Table of Contents

Foreword by Royal Hansen xix

Foreword by Michael Wildpaner xxiii

Preface xxv

Part I. Introductory Material

1. The Intersection of Security and Reliability 3
On Passwords and Power Drills 3
Reliability Versus Security: Design Considerations 4
Confidentiality, Integrity, Availability 5
Confidentiality 6
Integrity 6
Availability 6
Reliability and Security: Commonalities 7
Invisibility 7
Assessment 8
Simplicity 8
Evolution 9
Resilience 9
From Design to Production 10
Investigating Systems and Logging 11
Crisis Response 11
Recovery 12
Conclusion 13

2. Understanding Adversaries. 15
Attacker Motivations 16
Attacker Profiles 18
Hobbyists 18
Vulnerability Researchers 18
Governments and Law Enforcement 19
Activists 21
Criminal Actors 22
Automation and Artificial Intelligence 24
Insiders 24
Attacker Methods 30
Threat Intelligence 30
Cyber Kill Chains™ 31
Tactics, Techniques, and Procedures 32
Risk Assessment Considerations 32
Conclusion 34

Part II. Designing Systems

3. Case Study: Safe Proxies. 37
Safe Proxies in Production Environments 37
Google Tool Proxy 40
Conclusion 42

4. Design Tradeoffs. 43
Design Objectives and Requirements 44
Feature Requirements 44
Nonfunctional Requirements 45
Features Versus Emergent Properties 45
Example: Google Design Document 47
Balancing Requirements 49
Example: Payment Processing 50
Managing Tensions and Aligning Goals 54
Example: Microservices and the Google Web Application Framework 54
Aligning Emergent-Property Requirements 55
Initial Velocity Versus Sustained Velocity 56
Conclusion 59

5. Design for Least Privilege. 61
Concepts and Terminology 62
Least Privilege 62

Zero Trust Networking 62
Zero Touch 63
Classifying Access Based on Risk 63
Best Practices 65
Small Functional APIs 65
Breakglass 67
Auditing 68
Testing and Least Privilege 71
Diagnosing Access Denials 73
Graceful Failure and Breakglass Mechanisms 74
Worked Example: Configuration Distribution 74
POSIX API via OpenSSH 75
Software Update API 76
Custom OpenSSH ForceCommand 76
Custom HTTP Receiver (Sidecar) 77
Custom HTTP Receiver (In-Process) 77
Tradeoffs 77
A Policy Framework for Authentication and Authorization Decisions 78
Using Advanced Authorization Controls 79
Investing in a Widely Used Authorization Framework 80
Avoiding Potential Pitfalls 80
Advanced Controls 81
Multi-Party Authorization (MPA) 81
Three-Factor Authorization (3FA) 82
Business Justifications 84
Temporary Access 85
Proxies 85
Tradeoffs and Tensions 86
Increased Security Complexity 86
Impact on Collaboration and Company Culture 86
Quality Data and Systems That Impact Security 87
Impact on User Productivity 87
Impact on Developer Complexity 87
Conclusion 87

6. Design for Understandability. 89
Why Is Understandability Important? 90
System Invariants 91
Analyzing Invariants 92
Mental Models 93
Designing Understandable Systems 94
Complexity Versus Understandability 94

Breaking Down Complexity 95
Centralized Responsibility for Security and Reliability Requirements 96
System Architecture 97
Understandable Interface Specifications 98
Understandable Identities, Authentication, and Access Control 100
Security Boundaries 105
Software Design 111
Using Application Frameworks for Service-Wide Requirements 112
Understanding Complex Data Flows 113
Considering API Usability 116
Conclusion 119

7. Design for a Changing Landscape. 121
Types of Security Changes 122
Designing Your Change 122
Architecture Decisions to Make Changes Easier 123
Keep Dependencies Up to Date and Rebuild Frequently 123
Release Frequently Using Automated Testing 124
Use Containers 124
Use Microservices 125
Different Changes: Different Speeds, Different Timelines 127
Short-Term Change: Zero-Day Vulnerability 129
Medium-Term Change: Improvement to Security Posture 132
Long-Term Change: External Demand 136
Complications: When Plans Change 138
Example: Growing Scope—Heartbleed 140
Conclusion 141

8. Design for Resilience. 143
Design Principles for Resilience 144
Defense in Depth 145
The Trojan Horse 145
Google App Engine Analysis 147
Controlling Degradation 150
Differentiate Costs of Failures 152
Deploy Response Mechanisms 154
Automate Responsibly 158
Controlling the Blast Radius 159
Role Separation 162
Location Separation 162
Time Separation 166
Failure Domains and Redundancies 166

Failure Domains 167
Component Types 169
Controlling Redundancies 172
Continuous Validation 174
Validation Focus Areas 175
Validation in Practice 176
Practical Advice: Where to Begin 179
Conclusion 181

9. Design for Recovery. 183
What Are We Recovering From? 184
Random Errors 184
Accidental Errors 185
Software Errors 185
Malicious Actions 185
Design Principles for Recovery 186
Design to Go as Quickly as Possible (Guarded by Policy) 186
Limit Your Dependencies on External Notions of Time 190
Rollbacks Represent a Tradeoff Between Security and Reliability 192
Use an Explicit Revocation Mechanism 200
Know Your Intended State, Down to the Bytes 204
Design for Testing and Continuous Validation 209
Emergency Access 210
Access Controls 211
Communications 212
Responder Habits 213
Unexpected Benefits 214
Conclusion 214

10. Mitigating Denial-of-Service Attacks. 217
Strategies for Attack and Defense 218
Attacker's Strategy 218
Defender's Strategy 219
Designing for Defense 220
Defendable Architecture 220
Defendable Services 222
Mitigating Attacks 223
Monitoring and Alerting 223
Graceful Degradation 223
A DoS Mitigation System 224
Strategic Response 225
Dealing with Self-Inflicted Attacks 226

User Behavior 226
Client Retry Behavior 228
Conclusion 228

Part III. Implementing Systems

11. Case Study: Designing, Implementing, and Maintaining a Publicly Trusted CA. 233
Background on Publicly Trusted Certificate Authorities 233
Why Did We Need a Publicly Trusted CA? 234
The Build or Buy Decision 235
Design, Implementation, and Maintenance Considerations 236
Programming Language Choice 237
Complexity Versus Understandability 238
Securing Third-Party and Open Source Components 238
Testing 239
Resiliency for the CA Key Material 240
Data Validation 241
Conclusion 241

12. Writing Code. 243
Frameworks to Enforce Security and Reliability 244
Benefits of Using Frameworks 245
Example: Framework for RPC Backends 247
Common Security Vulnerabilities 251
SQL Injection Vulnerabilities: TrustedSqlString 252
Preventing XSS: SafeHtml 254
Lessons for Evaluating and Building Frameworks 256
Simple, Safe, Reliable Libraries for Common Tasks 257
Rollout Strategy 258
Simplicity Leads to Secure and Reliable Code 259
Avoid Multilevel Nesting 260
Eliminate YAGNI Smells 260
Repay Technical Debt 261
Refactoring 262
Security and Reliability by Default 263
Choose the Right Tools 263
Use Strong Types 265
Sanitize Your Code 267
Conclusion 269

13. Testing Code. 271
Unit Testing 272
Writing Effective Unit Tests 272
When to Write Unit Tests 273
How Unit Testing Affects Code 274
Integration Testing 276
Writing Effective Integration Tests 277
Dynamic Program Analysis 277
Fuzz Testing 280
How Fuzz Engines Work 281
Writing Effective Fuzz Drivers 285
An Example Fuzzer 286
Continuous Fuzzing 289
Static Program Analysis 290
Automated Code Inspection Tools 291
Integration of Static Analysis in the Developer Workflow 296
Abstract Interpretation 299
Formal Methods 301
Conclusion 302

14. Deploying Code. 303
Concepts and Terminology 304
Threat Model 306
Best Practices 307
Require Code Reviews 307
Rely on Automation 308
Verify Artifacts, Not Just People 309
Treat Configuration as Code 310
Securing Against the Threat Model 311
Advanced Mitigation Strategies 314
Binary Provenance 314
Provenance-Based Deployment Policies 317
Verifiable Builds 319
Deployment Choke Points 325
Post-Deployment Verification 327
Practical Advice 328
Take It One Step at a Time 328
Provide Actionable Error Messages 328
Ensure Unambiguous Provenance 328
Create Unambiguous Policies 329
Include a Deployment Breakglass 329
Securing Against the Threat Model, Revisited 330

Conclusion 330

15. Investigating Systems. 333
From Debugging to Investigation 334
Example: Temporary Files 334
Debugging Techniques 336
What to Do When You're Stuck 344
Collaborative Debugging: A Way to Teach 348
How Security Investigations and Debugging Differ 350
Collect Appropriate and Useful Logs 351
Design Your Logging to Be Immutable 352
Take Privacy into Consideration 352
Determine Which Security Logs to Retain 354
Budget for Logging 357
Robust, Secure Debugging Access 359
Reliability 359
Security 359
Conclusion 360

Part IV. Maintaining Systems

16. Disaster Planning. 363
Defining "Disaster" 364
Dynamic Disaster Response Strategies 364
Disaster Risk Analysis 366
Setting Up an Incident Response Team 367
Identify Team Members and Roles 367
Establish a Team Charter 369
Establish Severity and Priority Models 369
Define Operating Parameters for Engaging the IR Team 370
Develop Response Plans 371
Create Detailed Playbooks 373
Ensure Access and Update Mechanisms Are in Place 373
Prestaging Systems and People Before an Incident 373
Configuring Systems 374
Training 375
Processes and Procedures 376
Testing Systems and Response Plans 376
Auditing Automated Systems 377
Conducting Nonintrusive Tabletops 378
Testing Response in Production Environments 379

Red Team Testing 381
Evaluating Responses 382
Google Examples 383
Test with Global Impact 383
DiRT Exercise Testing Emergency Access 384
Industry-Wide Vulnerabilities 384
Conclusion 385

17. Crisis Management. 387
Is It a Crisis or Not? 388
Triaging the Incident 389
Compromises Versus Bugs 390
Taking Command of Your Incident 391
The First Step: Don't Panic! 392
Beginning Your Response 392
Establishing Your Incident Team 393
Operational Security 394
Trading Good OpSec for the Greater Good 397
The Investigative Process 398
Keeping Control of the Incident 401
Parallelizing the Incident 401
Handovers 402
Morale 405
Communications 406
Misunderstandings 407
Hedging 407
Meetings 408
Keeping the Right People Informed with the Right Levels of Detail 409
Putting It All Together 410
Triage 411
Declaring an Incident 411
Communications and Operational Security 411
Beginning the Incident 411
Handover 412
Handing Back the Incident 413
Preparing Communications and Remediation 413
Closure 414
Conclusion 415

18. Recovery and Aftermath. 417
Recovery Logistics 418
Recovery Timeline 420

Planning the Recovery 421
Scoping the Recovery 421
Recovery Considerations 423
Recovery Checklists 428
Initiating the Recovery 429
Isolating Assets (Quarantine) 429
System Rebuilds and Software Upgrades 430
Data Sanitization 431
Recovery Data 432
Credential and Secret Rotation 433
After the Recovery 435
Postmortems 436
Examples 437
Compromised Cloud Instances 438
Large-Scale Phishing Attack 439
Targeted Attack Requiring Complex Recovery 441
Conclusion 442

Part V. Organization and Culture

19. Case Study: Chrome Security Team. 445
Background and Team Evolution 445
Security Is a Team Responsibility 448
Help Users Safely Navigate the Web 450
Speed Matters 450
Design for Defense in Depth 451
Be Transparent and Engage the Community 452
Conclusion 452

20. Understanding Roles and Responsibilities. 455
Who Is Responsible for Security and Reliability? 456
The Roles of Specialists 456
Understanding Security Expertise 458
Certifications and Academia 459
Integrating Security into the Organization 460
Embedding Security Specialists and Security Teams 462
Example: Embedding Security at Google 463
Special Teams: Blue and Red Teams 465
External Researchers 468
Conclusion 470

21. Building a Culture of Security and Reliability. 471
Defining a Healthy Security and Reliability Culture 473
Culture of Security and Reliability by Default 473
Culture of Review 474
Culture of Awareness 476
Culture of Yes 479
Culture of Inevitably 480
Culture of Sustainability 481
Changing Culture Through Good Practice 483
Align Project Goals and Participant Incentives 484
Reduce Fear with Risk-Reduction Mechanisms 484
Make Safety Nets the Norm 486
Increase Productivity and Usability 486
Overcommunicate and Be Transparent 488
Build Empathy 489
Convincing Leadership 490
Understand the Decision-Making Process 490
Build a Case for Change 491
Pick Your Battles 493
Escalations and Problem Resolution 494
Conclusion 494

Conclusion. 497

Appendix. A Disaster Risk Assessment Matrix. 499

Index. 501