–682
    informal arguments, 680–681
    overview of, 677
    requirements tracing/informal correspondence, 677–680
    review, 682–685
Requirements tracing, 677–680
Research Into Secure Operating Systems. *See* RISOS (Research Into Secure Operating Systems) study
Research Into Secure Operating Systems (RISOS) study
    fingerd buffer overflow flaw, 863, 864
    flaw classes, 849–851
    legacy of, 851
    overview of, 849
    xterm log file flaw, 861
Residential certificates, issuing, 478
Resource allocation system model, 210–215
Resource monitor, 213
Resource Public Key Infrastructure (RPKI), 361
Resource records (RRs), DNS and DNSSEC, 488
Resources
    constraint-based DoS model and, 209
    exhausted by malware, 796
    integrity of electronic communications, 1210
    leakage of information from shared, 581
    mitigation of covert channels via obfuscation, 616–617
    preserving confidentiality via hiding, 4
    problems from lack of, 21
    SYN flooding consuming, 216, 1026
    UIDs and exhaustion of, 1124
    virtual machines and physical, 1175
Responsibilities
    electronic communications policy at UCD, 1208, 1215
    establishing clear chains of, 20–21
    of users, 1234–1235
Restrictions
    as design principle, 455–457
    electronic communications policy at UCD, 1217–1218
    malware defense using specifications as, 817
    noninterference and policy composition, 277–279
    shared password requirements, 1100–1101
Retention, policy for, 129, 1227
Retina, eye biometrics, 443–444
Retirement, of DES, 302
Retrospective, for system security practicum, 1066–1068
Return oriented programming (ROP) flaw, UNIX, 848
Return-to-libc attacks, 848, 974–975
Returns control, Data Mark Machine, 563
Reusable components, system assembly from, 645
Reverse name lookup, DNS, 487
Review process, design meets requirements, 682–685
Revocation
    key, 358–359
    of rights, capability systems, 522
    rule, HRU vs. SPM, 82
Rewriting, program modification via, 590–591

Rights
  acceptable use policy for, 1208
  access control model entries as, 32–37
  capabilities and. *See* Capabilities
  delegation policy specifications in Ponder, 119–120
  determining system safety, 49–56
  formal model of Bell-LaPadula Model, 151–158
  Multics rules for, 158–161
  multiple parenting in ESPM, 83–88
  passing to other users, 269–270
  principle of least privilege, 457–458
  reducing to contain malware, 813–816
  SPM, 69–72
  SPM vs. HRU revocation rule, 82
  testing in TAM via ATAM, 99–101
Rights expression language, digital rights management, 243–244
Rights, Take-Grant Protection Model
  conspiracy, 66–68
  interpretation of, 61–63
  overview, 56–57
  sharing, 57–61
  theft and, 62–66
Rijndael, as AES, 303
Ring-based access control, 531–533
Ring compression, privilege and virtual machines, 1174
Ring policy, Biba Model, 177
Risk
  affecting level of trust in system, 730
  analysis, 17–18
RISOS study. *See* Research Into Secure Operating Systems (RISOS) study
Role account
  designing access to roles/commands, 1106–1110
  shared password problem, 1100–1101
  threats against, 1102–1103
  user interface design for access to, 1104–1105
Role-based access control (RBAC), 244–249
Role engineering, 248
Role mining, 249
Roles
  designing access to commands and, 1106–1110
  issuing certificates to principals as, 479
  keys can belong to, 354
  representing identity, 475–476
  role-based access control (RBAC), 244–249
  in second-level refinement to access control module, 1112–1114
ROM, for key storage, 354
Root key, instant messaging, 390–392
Rootkits, as pernicious Trojan horses, 777–779
ROP flaw. *See* return oriented programming (ROP) flaw
`RotWord` transformation, AES, 1202–1203
Round key
  AES, 1201–1203
  DES, 1191–1195
Round key schedule generation, AES, 1203
Rounds
  AES, 303–305, 1196
  DES, 300, 1191–1192
Routers, as filtering firewalls, 571
Routines
  access control record, 1115–1116
  error handling in reading/matching, 1116–1117
  obtaining location, 1114–1115
  refining high-level design to produce, 1111–1112
  in second-level refinement, 1112–1114
Routing, anonymity on web with onion, 494–498
RPKI. *See* Resource Public Key Infrastructure (RPKI)
RRs. *See* resource records (RRs)
RRSIG RR. *See* signature resource record (RRSIG RR)
RSA ciphers
  digital signatures, 319–321
  misordered blocks and, 368
  public key cryptography, 309–312
  as TLS interchange cipher, 394
  Yaksha security system based on, 357
RSA's SecurID system. phishing attack on, 803
RSA's SecurID system, phishing attack on, 803
RST packet, SYN flooding countermeasure, 217
RT. *See* recommender trust (RT)
Rule of transitive confinement, 581
Rules
  for access control by Boolean expressions, 35–36
  altered by adaptive directors, 946
  for Bell-LaPadula Model transformation, 155–158
  break-glass, 249
  Control Tree Logic (CTL), 1186
  of engagement, penetration studies, 828
  for Multics system rights, 158–161
  for multiple parenting in ESPM, 83–85
  natural deduction in predicate logic, 1185
  natural deduction in propositional logic, 1180–1182
  Network Security Monitor, 949
  programming, 1247–1249
  PVS proof checker, 715–716
  for Take-Grant Protection Model, 56–57, 61–63
  Trusted Solaris, 146–147
Rumpole's enforcement model, break-the-glass policy, 249–250
Running state
  denial of service protection base, 214
  model of resource allocation system, 210–211
Rust functional programming language, 721
RVM. *See* reference validation mechanism (RVM)

S

S-boxes
  AES, 304–305
  DES, 300–301, 1193–1194
S/Key system, one-time passwords, 437
SA. *See* Security association (SA)
SA database (SAD), IPsec, 405–409
SAD. *See* SA database (SAD)
Safe state, Banker's Algorithm, 203
Safety analysis
  of Augmented TAM (ATAM), 99–101
  comparing HRU and SPM, 82
  MTAM, 94
  SPM, 75–81
Safety, need for assurance of, 630
Salting, thwarting off-line dictionary attacks, 429–430
Sandboxes
  loading libraries for process confinement vs., 593
  malware attempts to evade detection in, 811
  malware containment via, 816–817
  process restriction using, 586–590

Sanitization, auditing system for log, 888–891
SAT system, detecting covert channels, 596–597
Saved set of privileges, processes, 524
Saved set (SS) privileges, Trusted Solaris, 525
Saved UID, 474
SCADA systems. *See* supervisory control and data acquisition (SCADA) systems, 582
Scalability, of formal verification methods, 733
Scanning
  as malware defense, 808–811
  network configuration for development system, 1046
Schematic Protection Model (SPM)
  of computer security, 68–69
  demand and create operations, 72–75
  ESPM, 83–88
  filter function, 70–71
  HRU vs., 82
  link predicate, 69–70
  putting it all together, 71–72
  results of SSR Protection Model, 68–69
  safety analysis, 75–81
  simulation and expressiveness of, 88–92
  Typed Access Matrix Model, 92–94
Schematic Send-Receive (SSR) Protection Model, 68
Schemes
  in access control models, 95–99
  comparing simulation in models, 89–90
  in digital rights management, 242
  as finite set of link predicates in SPM, 70
Science DMZ, 946
Scope
  electronic communications policy at UCD, 1213–1214
  limitations of TCSEC, 736
Scrum, Agile software development, 643
SD category. *See* System Development (SD) category
Second-level refinement, access control module, 1112–1114
Secret key cryptosystems. *See* Symmetric cryptosystems
Secret key digital signatures, 318
SECRET (S) security clearance, Bell-LaPadula Model, 142–146
Secrets
  minimizing in principle of open design, 462
  planning for compromised, 462
  sharing, 529–530
Secure communication mission, CSIRT, 986
Secure, definition of, 49–51
Secure field, cookies, 489
Secure Shell (SSH) protocol
  authentication for development system, 1054
  authentication for DMA WWW server, 1053–1054
  configuring inner firewall, 1016–1017
  network configuration for development system, 1045
  processes running on development system, 1060
  user configuration for DMZ WWW server, 1050
Secure Sockets Layer (SSL), 394, 400–402
Secure systems
  auditing mechanisms, 897–898
  basic security theorem and, 143, 145
  definition of, 109
Secure Xenix kernel, 602
Security
  assurance, 628–629
  CC protection profiles, 752–753
  electronic communications policy at UCD, 1225–1227, 1237–1239
  feasibility, in conception stage of life cycle, 636
  functional testing, 688–689
  kernels. *See* Kernels
  no longer exclusive realm of government/military, 730
  patches, 115–117
  principle of, 257–258
  problem definition, 752
  problems from overloaded administrators, 21
  specifications, 675–676
  structural testing, 688
  TCSEC domains, 734
  test suites, 689
  testing, 688–695
Security association (SA)
  AH protocol, 407–408
  bundle, 406
  ESP protocol, 408–409
  IPsec architecture, 404–407
  tunnel mode and transport mode, 406
Security classifications
  declassification problem, 162
  model instantiation in Multics, 161–163
  objects in Bell-LaPadula Model, 142
  principle of tranquility for security levels, 161–163
  Trusted Solaris, 146–151
Security clearance
  Bell-LaPadula and Chinese Wall Models, 234–236
  commercial vs. military integrity policies, 174
  formal model of Bell-LaPadula Model, 151–158
  forming security levels from, 143–144
  Lipner's full model, 181–182
  of subjects in Bell-LaPadula Model, 142
Security Features User's Guide (SFUG), TCSEC, 733
Security functional requirements (SFRs), CC, 752
Security functions
  design documentation, 665
  requirements tracing/informal correspondence, 677–680
  summary specification, design document, 665–666
Security gateway, IPsec, 403
Security levels
  in Bell-LaPadula and Chinese Wall Models, 234–236
  in Bell-LaPadula Model, formal model, 151–158
  change of access within, 143–146
  FIPS 140-2, 747–748
  in Lipner's full model, 181–182
  in Lipner's integrity matrix model, 178–180
  in Multics, 158–159
  principle of tranquility for, 161–163
  in Trusted Solaris, 148–151
Security life cycle, 22–24
Security mechanisms
  auditing, 897–900
  cost-benefit analysis of, 17
  design principles for. *See* Design principles
  laws and customs as constraints on, 19–20
  in layered architecture, 652–653
  protection state and, 31
  security and precision in, 131–135
  security policies vs., 9–10, 112–113
  supporting availability, 202
  TLS cryptographic, 394–396

Security models
  definition of, 632
  formal vs. informal, 700
Security-Oriented Analysis of Application Programs (SOAAP), 722–723
Security Parameters Index (SPI), 405–409
Security policy. *See also* Noninterference, and policy composition
  attacks violating, 959
  auditing to detect violations, 893–897
  definition of, 109, 631–632
  definition/requirements specification, 657–660
  determining safety of system, 49–51
  development of, 1006–1011
  example. *See* Academic computer security policy example
  firewalls, 570–571
  HRU vs. SPM, 82
  justifying requirements, 660–662
  languages, 118–126
  laws/customs as constraints on, 19–20
  malware causing violation of, 775–776
  nature of, 109–113
  penetration tests violate constraints stated in, 827–828
  protection state and, 31
  review, 136–139
  role of trust in, 115–117
  security and precision in, 131–136
  security mechanisms vs., 9–10
  as set of noninterference assertions, 262
  system security. *See* System security practicum
  types of, 7
  types of access control, 117–118
  using vulnerability to violate, 825
Security policy databases (SPDs), IPsec, 404–405
Security-preserving rules, Bell-LaPadula Model, 155
Security Requirements for Cryptographic Modules (IG), 748
Security specifications, 657, 675–677, 702
Security target (ST), CC, 751, 754–756
Security target (ST), ITSEC
  assurance requirements, 739
  defined, 738
  evaluation process, 741
  examples of informal arguments, 680–681
  justifying requirements, 660–662
  limitations of vendor-provided, 742
Security targets, vendor-provided, 742
Segment identifier, untrusted modules, 591
Segment matching, untrusted modules, 591
Segments, ring-based access control, 531–533
seL4 microkernel, as formally verified product, 722
Self-healing property, 374, 375–376
Self-issued certificate, X.509 PKI, 350
Self-organizing maps, anomaly detection, 928–930
Self-signed certificates, 349, 350
Semantic consistency principle, declassification, 163
Semantics, object name, 472–473
Semaphores, information flow using, 558–561
Sending instant messages, 392
Sendmail, penetration testing UNIX system, 841–842
Sensitive data
  Bell-LaPadula Model labels, 142
  consistency check in policy development, 1010–1011
  improper deletion of, 1131–1132
  Trusted Solaris labels, 146–151
  virtual machines and, 1172–1175
Separation of duty. *See* Principle of separation of duty
Separation of function, 174
Separation of privilege. *See* Principle of separation of privilege
Sequences of events, in specification-based detection, 938
Sequencing, improper, 1139
Servers, confinement problem, 579–582
Service providers, access control and, 579–580
Service specification, denial of service models, 208–210
Session keys
  Bellare-Rogaway protocol, 336
  interchange key vs., 332
  Kerberos protocol, 337–338
  Needham-Schroeder protocol, 333–335
  Otway-Rees protocol, 335–336
  public key exchange and authentication, 338–341
Session_id, TLS handshake protocol, 397
Sessions
  instant messaging setup, 391–392
  TLS, 393–394
Setuid programs, 474
SFRs. *See* security functional requirements (SFRs)
SFUG. *See* Security Features User's Guide (SFUG)
SHA-256-based password hashing, 1053
Shared resource matrix model, SDLC, 610
Shared resource matrix (SRM) methodology, 598–600
Sharing
  limiting for malware defense, 817–819
  limiting with least common mechanism, 463–464
  memory, 1121–1122, 1126–1128
  problems with password. *See* Program security practicum
  resources, 581, 594
  secrets, 529–530
Sharing rights, Take-Grant Protection Model
  conspiracy, 66–68
  overview of, 57–61
  theft, 62–66
Shift (Caesar) cipher, 289–291, 294
`ShiftRows` transformation, AES, 304, 1197–1199, 1203–1205
Shipping, deployment stage of life cycle, 637
Side channel attacks
  covert channels vs., 581
  and deducibility, 280–282
  defined, 280
  as form of covert channel, 582
Sidewinder firewall
  restricting access via type checking, 529
  as sandbox built into kernel, 587
Siemens systems, targeted by Stuxnet worm, 792
Signal handlers, improper operation of, 1141
Signal protocol, instant messaging, 390
Signature-based detection, incident prevention, 972
Signature block, scanning as malware defense, 808
Signature chains. *See* Certificate signature chains
Signature, malware, 809
Signature resource record (RRSIG RR), DNSSEC, 488
Signature_algorithm, TLS handshake protocol, 398
Signatures
  adding dynamically in IDIOT system, 934
  malware, 809–810
Signcryption, cryptographic primitive, 326

Signed pre-key pair SPK, instant messaging, 390–392
Simplicity, as design principle, 455–456
Simulation, ESPM vs. SPM, 88–90
Simultaneity policy, finite waiting time policy, 207–208
Simultaneous users, copied material and, 1209
Single flux botnet, 795
Single key cryptosystems. *See* Symmetric cryptosystems
Single-level directories (SLDs), Trusted Solaris, 148–149
Skipjack symmetric cipher, 355–356
SL security level. *See* System Low (SL) security level
SLDs. *See* single-level directories (SLDs)
Sleeping state
  denial of service protection base, 214
  resource allocation system, 210
Slicing technique, audit browsing, 909
Smallest bucket problem, pronounceable passwords, 420–421
Smart cards, for key storage, 354
Smart terminals, user security and, 1085–1086
SMTP
  anticipating attacks in network security, 1027
  configuring inner firewall, 1015–1016
  configuring outer firewall, 1015
  network configuration for development system, 1045–1046
Smurf attack, as amplification attack, 221
SMV. *See* Symbolic Model Verifier (SMV)
Snooping (eavesdropping), 7
SOAAP. *See* Security-Oriented Analysis of Application Programs (SOAAP)
Social engineering, 22
Software
  adware entering system via, 798–799
  design. *See* Design assurance, system/software
  development life cycle, 635–639
  fault isolation, 587, 591
Software as a service cloud, 1024
Software development models
  Agile, 641–644
  other, 644–645
  waterfall life cycle model, 640–641
Software Tools (T) category, Lipner, 179–180, 181
SOG-IS. *See* Senior Officials Group Information Systems Security (SOG-IS)
Sony, DRM implementation, 244
Soundness, of information flow rules, 561–562
Source code, identifying covert channels in, 601
SPDs. *See* security policy databases (SPDs)
Speaker recognition, biometrics authentication, 443
Speaker verification, biometrics authentication, 443
Spearphishing, tailored for particular victim, 802–803
SPECIAL formal specification language
  eliminated as specification language for EHDM, 710
  in formal verification example, 701–702
  MLS tool and, 707–708
  precise semantics of, 702–705
  strengths of, 703
Special topics
  attack and response. *See* Attack and response
  auditing. *See* Auditing
  intrusion detection. *See* Intrusion detection
  malicious logic. *See* Malware
  vulnerability analysis. *See* Vulnerability analysis
Specification-based detection, 920, 938–942
Specifications
  access control matrix and, 32
  assurance and, 13–14
  defined, 657, 702
  design satisfying, 14–15
  external interfaces, 666–668
  formal. *See* Formal specifications
  Gypsy for external/internal, 712
  implementation satisfying, 15–16
  internal design, 673–674
  modification, 675
  overview of, 658
  policy definition and requirements, 657–660
  PVS based on writing, 713–715
  requirements tracing/informal correspondence, 677–680
  as restrictions, in malware defense, 817
  security, 657, 675–677, 702
  security functions summary, 665–666
  security testing, 688–689, 693–695
  service, 208–210
SPECSEC class, OSSTMM, 834
SPI. *See* Security Parameters Index (SPI)
SPM. *See* Schematic Protection Model (SPM)
Spoofing attacks
  as deception and usurpation, 7
  DNSSEC immediately detecting, 488
  on facial recognition systems, 444
  on identity of host on web, 485
  remote shell connection, 966–971
Sprints, Scrum, 643
Spyware, 799–800
SRI model, Bell-LaPadula Model vs., 707–708
SRM. *See* shared resource matrix (SRM)
SS privileges. *See* saved set (SS) privileges
Ssc-preserving rules, Bell-LaPadula Model, 155–156
SSE-CMM. *See* System Security Engineering Capability Maturity Model (SSE-CMM)
SSH protocol. *See* Secure Shell (SSH) protocol
SSL. *See* Secure Sockets Layer (SSL)
SSR Protection Model. *See* Schematic Send-Receive (SSR) Protection Model
ST reference, 754
ST. *See* security target (ST)
Stacking, PAM modules, 448
Stages, of life cycle process, 635–639
Stand-alone technique, formal specification in, 703
Stand up meeting, Scrum, 643
Standards, examining vulnerabilities, 864–868
STAT, misuse intrusion detection, 934–937
State
  cookies and, 488–490
  system, 31
State-based auditing, 894
State-based denial of service model, 210–215
State machine model, 277–279
State-matching reductions, 97–101
State transitions
  affecting protection state, 32
  ATAM, 99–101
  comparing schemes/security properties, 95–98
  deterministic noninterference and, 259–261
  model of resource allocation system, 211–212
  security in terms of, 271–272
  unwinding theorem, 263–265

Stateful firewalls, 572
Statement of importance, electronic communications policy, 127
Statements. *See* Program statements
Static analysis, malware detection, 811
Static identifiers, on web, 485–487
Static intrusion detection models, 920
Static keystroke recognition, biometric authentication, 444
Static mechanisms, information flow
  assignment statements, 551
  compound statements, 551–552
  conditional statements, 552–553
  declarations, 549–550
  exceptions and infinite loops, 557–558
  goto statements, 554–556
  iterative statements, 553–554
  overview of, 548
  procedure calls, 556–557
  program statements overview, 550–551
Static rights, access control by history, 36–37
Statistical analysis, as malware defense, 819
Statistical methods, anomaly detection, 921–922
Statistical regularities, ciphertext problems, 368–369
Statistics, cryptosystem attack using, 291
Stealing, as theft in Take-Grant Protection Model, 63–66
Stealth virus, 786
Storage
  of access control data, 1108–1110
  in cloud, 1025
  covert channels constraining access to, 581
Storage, key
  key escrow, 354–355
  key escrow system and Clipper chip, 355–357
  other approaches to, 357–358
  overview of, 353–354
  Yaksha security system, 357
Stream ciphers
  block ciphers vs., 371
  generating random, infinitely long key, 370
  self-synchronous, 373–374
  synchronous, 371–373
Strength, password, 432–434
Strict conformance, CC methodology, 752
Strict integrity policy (Biba's model), 177–178
Strong hash function, 316
Strong mixing function, 342
Strong one-way hash function, 316
Strong tranquility principle, 162
Structural (white box) testing, 688
Structure, AES, 303–304
Structure, DES, 300
Structured protection, TCSEC, 734
Student information, electronic communication privacy, 1223–1224
Stuxnet worm, 792
`SubBytes` transformation, AES encryption, 304, 1197–1199, 1203–1205
Subcomponents, 663–664
Subgoals, of attacks, 960–966
Subject alternative name extension, X.509 PKI certificates, 352
Subject key identifier extension, X.509 PKI certificates, 351
Subjects
  access control matrix model and, 32–37
  adding categories to security classification of, 143–146
  Aggressive Chinese Wall Model, 233–234
  basic results of determining system safety, 51–52
  Biba model for integrity policy, 175–178
  capabilities and, 518
  Chinese Wall Model, formal model, 230–233
  Chinese Wall Model, informal description, 229–230
  DTEL associating domains with, 122–125
  example model instantiation in Multics, 158–161
  formal model of Bell-LaPadula Model, 151–158
  link predicate in SPM as relation between two, 69–70
  Lipner's full model of security/integrity levels for, 182
  Lipner's security levels for, 179–180
  principle of tranquility for security levels of, 161–163
  Propagated Access Control List (PACL), 533–534
  as protection types in SPM, 69
  security clearance in Bell-LaPadula Model, 142–143
  Take-Grant Protection Model. *See* Take-Grant Protection Model
  Trusted Solaris security classification/categories, 146–151
Subnets, wireless networks, 1023
Substitution ciphers
  one-time pad, 299
  overview of, 292–295
  Vigenére cipher, 295–299
Substitution, DES using, 300
Subsystem, defined, 663
Subtrace of trace, 938
`SubWord` transformation, AES round key generation, 1202–1203
Suitability analysis map, 660–662
Supervised machine learning methods, anomaly detection, 924
Supervisory control and data acquisition (SCADA) systems, 582
Surreptitious forwarding attack, 322
SVM. *See* support vector machine (SVM)
Symbolic logic
  overview of, 1179
  predicate logic, 1184–1185
  propositional logic, 1179–1184
  review exercises, 1188–1189
  temporal logic systems, 1186–1188
  used in formal proof technologies, 700
Symbolic Model Verifier (SMV)
  experience with, 720
  proof theory, 718–720
  specification language, 716–718
Symmetric cryptosystems
  AES, 303–306
  DES, 299–302
  key exchange, 333–336
  other modern symmetric ciphers, 302–303
  overview of, 291
  as single key or secret key, 291
  substitution ciphers, 292–299
  transposition ciphers, 291–292
Symmetric key exchange protocol, Diffie-Hellman as, 340
SYN/ACK packet
  availability and SYN flood attack, 215, 218–221
  remote shell (*rsh*) attack on, 966–967
SYN cookies, and flooding attacks, 219–220

SYN flood attack, availability and, 215–221, 1026
SYN packets
  availability during flooding attacks, 218–221
  SYN flooding countermeasures, 217
Synchronization
  coding faults in Aslam's model, 859
  flaws, 853–854
  one-time passwords and, 436–438
Synchronous stream ciphers, 371–374
Syntactic issues, auditing system design, 887–888
System administrator, authentication, 1054
System Development (SD) category, Lipner, 179–180
System Low (ISL) integrity classification, Lipner, 181–182
System Low (SL) security level, Lipner, 178–180, 182
System Program (ISP) integrity classification, Lipner, 181–182
System Security Engineering Capability Maturity Model (SSE-CMM), 628, 765–768
System security practicum
  authentication, 1053–1055
  files, 1061–1066
  introduction, 1035–1036
  networks, 1042–1047
  policy, 1036–1041
  processes, 1055–1061
  retrospective, 1066–1068
  review, 1068–1072
  users, 1048–1053
System trace, definition of, 938
Systems
  architecture, 728, 732
  assembly from reusable components, 645
  digital forensics for entire, 990–992, 998–999
  evaluation of. *See* Evaluation of systems
  intrusion detection, 972
  logs, designing auditing system, 891–893
  monitoring, 1225
  problems from administrators, 22
  testing, 640–641, 688
Systems, building with assurance
  documentation and specification, 675–677
  implementation. *See* Implementation assurance
  operation and maintenance, 695–696
  overview of, 673
  requirements definition. *See* Requirements, definition and analysis
  requirements design. *See* Requirements, justifying that design meets
  review, 696–698
  system/software design. *See* Design assurance, system/software

T

T (Software Tools) category, Lipner, 179–180, 181
T (tainted) lower level security model, Android, 568
Tags, protecting capabilities via, 519
Taint sinks, Android, 568
TaintDroid, Android, 568–570
Tainted (T) lower level security model, Android, 568
Take-Grant Protection Model
  of computer security, 56
  conspiracy in, 66–68
  controlling copying of capabilities, 523
  create rule, 57
  demand and create operations in SPM vs., 72–75
  formulating as instance of SPM, 72
  grant rule, 56
  interpretation of model, 61–63
  principle of least authority and, 458
  remove rule, 57
  review, 68
  schemes and security properties, 95–99
  sharing of rights, 57–61
  SPM subsuming, 82
  take rule, 56
  theft in, 62–66
Take rule, Grant-Protection Model
  conspiracy, 61–63
  demand and create operations, 72
  overview of, 56
  sharing of rights, 58–59
  theft, 64–66
TAM Model. *See* Type Access Matrix (TAM) Model
Target
  of attacks, 960
  moving target defense, 973
Target Corporation breach of 2013, 638
Target of evaluation (TOE), 738, 752–756
Target selection phase, computer worms, 791–792
Tautologies, 1180–1181
Tautology, natural deduction in propositional logic, 1180
Taxonomy, enforcing security via NRL, 857–859
TCB. *See* trusted computing base (TCB)
TCC. *See* type correctness condition (TCC)
TCP intercept mode, SYN flooding countermeasures, 217
TCP state, 218–221
TCP three-way handshake
  auditing to detect known violations of policy, 896–897
  availability and SYN flood attack, 215–216
  SYN flooding countermeasures, 217
TCP wrappers, configuring network development system, 1046
Tcp_wrappers program, transition-based auditing, 895
TCSEC. *See* Trusted Computer System Evaluation Criteria (TCSEC)
TDI. *See* Trusted Database Management System Interpretation (TDI)
Technical review board (TRB), TCSEC, 735
Technical review, design meets requirements, 683
Telecommunications class, OSSTMM, 834
Telephone conversations, privacy protection/limits, 1224
Temporal logic systems, types of, 1186
Terminate and stay resident (TSR) viruses, 785–786
Test assertions, security testing using PGWG, 690, 693
Test matrices, security testing using PGWG, 690–692
Testing
  for informal validation of design/implementation, 1142
  satisfying assurance via, 16
  security, 688–689
  TCSEC, 733
Text display technique, audit browsing, 908
TFM. *See* Trusted Facility Manual (TFM)
Theft, in Take-Grant Protection Model, 62–66
Theorem prover
  Bledsoe, 712–713
  Boyer-Moore, 709–710

  EHDM, 710–711
  PVS tightly integrated with, 713
Theorems
  formal mathematical specifications in BLP security policy, 702–705
  undecidability of virus detection, 803–808
Theories, PVS language and, 714
Theory of computer viruses, 803–807
Theory of penetration analysis, Gupta and Gligor, 868–873
Therac 25 computer-based electron accelerator radiation therapy, 630–631
Third party
  independent testing, 688
  spyware recording data for use by, 799–800
Threats
  from botnets, 793–796
  building trusted system, 650–651
  classes of, 7
  identifying in conception stage of life cycle, 636
  mapping to requirements, 661–662
  monitoring with audit trails, 879
  overview of, 6–9
  role accounts accessed by authorized users, 1102–1103
  Security Problem Definition, CC, 752
  vulnerabilities vs., 650
Three-key Triple DES mode, 377
Three Mile Island nuclear failure, 631
Threshold metrics, anomaly detection, 921
Threshold scheme
  one-time pad as, 299
  principle of separation of privilege via, 530
Thumbprinting, tracing attack through network, 980
Ticket-granting servers, Kerberos, 337–338
Tickets
  Kerberos, 337–338
  multiple parenting in ESPM, 83–88
  safety analysis of SPM, 75–81
  SPM, 69–75
Tiger team attack. *See* Penetration studies
TIM research system, Digital Equipment Corporation, 922–923
Time
  covert channels and, 581–582
  covert timing channels, 594
  of day, access control module, 1113
  hardware challenge-response procedures and, 439
  improper choice of operand/operation and, 1140–1141
  interpreting for key storage, 357–358
  of introduction, in NRL taxonomy flaws, 857
  mitigating covert channels via, 617
  in password aging, 435
  risk changing with, 18
  in temporal logic systems, 1186
Time-based inductive learning, anomaly detection, 922–923
Time-Based One-Time Password Algorithm (TOTP), 438
Time flaw attacks, Otway-Rees protocol vulnerability, 336
Time-of-check-to-time-of-use problem, race conditions, 1128–1129
Timestamps
  cryptographic key infrastructure and, 344
  detecting replay attacks, 335–336
Timestomp plug-in, 995
TLS protocol. *See* Transport Layer Security (TLS) protocol
TNI. *See* Trusted Network Interpretation (TNI)
TOE Security Functions (TSF), CC, 750
TOE Security Policy (TSP), CC, 750
TOE. *See* target of evaluation (TOE)
Token-subsequence signatures, worm detection, 810
Tokens
  hardware challenge-response procedures and, 439
  using cookies for authentication, 490
TOP SECRET (TS), Bell-LaPadula Model, 142–146
Topmost goal, covert flow trees, 605
Tor onion router, 497–499
Torpig botnet, 793–796
Total isolation, 580, 616
Total ordering, lattices, 1153–1154
TOTP. *See* Time-Based One-Time Password Algorithm (TOTP)
Toyota manufacturing, Kanban, 643
TPE. *See* Trident Polymorphic Engine (TPE)
TPs. *See* transformation procedures (TPs)
Traces
  of events, 938–939
  every contact leaves, 987–990
  left by anti-forensic tools, 996
Traditional scheme, salting, 429
Traffic analysis, as cryptanalysis, 383–384
Traffic-policing component, D-WARD, 218
Traffic Validation Architecture, 224
Training data
  anomaly detection using distance to neighbor, 931
  anomaly detection with Markov models, 924
  anomaly detection with neural nets, 928–929
  anomaly detection with self-organizing maps, 929–930
  in systems using clustering, 926
Tranquility principle, 161–164, 885
Transactions
  as basic operation. *See* Clark-Wilson integrity model
  integrity security policies, 114
  maintaining state to simplify, 488–490
  RBAC, 244–249
Transceivers, AAFID, 953
Transformation procedures, as sequences of state transitions, 38
Transformation procedures (TPs), Clark-Wilson integrity model, 184–188
Transformations, AES
  decryption, 1200–1201
  encryption, 1197–1199
  order for encryption vs. decryption, 1203–1205
  round key generation, 1201–1203
Transformations, cryptosystem, 290
Transitions
  auditing based on, 881–883, 895
  logging based on, 895
  protection state, 37–41
Transitive confinement, rule of, 581
Transitive nonlattice information flow policies, 544–545
Transitivity of trust, 189
Translucent cryptography, 358
Transmatrix procedure, 555–557
Transport adjacency, IPsec, 406
Transport Layer Security (TLS) protocol
  alert protocol, 399
  application data protocol, 400
  change cipher spec protocol, 399
  cryptographic mechanisms, 394–396

  handshake protocol, 397–399
  Heartbeat protocol extension, 399–400
  overview of, 393–394
  record layer, 396
  SSLv3 vs. TLSv1.2, 400–401
Transport layer (TLS and SSL) security
  application data protocol, 400
  Heartbeat protocol extension, 399–400
  overview of, 393–394
  problems with SSL, 401–402
  SSLv3 vs. TLSv1.2, 400–401
  supporting cryptographic mechanisms, 394–396
  TLS alert protocol, 399
  TLS change cipher spec protocol, 399
  TLS handshake protocol, 397–399
  TLS record protocol, 396
Transport mode, IPsec, 403
Transport mode SAs, IPsec, 406
Transposition cipher, 291
Transposition, DES using, 300
TRB. *See* technical review board (TRB)
Tree authentication scheme, Merkle, 344–345
Trees, attack, 961–964, 965–971
The Trial (Kafka), 500
Trident Polymorphic Engine (TPE), 788
Trident Vulnerabilities, Pegasus spyware, 799–800
Triple DES mode, 302
Tripwire
  low-level policy language, 125–126
  scanning as malware defense, 808
  static analysis and, 894
Trojan horse
  accessing role accounts, 1102–1103
  adware as, 797
  computer viruses as, 781
  overview of, 776–777
  propagating, 779–780
  ransomware as, 800–801
  rootkits as pernicious, 777–779
  spyware as, 799–800
  Stuxnet worm carried via, 792
  theft in Take-Grant Protection Model, 63
  triggering with logic bombs, 797
Trojan.Peacomm bot, 794
True positive (detection) rate, intrusion detection, 925
Trust
  assumptions and, 11–12
  assurance and, 12–13, 627–629
  attempted attacks within DMZ/misuse of, 1027
  Biba model integrity levels and, 175–178
  certificates and assurance of, 482–484
  confidentiality policies and, 114
  DNSSEC improving DNS, 488
  formal evaluation and, 728–729
  of host name in DNS database, 486
  information flow metrics containing malware, 813
  integrity policies and, 114–115
  ITSEC levels of, 731, 738
  malware defense and notion of, 819–820
  malware detection on data/information, 812
  meaning of identity in, 481–484
  no longer realm of government/military, 730
  problems with PKIs, 352–353
  propagation, 190
  reducing user rights for malware containment, 815–816
  role in computer security, 115–117
  security issues with DNS, 487
  in system, 1123
Trust anchor, CA certificate as, 351
Trust models
  integrity models vs., 189
  overview of, 189–191
  policy-based, 191–194
  reputation-based, 194–196
Trusted Computer System Evaluation Criteria (TCSEC)
  assurance requirements, 732–733
  Canadian efforts, 737–738
  CISR, 742–743
  defined, 727
  evaluation classes, 733–734
  evaluation process, 734–735
  Federal Criteria developed to replace, 744–745
  functional requirements, 731–732
  impacts, 735–737
  ITSEC evaluation vs., 741–742
  overview of, 730–731
  requirements in ITSEC not found in, 739–740
Trusted Database Management System Interpretation (TDI), TCSEC, 736
Trusted distribution, TCSEC assurance, 732
Trusted modules, software fault isolation for, 591
Trusted Network Interpretation (TNI), TCSEC, 736
Trusted path requirements, TCSEC, 732
Trusted, remote shell (*rsh*) attack and, 966–967
Trusted Solaris example
  directories and labels, 148–151
  limiting sharing for malware defense, 818
  privileges in, 525–526
  security classification and categories, 146–148
  weak tranquility in, 162–163
Trusted systems
  defined, 629
  development of, 632–634
  formal security evaluation in, 728–730
  scanning as malware defense, 808–809
Trusted systems, building secure and
  Agile software development, 641–644
  life cycle, 634–639
  other models of software development, 644–645
  waterfall life cycle model, 639–641
Trusted third party
  key escrow system, 354–355
  secret key digital signatures rely on, 318
  symmetric key exchange relies on, 333
Trusted users, Multics system, 158–161
Trustworthiness
  integrity as data or resource, 5–6
  methodologies assigning levels of, 629
  trust as measure of, 628
Truth tables, natural deduction, 1182–1183
TS. *See* TOP SECRET (TS)
TSF. *See* TOE Security Functions (TSF)
TSP. *See* TOE Security Policy (TSP)
TSR viruses. *See* terminate and stay resident (TSR) viruses
Tunnel mode, IPsec, 403
Tunnel mode SAs, IPsec, 406
Turing machine, 52–56, 804–808
Two-bit machine, 259–263, 272–273

Two-factor authentication, 446–447
Two-key Triple DES mode, 376
Two-level security model, Android, 568
Twofish cipher, 303
Type-1 hypervisor, 583, 1172–1173
Type-2 hypervisor, 583, 1172–1173
Type Access Matrix (TAM) Model
  ATAM, 99–101
  Dynamic TAM Model, 102
  MTAM, 93–94
  overview of, 92–93
Type checking
  designing for validation, 1137–1138
  improper validation and, 1134
  locks and keys access control, 527–528
Type flaw attacks, ciphertext problems, 369–370
Type-safe programming languages, implementing confinement, 592
Types
  CAPSL specification, 721
  DTEL based on, 121–125
  SPECIAL specification and, 703–705
  Typed Access Matrix Model, 92–94

U

UA. *See* user agent (UA)
UC Davis. *See* Academic computer security policy example
UC clearance. *See* UNCLASSIFED (UC) clearance
UDIs. *See* unconstrained data items (UDIs)
UID. *See* Unique Identifier for Device (UID)
UIDs. *See* User identification numbers (UIDs)
Unacceptable conduct, electronic communications policy, 1209–1212
Unauthorized access, electronic communications policy, 1210
Unauthorized (insecure) states, security policies, 109–113
Uncertainty. *See* Entropy and uncertainty
UNCLASSIFED (UC) clearance, Bell-LaPadula Model, 142–146
Unconditional commands, TAM, 93
Unconstrained data items (UDIs), Clark-Wilson integrity model, 184–186
Unique objects, require unique names, 1130
Unit testing, 640, 688–689
United Kingdom IT Security Evaluation and Certification Scheme Certification Body, 738
UNIVAC 1108 computer, virus development, 782
Universal security analysis instance, 96, 98
UNIX systems
  authentication for DMA WWW server, 1053
  built-in security in vs. adding later, 656–657
  Clark-Wilson model implementation, 186–187
  Internet worm, 790–791
  known security flaws, 846–848
  malware detection on data, 812
  multifactor authentication, 447–448
  opening files, 518
  password mechanism, 417–418
  penetrating, 841–843
  privileges, 524–526
  process configuration for system security, 1056–1057
  representing accounts by UID, 1049
  specification-based intrusion detection, 939–941
  spread of viruses, 782
  trusted processes, 815
  type checking, 528
  types of file names, 472–473
  user identity, 474
Unsafe instruction, untrusted modules, 591
Unsafe state, Banker's Algorithm, 203
Unsupervised machine learning methods, anomaly detection, 924
Untainted (U) higher level security model, Android, 568
Untrained personnel problems, 21–22
Untrusted modules, software fault isolation for, 591–592
Unwinding theorem, 263–268, 596–597
Upper bound, lattices, 1154–1155
USENET news network, logic bomb, 797
User accounts, password sharing/administrative roles, 1100–1103
User advisories, electronic communications policy
  introduction, 1234
  overview of, 129–130
  privacy expectations, 1235–1236
  privacy limits, 1237–1239
  privacy protections, 1236–1237
  security considerations, 1239–1241
  user responsibilities, 1234–1235
User agent (UA), network mail service, 384–385
User agreement
  constraint-based DoS model, 205–207
  denial of service models, 204
  denial of service protection base, 213–214
  finite waiting time policy, 207–208
User classes
  configuring internal network, 1021–1022
  policy development practicum, 1008–1011
User identification numbers (UIDs)
  access control module refinement, 1113
  FreeBSD 10.3 and audit, 1049
  improper choice of operand/operation, 1140
  privileges and, 524–525
  process configuration on development system, 1049
  resource exhaustion and, 1124
  UNIX accounts represented by, 1049
  user configuration for development system, 1051–1052
User identity, 473–475
User interface
  access to roles/commands, 1106
  designing for program security, 1104–1105
  testing module, 1143–1144
User security component, key escrow systems, 355
User security practicum
  access, 1074–1079
  devices, 1084–1087
  electronic communications, 1092–1094
  files, 1080–1084
  overview of, 1072
  policy, 1072–1073
  processes, 1087–1092
  review, 1094–1097
Users
  checking input from untrusted sources, 1136–1137
  system security practicum, 1048–1053
  unacceptable conduct, 1210–1211
USTAT, misuse intrusion detection, 935–937
Usurpation, as class of threat, 7–9

V

Valid access lists (VALs), malware containment, 815–816
Validation
  of access control entries, 1124
  failure and CCM, 379
Validation, improper
  bounds checking, 1133–1134
  checking for valid data, 1135–1136
  checking input, 1136–1137
  designing for validation, 1137–1138
  error checking, 1134–1135
  flaws, 853
  improper indivisibility, 1138–1139
  improper sequencing, 1139
  overview of, 1132–1133
  type checking, 1134
Values, cookie, 489
Variable classes, information flow, 565–566
Variables
  checking that values are valid, 1135
  identifying covert channels, 601–602
VAX-11/750 computer, developing virus for, 781–782
VAX architecture, privilege and virtual machines, 1172–1175
VAX hardware, virtual machines, 583–585
VAX VMM system, auditing mechanisms, 897–898
VAX/VMM system, paging and, 1176
VAX/VMS system, paging and, 1176
VCMS control structure, Intel VT-x, 1174–1175
VCs. *See* verification conditions (VCs)
Vendor Security Analyst, RAMP, 735
Verbs, access control by Boolean expression, 35–36
Verification
  formal techniques for, 699–702
  as goal of Gypsy language, 711
  in HDM, 707–708
  implementation phase issues, 15
  TCSEC assurance requirements, 733
Verification conditions (VCs), in HDM, 708
Verified protection, TCSEC evaluation classes, 734
Verisign Corporation, CA issuance/authentication policies, 477
Version control and tracking, implementation management, 686
VFS layer. *See* virtual file system (VFS) layer
VFUNs, 703–704, 708
Victim, remote shell (*rsh*) attack, 966–967
View entry, CWE, 867
Vigenére cipher
  autokey versions of, 373–374
  one-time pad as variant of, 299
  overview of, 294–299
  as stream cipher, 370, 373–374
Violable prohibition/limit class of flaw, RISOS study, 850–851
Violations of law and policy
  acceptable use policy at UCD, 1209
  electronic communications policy, 1215
Virtual circuits, Tor, 497–498
Virtual file system (VFS) layer, adore-ng rootkit compromising, 778
Virtual machine monitor (hypervisor), 1171–1172
Virtual machine monitor (VMM), 583–584, 587, 1171–1172
Virtual machines
  defined, 583
  exercises, 1176–1177
  malware attempts to evade detection in, 811
  malware containment via, 816–817
  monitor, 1171–1172
  overview of, 1171
  paging and, 1175–1176
  physical resources and, 1175
  privilege and, 1172–1175
  providing isolation via, 583–585
  structure of, 1171
Virtual private network (VPNs), 1015, 1023–1024
Virtualization fault, Intel VT-i architecture, 1174
Virus detection problem, 803–808
Viruses. *See* Computer viruses
Visible functions, SPECIAL specification and, 703–704
Visual Network Rating Methodology (VNRM), 647
VMM. *See* virtual machine monitor (VMM)
VMX root and nonroot operations, Intel VT-x, 1174
VNRM. *See* Visual Network Rating Methodology (VNRM)
Voice recognition systems, biometrics, 443
VPNs. *See* virtual private network (VPNs)
VT-i architecture, Intel, 1174
VT-x architecture, Intel, 1174–1175
Vulnerabilities
  Apple patches for Trident, 800
  penetration studies to find, 827–828
  threats vs., 650
Vulnerability analysis
  configuring outer firewall, 1015
  frameworks, 849–864
  Gupta and Gligor's penetration analysis theory, 868–873
  introduction, 825–827
  overview of, 825
  penetration studies. *See* Penetration studies
  review, 873–878
  standards, 864–868
  vulnerability classification frameworks, 845–848
Vulnerability classification frameworks, 845–848
Vulnerability (or security flaw), defined, 825

W

Waiting time policy
  denial of service models, 204
  denial of service protection base, 213
  SYN flooding analysis and, 216
Walkthroughs (code review), implementation management, 687–688
Warning message, password aging, 435–436
Waterfall life cycle model, 639–641
Watergate scandal, 500
Weak tranquility principle, 162–163
Weakness base entry, CWE, 867
Weakness class entry, CWE, 867
Weakness variant entry, CWE, 867
Web, anonymity on
  for better or worse, 499–501
  electronic mail anonymizers, 491–494
  onion routing, 495–499
  overview of, 490–491

Web, identity on
  DNS security extensions, 487–488
  host identity, 484–485
  overview of, 484
  security issues with DNS, 487
  state and cookies, 488–490
  static and dynamic identifiers, 485–487
Well-formed transactions, integrity of data, 183–184
WFFs. *See* well-formed formulas (WFFs)
Whistleblower policy, 129
Windows Event Log Service, 882–883
Windows systems
  components/subcomponents, 663–664
  and monitors, 1086–1087
  penetration of, 843–844
  Trojan.Peacomm bot infecting, 794
  Windows 10 logger, 882–883
Wireless communication class, OSSTMM, 834
Wireless networks, security practicum, 1023–1024
Witness, graph rewriting rules as, 57
WordPerfect cipher, Kerberos, 338
Workstation security requirements, CISR, 743
Worms, computer, 790–792
Wrappers, blocking attacks, 977–978
Writable devices, user security, 1084–1085
Write right, access control matrix, 33
Writing, and Trusted Solaris, 147
Written passwords, obscuring, 419–420
WWW-clone, updating DMZ web server, 1019

X

X.509
  certificate conflicts, 479–480
  certificate signature chains, 346–348
  certificates and assurance of trust, 482–483
  PGP certificate signature chains vs., 349–350
  PKI, 350–352
  PKI certificate revocation, 359
  public-key certificates using Distinguished Names, 476
  using Resource Public Key Infrastructure, 361
XCP anti-piracy software, Sony BMG, 778–779
Xen 3.0 hypervisor, 584
XML. *See* extensible markup language (XML)
XP. *See* Extreme Programming (XP)
Xterm flaw, 859–862
Xterm security flaw, UNIX, 846–847

Y

Yaksha security system, 357
Yu-Gligor denial of service model, 204–205, 216

Z

Zmist computer virus, 789
Zones
  DMZ. *See* demilitarized zone (DMZ)
  in Trusted Solaris, 149–150