"All-in-One Is All You Need."

ALL-IN-ONE

CSSLP®

Certified Secure Software Lifecycle Professional

EXAM GUIDE

THIRD EDITION

Online content includes:

- Test engine that provides full-length practice exams or customized quizzes by chapter or exam domain

Complete coverage of all CSSLP exam domains

Ideal as both a study tool and an on-the-job reference

Filled with practice exam questions and explanations

WM. ARTHUR CONKLIN,
PhD, CompTIA Security+™, CISSP®, CSSLP

DANIEL SHOEMAKER,
PhD

McGraw Hill

ALL IN ONE

CSSLP®

Certified Secure Software Lifecycle Professional

EXAM GUIDE

Third Edition

Wm. Arthur Conklin
Dan Shoemaker

New York Chicago San Francisco
Athens London Madrid Mexico City
Milan New Delhi Singapore Sydney Toronto

ISBN: 978-1-26-425821-5
MHID: 1-26-425821-6

The material in this eBook also appears in the print version of this title: ISBN: 978-1-26-425820-8,
MHID: 1-26-425820-8.

eBook conversion by codeMantra
Version 1.0

To Susan Conklin, my muse.
You make all journeys worth taking,
and your love carries me forward each day.
—Art Conklin

To Tamara,
Your love and support make all I do possible.
—Dan Shoemaker

This page intentionally left blank

ABOUT THE AUTHORS

Wm. Arthur Conklin, PhD, CompTIA Security+, CISSP, CSSLP, GISCP, GCFA, GCIA, GRID, GCIP, CRISC, CASP, is a professor and director of the Center for Information Security Research and Education in the College of Technology at the University of Houston. He holds two terminal degrees, a PhD in business administration (specializing in information security) from The University of Texas at San Antonio (UTSA) and a degree in electrical engineering (specializing in space systems engineering) from the Naval Postgraduate School in Monterey, California. He is a fellow of ISSA and (CS)2AI and a senior member of ASQ, IEEE, and ACM. Dr. Conklin's research interests lie in the areas of critical infrastructure cybersecurity. He has coauthored numerous books on information security and has written and presented numerous conference and academic journal papers. He has more than 20 years of teaching experience at the college level and has assisted in building two information security programs that have been recognized by the National Security Agency (NSA) and Department of Homeland Security (DHS) as Centers of Academic Excellence in Cyberdefense Education.

Dan Shoemaker, PhD (University of Detroit Mercy), is a professor and the director of the Graduate Program in Cybersecurity at UDM's Center for Cyber Security and Intelligence Studies. This center includes the NSA Center of Academic Excellence in Information Assurance Education. Dr. Shoemaker is a full-time professor, with 26 years as department chair. Dr. Shoemaker was one of the earliest academic participants in the development of software engineering as a discipline, starting at the Software Engineering Institute (SEI) in the fall of 1987. As the co-chair for the DHS National Workforce Training and Education Initiative for Software and Supply Chain Assurance, he is one of the three authors of the Software Assurance Common Body of Knowledge (CBK). Dr. Shoemaker is a Michigan man at heart, having received his degrees at the University of Michigan. He was named a Distinguished Visitor of the IEEE in 2020. He has written 12 books in the field and has more than 200 publications. He speaks extensively on security topics, both in the United States and internationally. He also served as an SME for the Securely Provision Knowledge Area for both versions of the NIST-NICE workforce framework. Finally, he recently served as an SME for the ACM Joint Committee's CSEC2017. He does all of this with the help and support of his wife, Tamara.

About the Technical Editor

Brian Barta, CISM, CSSLP, CISSP, CRISC, CISA, CCSK, CCSP, GROL, CEH, LSSBB, GIAC-GCIH, MCSE, is a proven leader with global security program management experience and a reputation for delivering enhanced results through the strategic implementation of organizational information security technology and process transformation. He has served as the director of Cloud Security Governance, Risk and Compliance for SaaS Cloud Security at Oracle, as well as the manager of Group Health's Governance, Risk and Compliance program where he established a program that aligned to business risks and is understood by business partners. Prior to supporting Group Health's success, he spent three years as the vulnerability management and advisory services security manager for T-Mobile improving the operational effectiveness and efficiencies of the program. Dr. Barta has more than 20 years of hands-on, in-the-weeds information security experience as well as the plain-speak, seasoned management buy-in skills required for security success in federal, state, and private industries. His passion is the measurable improvement of information security and regulatory compliance programs with a focus on security enablement of the business teams. He enjoys finding the problems and working with folks to fix them. He has also served as the chief information security officer at Washington State's Employment Security Department, information technology security architect at Washington State's Department of Corrections, and a variety of information security roles for the Department of Defense. His personal motto is "If you do what you love, you'll never work a day in your life." You can reach out to him via LinkedIn: https://www.linkedin.com/in/informationsecurityleadership.

CONTENTS AT A GLANCE

CONTENTS

ACKNOWLEDGMENTS

We, the authors of *CSSLP Certification All-in-One Exam Guide, Third Edition*, have many individuals who we need to acknowledge—individuals without whom this effort would not have been successful.

The list needs to start with those folks at McGraw Hill who worked tirelessly with the project's authors and led us successfully through the minefield that is a book schedule and who took our rough chapters and drawings and turned them into a final, professional product we can be proud of. We thank the good people from the acquisitions team, Wendy Rinaldi and Emily Walters. Wendy, our acquisitions editor, navigated a myriad of problems, put up with delays and setbacks, and made life easier for the author team. This project would not have been possible without her. And we thank the editorial and production team, Janet Walden and Thomas Somers; and at KGL, our project manager, Nitesh Sharma, and the composition and illustration teams. We also thank the technical editor, Brian Barta; the copy editor, Kim Wimpsett; the proofreader, Rachel Fogelberg; and the indexer, Ted Laux, for all their attention to detail that made this a finer work after they finished with it.

We would also like to thank Mary Ann Davidson, from Oracle Corporation, as she has provided inspiration as to the importance of getting software right and the real challenges of doing it in the real world, with business and technical considerations at every turn. She championed the cause of "an ounce of prevention is worth a pound of patching" while reminding us of the challenges of regression testing and the customer value of doing the right thing. The software development world needs more champions like her.

Dan Kaminsky, a true security pioneer, taken from us way too soon, stated that "The reality of most software development is that the consequences of failure are simply nonexistent." May this book and practitioners heed these words and seek to make them history.

This page intentionally left blank

INTRODUCTION

Computer security is becoming increasingly important today as we are becoming more reliant upon computers and the number of security incidents is steadily increasing. Vulnerable software is one of the root causes of many security incidents, and given the increasingly complex nature of software, this is not an issue that will be solved in the near term. Reducing the number and severity of vulnerabilities is both possible and useful in software projects. The principles behind the CSSLP certification can provide a roadmap to this goal.

Why Focus on Software Development?

Software vulnerabilities are preventable. Reducing the number and severity of vulnerabilities in software is not a trivial task; it is one that is complex and difficult to execute. Years of experience across numerous software development firms have resulted in proven methods of improving the software development process. Using these principles, development teams can produce software that has fewer vulnerabilities, and those that are found are of lesser risk. This reduces the total cost of development over the entire development lifecycle. This also improves the overall enterprise security posture of the users of the software, reducing their costs as well. Reduced risk, reduced cost, improved customer relations, and the advantages of improving the development process make the hard tasks required worth undertaking.

The Role of CSSLP

Creating and managing the necessary processes to build a secure development lifecycle is a significant task. The CSSLP credential speaks to the knowledge needed to make this possible. Software development is a team activity, and it is one that requires a series of processes in the enterprise. The tasks required to operate within a security-focused development environment require a workforce with an enhanced skillset. In addition to their individual skills in their areas of expertise, team members need to have an understanding of how a security-enhanced software development lifecycle process works. The body of knowledge for CSSLP covers these essential elements, and whether you are a designer, developer, tester, or program manager, the body of knowledge prepares you for operating in this environment.

How to Use This Book

This book covers everything you'll need to know for (ISC)²'s CSSLP exam. Each chapter covers specific objectives and details for the exam, as defined by (ISC)². We've done our best to arrange these objectives in the order set by (ISC)². Some topics may span objectives, so be sure to use the whole book for a complete reference.

Each chapter has several components designed to effectively communicate the information you'll need for the exam:

- Sidebars are designed to point out information, tips, and stories that will be helpful in your day-to-day responsibilities. In addition, they're just downright fun sometimes. Please note that although these entries provide real-world accounts of interesting pieces of information, they are sometimes used to reinforce testable material. Don't just discount them as simply "neat"—some of the circumstances and tools described in these sidebars may prove the difference in correctly answering a question or two on the exam.
- Exam Tips are exactly what they sound like. These are included to point out a focus area you need to concentrate on for the exam. No, they are not explicit test answers. Yes, they will help you focus your study.
- Notes provide interesting tidbits of information that are relevant to the discussion and point out extra information. Just as with the sidebars, don't discount them.

The Examination

Before we get to anything else, let us be frank: *This book will help you pass your test.* We've taken great pains to ensure that everything (ISC)² has asked you to know before taking the exam is covered in the book. Software development is a real task, and the information in this book needs to be included within the context of your experience in the development process. To get the value of the material in the book, it is important to combine it with the domain knowledge of software development processes.

Speaking of the test, these exam tips should help you:

- Be sure to pay close attention to the Exam Tips in the chapters. They are there for a reason. And retake the practice exams—both the end-of-chapter exams and the electronic exams. Practice will help, trust us.
- The exam is 125 questions, all multiple choice with four answers, and you are allowed to mark and skip questions for later review. Go through the entire exam, answering the ones you know beyond a shadow of a doubt. On the ones you're not sure about, *choose an answer anyway* and mark the question for further review (you don't want to fail the exam because you ran out of time and had a bunch of questions that didn't even have an answer chosen). At the end, go back and look at the ones you've marked. Change your answer only if you are absolutely, 100 percent sure about it.

- You will, with absolute certainty, see a couple of questions that will blow your mind. On every exam there are questions you will not recognize. When you see them, don't panic. Use deductive reasoning and make your best guess. Almost every single question on this exam can be whittled down to at least 50/50 odds on a guess. There is no penalty for guessing, so answer all questions.

Finally, dear reader, thank you for picking this book. We sincerely hope your exam goes well and wish you the absolute best in your upcoming career. Learn and use the material for good, and make better software.

Exam Objective Map

The following table has been constructed to allow you to cross-reference the official exam objectives with the objectives as they are presented and covered in this book.

Official Exam Objective	Chapter	All-in-One Coverage	Page
Domain 1: Secure Software Concepts			
1.1 Core Concepts	1	Core Concepts	3
1.2 Security Design Principles	2	Security Design Principles	31
Domain 2: Secure Software Requirements			
2.1 Define Software Security Requirements	3	Define Software Security Requirements	55
2.2 Identify and Analyze Compliance Requirements	4	Identify and Analyze Compliance Requirements	65
2.3 Identify and Analyze Data Classification Requirements	4	Identify and Analyze Compliance Requirements	65
2.4 Identify and Analyze Privacy Requirements	4	Identify and Analyze Compliance Requirements	65
2.5 Develop Misuse and Abuse Cases	5	Misuse and Abuse Cases	93
2.6 Develop Security Requirement Traceability Matrix (STRM)	5	Misuse and Abuse Cases	93
2.7 Ensure Security Requirements Flow Down to Suppliers/Providers	5	Misuse and Abuse Cases	93
Domain 3: Secure Software Architecture and Design			
3.1 Perform Threat Modeling	6	Secure Software Architecture	105
3.2 Define the Security Architecture	6	Secure Software Architecture	105
3.3 Performing Secure Interface Design	7	Secure Software Design	133
3.4 Performing Architectural Risk Assessment	7	Secure Software Design	133
3.5 Model (Non-Functional) Security Properties and Constraints	7	Secure Software Design	133

Official Exam Objective	Chapter	All-in-One Coverage	Page
3.6 Model and Classify Data	7	Secure Software Design	133
3.7 Evaluate and Select Reusable Secure Design	7	Secure Software Design	133
3.8 Perform Security Architecture and Design Review	7	Secure Software Design	133
3.9 Define Secure Operational Architecture (e.g., deployment topology, operational interfaces)	7	Secure Software Design	133
3.10 Use Secure Architecture and Design Principles, Patterns, and Tools	7	Secure Software Design	133
Domain 4: Secure Software Implementation			
4.1 Adhere to Relevant Secure Coding Practices (e.g., standards, guidelines and regulations)	8	Secure Coding Practices	157
4.2 Analyze Code for Security Risks	9	Analyze Code for Security Risks	181
4.3 Implement Security Controls (e.g., watchdogs, File Integrity Monitoring (FIM), anti-malware)	10	Implement Security Controls	199
4.4 Address Security Risks (e.g., remediation, mitigation, transfer, accept)	10	Implement Security Controls	199
4.5 Securely Reuse Third-Party Code or Libraries (e.g., Software Composition Analysis (SCA))	10	Implement Security Controls	199
4.6 Securely Integrate Components	10	Implement Security Controls	199
4.7 Apply Security During the Build Process	10	Implement Security Controls	199
Domain 5: Secure Software Testing			
5.1 Develop Security Test Cases	11	Security Test Cases	217
5.2 Develop Security Testing Strategy and Plan	12	Security Testing Strategy and Plan	229
5.3 Verify and Validate Documentation (e.g., installation and setup instructions, error messages, user guides, release notes)	13	Software Testing and Acceptance	241
5.4 Identify Undocumented Functionality	13	Software Testing and Acceptance	241
5.5 Analyze Security Implications of Test Results (e.g., impact on product management, prioritization, break build criteria)	13	Software Testing and Acceptance	241

Official Exam Objective	Chapter	All-in-One Coverage	Page
5.6 Classify and Track Security Errors	13	Software Testing and Acceptance	241
5.7 Secure Test Data	13	Software Testing and Acceptance	241
5.8 Perform Verification and Validation Testing	13	Software Testing and Acceptance	241
Domain 6: Secure Software Lifecycle Management			
6.1 Secure Configuration and Version Control (e.g., hardware, software, documentation, interfaces, patching)	14	Secure Configuration and Version Control	259
6.2 Define Strategy and Roadmap	14	Secure Configuration and Version Control	259
6.3 Manage Security Within a Software Development Methodology	14	Secure Configuration and Version Control	259
6.4 Identify Security Standards and Frameworks	14	Secure Configuration and Version Control	259
6.5 Define and Develop Security Documentation	14	Secure Configuration and Version Control	259
6.6 Develop Security Metrics (e.g., defects per line of code, criticality level, average remediation time, complexity)	14	Secure Configuration and Version Control	259
6.7 Decommission Software	14	Secure Configuration and Version Control	259
6.8 Report Security Status (e.g., reports, dashboards, feedback loops)	14	Secure Configuration and Version Control	259
6.9 Incorporate Integrated Risk Management (IRM)	15	Software Risk Management	273
6.10 Promote Security Culture in Software Development	15	Software Risk Management	273
6.11 Implement Continuous Improvement (e.g., retrospective, lessons learned)	15	Software Risk Management	273
Domain 7: Secure Software Deployment, Operations, Maintenance			
7.1 Perform Operational Risk Analysis	16	Secure Software Deployment	285
7.2 Release Software Securely	16	Secure Software Deployment	285
7.3 Securely Store and Manage Security Data	16	Secure Software Deployment	285
7.4 Ensure Secure Installation	16	Secure Software Deployment	285

Official Exam Objective	Chapter	All-in-One Coverage	Page
7.5 Perform Post-Deployment Security Testing	16	Secure Software Deployment	285
7.6 Obtain Security Approval to Operate (e.g., risk acceptance, sign-off at appropriate level)	17	Secure Software Operations and Maintenance	301
7.7 Perform Information Security Continuous Monitoring (ISCM)	17	Secure Software Operations and Maintenance	301
7.8 Support Incident Response	17	Secure Software Operations and Maintenance	301
7.9 Perform Patch Management (e.g., secure release, testing)	17	Secure Software Operations and Maintenance	301
7.10 Perform Vulnerability Management (e.g., scanning, tracking, triaging)	17	Secure Software Operations and Maintenance	301
7.11 Runtime Protection (e.g., Runtime Application Self-Protection (RASP), Web Application Firewall (WAF), Address Space Layout Randomization (ASLR))	17	Secure Software Operations and Maintenance	301
7.12 Support Continuity of Operations	17	Secure Software Operations and Maintenance	301
7.13 Integrate Service Level Objectives (SLO) and Service Level Agreements (SLA) (e.g., maintenance, performance, availability, qualified personnel)	17	Secure Software Operations and Maintenance	301
Domain 8: Secure Software Supply Chain			
8.1 Implement Software Supply Chain Risk Management	18	Software Supply Chain Risk Management	317
8.2 Analyze Security of Third-Party Software	18	Software Supply Chain Risk Management	317
8.3 Verify Pedigree and Provenance	18	Software Supply Chain Risk Management	317
8.4 Ensure Supplier Security Requirements in the Acquisition Process	19	Supplier Security Requirements	327
8.5 Support contractual requirements (e.g., Intellectual Property (IP) ownership, code escrow, liability, warranty, End-User License Agreement (EULA), Service Level Agreements (SLA))	19	Supplier Security Requirements	327

CSSLP Version 3 (2020)

In the summer of 2020, (ISC)^2 updated the domains of the CSSLP exam to more accurately reflect the current practice of secure development. This action was taken as a result of an analysis of job task analysis data. There are minor changes in the knowledge, skills, and abilities associated with the CSSLP domains. The domain weighting also has changed, as illustrated next.

CSSLP Certification All-in-One Exam Guide, Third Edition takes these changes into account and adds detailed information across all domains. The length and time for the exam will be the same, but the content is updated to reflect the new domain weightings.

Domains	Weight
1. Secure Software Concepts	10%
2. Secure Software Requirements	14%
3. Secure Software Architecture and Design	14%
4. Secure Software Implementation	14%
5. Secure Software Testing	14%
6. Secure Software Lifecycle Management	11%
7. Secure Software Deployment, Operations, Maintenance	12%
8. Secure Software Supply Chain	11%
Total	**100%**

This page intentionally left blank

PART I

Secure Software Concepts

This page intentionally left blank

CHAPTER 1

Core Concepts

In this chapter you will

- Learn about confidentiality, integrity, and availability
- Understand authentication and authorization
- Explore accountability
- Learn about nonrepudiation

Secure software development is intimately tied to the information security domain. For members of the software development team to develop secure software, a reasonable knowledge of security principles is required. The first knowledge domain area, Secure Software Concepts, comprises a collection of principles, tenets, and guidelines from the information security domain. Understanding these concepts as they apply to software development is a foundation of secure software development.

Security can be defined in many ways, depending upon the specific discipline that it is being viewed from. From an information and software development point of view, some specific attributes are commonly used to describe the actions associated with security: confidentiality, integrity, and availability (CIA). A second set of action-oriented elements, authentication, authorization, and accountability, provide a more complete description of the desired tasks associated with the information security activity. A final term, *nonrepudiation*, describes an act that one can accomplish when using the previous elements. An early design decision is determining what aspects of protection are required for data elements and how they will be employed.

EXAM TIP The term CIA is commonly used in the security industry to refer to confidentiality, integrity, and availability. Understanding the application of these three terms is important for the exam.

Confidentiality

Confidentiality is the concept of preventing the disclosure of information to unauthorized parties. Keeping secrets secret is the core concept of confidentiality. The identification of authorized parties makes the attainment of confidentiality dependent upon the concept of authorization, which is presented later in this chapter. There are numerous methods of keeping data confidential, including access controls and encryption.

The technique employed to achieve confidentiality depends upon whether the data is at rest, in transit, or in use. Access controls are typically preferred for data in use and at rest, while encryption is common for data in transit and at rest.

Issues involving protecting data from unauthorized disclosure can be decomposed to confidentiality requirements. The policy will define confidentiality requirements in terms of who can exchange what information between what endpoints. The key elements to determine are who the authorized users are and for what specific data elements or types. It can then be assumed that any other users would be unauthorized. Based on this set of information, the following items can be enumerated:

- Who is authorized to see what specific data elements
- What mechanism should be employed to enforce confidentiality
- What are the business requirements with respect to data collection, transfer, storage, and use with respect to confidentiality

Implementing Confidentiality

Confidentiality is the protection of information from observation by unauthorized parties. This is done primarily through the use of encryption to make the data unreadable. Encryption acts by scrambling the data in a manner that hides the true data values from observation. Encryption is a process that requires an algorithm and a key, and those who possess the correct keys can access the data. Confidentiality can be achieved by several means, both overt and covert. The typical mechanism is encryption, and it is crucial that best practices be employed in this aspect, for the typical failure of cryptographic protections falls into a few common categories. Encryption relies upon two elements, an algorithm and a key. Failure to use algorithms from approved cryptographic libraries, i.e., rolling your own crypto, almost always leads to failure. Also, failing to protect the key leads to failure.

The challenge to covert methods is simple: once found, they are no longer covert, and the secrecy value is summarily lost. Modern cryptographic processes are built upon a foundation of overt implementation of approved encryption algorithms, with steps taken to secure the key.

NOTE For cryptography to be successful, it must be properly implemented. Use only approved cryptographic libraries and functions. Never create your own implementations. Always protect the key, and never store it in code or a file where it can be stolen.

Integrity

Integrity is similar to confidentiality, except rather than protecting the data from unauthorized access, *integrity* refers to protecting the data from unauthorized alteration. Unauthorized alteration is a more fine-grained control than simply authorizing access.

Users can be authorized to view information but not alter it, so integrity controls require an authorization scheme that controls update and delete operations. For some systems, protecting the data from observation by unauthorized parties is critical, whereas in other systems, it is important to protect the data from unauthorized alteration. Controlling alterations, including deletions, can be an essential element in a system's stability and reliability. Integrity can also play a role in the determination of authenticity.

Issues involving protecting data from unauthorized alteration can be decomposed to integrity requirements. The policy will define integrity requirements in terms of who can alter which information elements between what endpoints. The key elements to determine are who is authorized to alter which specific data streams. It can then be assumed that any other users' alterations would be unauthorized. Based on this set of information, the following items can be enumerated:

- Who is authorized to alter which specific data elements
- What mechanism should be employed to detect errors and enforce integrity
- What are the business requirements with respect to data collection, transfer, storage, and use with respect to integrity

Because integrity can be expressed in the form of data errors, consideration should be given to requirements that can be used as a basis for monitoring error conditions.

Implementing Integrity

Integrity refers to protecting data from unauthorized alteration. Alterations can come in the form of changing a value or deleting a value. Integrity builds upon confidentiality, for if data is to be altered or deleted, then it is probably also visible to the user account. Users can be authorized to view information, but not alter it, so integrity controls require an authorization scheme that controls update and delete operations.

Protecting integrity can be done through the use of access control mechanisms. Individual specific levels of access with respect to read, write, and delete can be defined. Applying controls is just a piece of the integrity puzzle. The other aspect involves determining whether data has been altered or not. In some cases, this is just as important, if not more so, as the access control part. Determining whether the information has been altered or not requires some form of control other than simple access control.

If there is a requirement to verify the integrity of a data element throughout its lifecycle, the cryptographic function of hashing can be used. Hash functions can determine whether single or multiple bits of data change from the original form. The principle is simple: Take the original data, run it through a hash function, and record the result. At any subsequent time, running the current value of the data through the same hash function will produce a current digest value. If the two digest values match, then the data is unchanged. This can be for something small, such as a few bytes, to an entire file system—in all cases, the digest size is fixed based on the hash algorithm.

The use of cryptographic methods to digitally sign an item, or in the case of finished code, code-signing, proved a method of ensuring that an item is free of unauthorized modifications, or has not changed since signing. The addition of the identity of the signing party helps users determine authenticity.

The design issue is, how does one implement this functionality? When are hash digests calculated? How are the digests stored and transmitted by a separate channel to the party checking? Adding the necessary elements to allow hash-based checking of integrity requires the designing of the necessary apparatus for parties to be able to verify integrity. In some cases, like in Microsoft Update Service, this entire mechanism is designed into the transfer mechanism so that it is invisible to the end user. The design phase is where these decisions need to be made so that the coding phase can properly implement the desired behavior.

Availability

Access to systems by authorized personnel can be expressed as the system's *availability*. Availability is concerned with two issues: ensuring systems are available for authorized users when they require those systems and denying access to unauthorized users at all other times. Policy elements associated with determining access can be translated into the availability requirements. Availability is an often-misunderstood attribute, but its value is determined by the criticality of the data and its purpose in the system. For systems such as e-mail or web browsing, temporary availability issues may not be an issue at all. For IT systems that are controlling large industrial plants, such as refineries, availability may be the most important attribute. The criticality of data and its use in the system are critical factors in determining a system's availability. The challenge in system definition and design is to determine the correct level of availability for the data elements of the system. Common availability issues include denying illegitimate access to systems as well as preventing availability attacks such as denial of service.

NOTE If a system prevents a user from seeing a document they are authorized to see, this is a security failure, specifically an availability failure. Calling it "security" is wrong; it is a failure, and in the case of data access it can lead to wrong decisions.

The objective of security is to apply the appropriate measures to achieve a desired risk profile for a system. One of the challenges in defining the appropriate control objectives for a system is classifying and determining the appropriate balance of the levels of confidentiality, integrity, and availability in a system across the data elements. Although the attributes are different, they are not necessarily contradictory. They all require resources, and determining the correct balance between them is a key challenge early in the requirements and design process.

Confidentiality, Integrity, and Availability Requirements

At the core of software is data, and one of the key policy elements that needs to be defined for the software development team is the required level of data protection. Defining the required specific level of protection in terms of confidentiality, integrity, and availability is a key security requirement. Per Federal Information Processing Standard (FIPS) 199 and 44 U.S.C., Sec. 3542, the terms confidentiality, integrity, and availability are defined as follows:

CONFIDENTIALITY
"Preserving authorized restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information…"
A loss of confidentiality is the unauthorized disclosure of information.

INTEGRITY
"Guarding against improper information modification or destruction, and includes ensuring information non-repudiation and authenticity…"
A loss of integrity is the unauthorized modification or destruction of information.

AVAILABILITY
"Ensuring timely and reliable access to and use of information…"
A loss of availability is the disruption of access to or use of information or an information system.

Authentication

Authentication is the process of determining the identity of a user. All processes in a computer system have an identity assigned to them so that a differentiation of security functionality can be employed. Authentication is a foundational element of security, as it provides the means to define the separation of users by allowing the differentiation between authorized and unauthorized users. In systems where all users share a single account, they share the authentication and identity associated with that account.

It is the job of authorization mechanisms to ensure that only valid users are permitted to perform specific allowed actions. Authentication, on the other hand, deals with verifying the identity of a subject. To help understand the difference, consider the example of an individual attempting to log in to a computer system or network. Authentication is the process used to verify to the computer system or network that the individual is who they claim to be. Authorization refers to the rules that determine what that individual can do on the computer system or network after being authenticated. Three general methods are used in authentication. To verify their identity, a user can provide

- Something you know
- Something you have
- Something about you (something that you are)

The most common authentication mechanism is to provide something that only you, the valid user, should know. The most common example of something you know is the use of a userid (or username) and password. In theory, since you are not supposed to share your password with anybody else, only you should know your password, and thus by providing it you are proving to the system that you are who you claim to be. Unfortunately, for a variety of reasons, such as the fact that people tend to choose very poor and easily guessed passwords or share them, this technique to provide authentication is not as reliable as it should be. Other, more secure authentication mechanisms are consequently being developed and deployed.

NOTE Originally published by the U.S. government in one of the "rainbow series" of manuals on computer security, the categories of shared "secrets" are as follows:

- What users know (such as a password)
- What users have (such as tokens)
- What users are (static biometrics such as fingerprints or iris pattern)

Today, because of technological advances, new categories have emerged, patterned after subconscious behaviors and measurable attributes:

- What users do (dynamic biometrics such as typing patterns or gait)
- Where a user is (actual physical location)

Another common method to provide authentication involves the use of something that only valid users should have in their possession, commonly referred to as a *token*. A physical-world example of this is the simple lock and key. Only those individuals with the correct key will be able to open the lock and thus achieve admittance to your house, car, office, or whatever the lock was protecting. For computer systems, the token frequently holds a cryptographic element that identifies the user. The problem with tokens is that people can lose them, which means they can't log in to the system, and somebody else who finds the key may then be able to access the system, even though they are not authorized. To address the lost token problem, a combination of the something-you-know and something-you-have methods is used—requiring a password or personal identification number (PIN) in addition to the token. The key is useless unless you know this code. An example of this is the automatic teller machine (ATM) card most of us carry. The card is associated with a PIN that only you should know. Knowing the PIN without having the card is useless, just as having the card without knowing the PIN will also not provide you access to your account. Properly configured tokens can provide high levels of security at an expense only slightly higher than passwords.

The third authentication method involves using something that is unique about the user. We are used to this concept from television police dramas, where a person's fingerprints or a sample of their DNA can be used to identify them. The field of authentication that uses something about you or something that you are is known as *biometrics.* A number of different mechanisms can be used to accomplish this form of authentication, such as a voice print, a retinal scan, or hand geometry. The downside to these methods is the requirement of additional hardware and the lack of specificity that can be achieved with other methods.

EXAM TIP Three general factors are used in authentication. To verify your identity, you can provide something you know, something you have, or something about you (something that you are). When two different factors of this list are used together, it is referred to as *two-factor authentication*.

Multifactor Authentication

Multifactor authentication (or multiple-factor authentication) is simply the combination of two or more types of authentication. Five broad categories of authentication can be used: what you are (for example, biometrics), what you have (for instance, tokens), what you know (passwords and other information), somewhere you are (location), and something you do (physical performance). Two-factor authentication combines any two of these before granting access. An example would be a card reader that then turns on a fingerprint scanner—if your fingerprint matches the one on file for the card, you are granted access. Three-factor authentication would combine all three types, such as a smart card reader that asks for a PIN before enabling a retina scanner. If all three correspond to a valid user in the computer database, access is granted.

NOTE Whereas two-factor authentication combines any two methods, matching items such as a token with a biometric, three-factor authentication combines any three, such as a passcode, biometric, and a token.

Multifactor authentication methods greatly enhance security by making it difficult for an attacker to obtain all the correct materials for authentication. They also protect against the risk of stolen tokens, as the attacker must have the correct biometric, password, or both. More important, multifactor authentication enhances the security of biometric systems by protecting against a stolen biometric. Changing the token makes the biometric useless unless the attacker can steal the new token. It also reduces false positives by trying to match the supplied biometric with the one that is associated with the supplied token. This prevents the computer from seeking a match using the entire database of biometrics. Using multiple factors is one of the best ways to ensure proper authentication and access control.

Identity Management

Determining a user's identity requires an authentication process. Authentication is an identity verification process that attempts to determine whether users are who they say they are. Typically, a user enters some token to demonstrate to the system that they are who they purport to be. This requires a previous identification step, where the original record of the user is created. The identification step requires the following issues to be determined by requirements:

- Mechanism to be used for identity
- Management of identities, including reaffirmations

The mechanism to be used for identity sets the means to be used to authenticate users when services are requested. Identity is a step usually taken only once—when a user is established in the authentication system and a shared secret is created. This shared secret is what will be used to confirm identity during authentication events.

Identity management (IDM) is the comprehensive set of services related to managing the use of identities as part of an access control solution. Strictly speaking, the identity process is one where a user establishes their identity. Authentication is the act of verifying the supplied credentials against the set established during the identity process. The term *identity management* refers to the set of policies, processes, and technologies for managing digital identity information. *Identity and access management* (IAM) is another term associated with the comprehensive set of policies, processes, and technologies for managing digital identity information.

Identity management is a set of processes associated with the identity lifecycle, including the provisioning, management, and deprovisioning of identities. The provisioning step involves the creation of a digital identity from an actual identity. The source of the actual identity can be a person, a process, an entity, or virtually anything. The identity process binds some form of secret to the digital identity so that at future times, the identity can be verified. The secret that is used to verify identity is an item deserving specific attention as part of the development process. Protecting the secret, yet making it usable, are foundational elements associated with the activity.

In a scalable system, management of identities and the associated activities needs to be automated. A large number of identity functions needs to be handled in a comprehensive fashion. Changes to identities, the addition and removal of roles, changes to rights and privileges associated with roles or identities—all of these items need to be done securely and logged appropriately. The complexity of the requirements makes the use of existing enterprise systems an attractive option when appropriate.

EXAM TIP Security controls are audited annually under Sarbanes-Oxley (SOX) section 404, and IAM controls are certainly security controls. Designing and building IAM controls to support this operational issue is a good business practice.

Identity management can be accomplished through third-party programs that enhance the operating system offerings in this area. The standard operating system implementation leaves much to be desired in the management of user-selected provisioning or changes to identity metadata. Many enterprises have third-party enterprise-class IDM systems that provide services such as password resets, password synchronization, single sign-on, and multiple identity methods.

Having automated password resets can free up significant help-desk time and provide faster service to users who have forgotten their password. Automated password reset systems require a reasonable set of challenges to verify that the person requesting the reset is authorized to do so. Then, the reset must occur in a way that does not expose the old password. E-mail resets via a uniform resource locator (URL) are one common method employed for the reset operation. Users can have multiple passwords on different systems, complicating the user activity. Password synchronization systems allow a user to

synchronize a set of passwords across connected but different identity systems, making it easier for users to access the systems. Single sign-on is an industrial-strength version of synchronization. Users enter their credentials into one system, and it connects to the other systems, authenticating based on the entered credentials.

Identity Provider

The term *identity provider* (IdP) is used to denote a system or service that creates, maintains, and manages identity information. IdPs can range in scale and scope—from operating for a single system to operating across an enterprise. Additionally, they can be operated locally, distributed, or federated, depending on the specific solution. Multiple standards have been employed to achieve these services, including those built on the Security Assertion Markup Language (SAML), OpenID, and OAuth.

TIP The IdP creates, manages, and is responsible for authenticating identity.

Identity Attributes

How would you describe the elements of an identity? *Identity attributes* are the specific characteristics of an identity—name, department, location, login ID, identification number, e-mail address, and so on—that are used to accurately describe a specific entity. These elements are needed if one is to store identity information in some form of directory, such as an LDAP directory. The particulars of a schema need to be considered to include attributes for people, equipment (servers and devices), and services (apps and programs), as any of these can have an identity in a system. The details of schemas have already been taken care of via Active Directory, various IdPs, and so on, so this is not something that needs to be created; however, it does need to be understood.

Certificates

Certificate-based authentication is a means of proving identity via the presentation of a certificate. *Certificates* offer a method of establishing authenticity of specific objects such as an individual's public key or downloaded software. A digital certificate is a digital file that is sent as an attachment to a message and is used to verify that the message did indeed come from the entity it claims to have come from. Using a digital certificate is a verifiable means of establishing possession of an item (specifically, the certificate). When the certificate is held within a store that prevents tampering or extraction, this becomes a reliable means of identification, especially when combined with an additional factor such as something you know or a biometric.

Identity Tokens

An *Identity Token* is a security token that represents claims about the authentication of an entity by a specific authentication scheme. An *access token* is a form of identity token in physical form that identifies specific access rights and, in authentication, falls into the "something you have" factor. The advantages of token-based systems include the fact

that any token can be deleted from the system without affecting any other token or the rest of the system. The tokens themselves can also be grouped in multiple ways to provide different access levels to different groups of people. Smart cards can also be used to carry identification tokens. The primary drawback of token-based authentication is that only the token is being authenticated. Therefore, the theft of the token could grant anyone who possesses the token access to what the system protects.

The risk of theft of the token can be offset by the use of multifactor authentication. One of the ways that people have tried to achieve multifactor authentication is to add a biometric factor to the system. A less expensive alternative is to use hardware tokens in a challenge/response authentication process. In this way, the token functions as both a "something you have" and "something you know" authentication mechanism. Several variations on this type of device exist, but they all work on the same basic principles. The device has an LCD screen and may or may not have a numeric keypad. Devices without a keypad will display a password (often just a sequence of numbers) that changes at a constant interval, usually about every 60 seconds. When an individual attempts to log in to a system, they enter their own user ID number and then the number that is displayed on the LCD. These two numbers are either entered separately or concatenated. The user's own ID number is secret, and this prevents someone from using a lost device. The system knows which device the user has and is synchronized with it so that it will know the number that should have been displayed. Since this number is constantly changing, a potential attacker who is able to see the sequence will not be able to use it later, since the code will have changed.

SSH Keys

SSH keys are access credentials used by the Secure Shell (SSH) protocol. They function like usernames and passwords, but SSH keys are primarily used for automated processes and services. SSH keys are also used in implementing single sign-on (SSO) systems used by system administrators. SSH keys are exchanged using public key cryptography, and the keys themselves are digital keys.

Smart Cards

Smart cards are devices that store cryptographic tokens associated with an identity. The form factor is commonly a physical card, credit card sized, that contains an embedded chip that has various electronic components to act as a physical carrier of information.

Implementing Authentication

Authentication is the process of verifying that a user is who they claim to be and applying the correct values in the access control system. The level of required integration is high, from the storage systems that store the credentials and the access control information to the transparent handling of the information establishing or denying the validity of a credential match. When referring to authentication, one is referring to the process of verification of an identity. When one refers to the authentication system, one is typically referring to the underlying operating system aspect, not the third-party application that sits on top.

Authentication is, therefore, a validation that the user is presenting the known shared secret. This requires the establishment of a specific authentication method and a means of protecting the shared secret from disclosure. Other issues of authentication that need to be resolved include

- Method of authentication management
- Strength of authentication method

An authentication system may be as simple as a plaintext password system. An example of this would be basic authentication of a web-based system, where credentials are passed in the clear. For some uses, where no real security requirements exist, this provides sufficient control. Basic authentication provides no protection for credentials being passed between systems, so in reality, there is no security provided by this authentication method. For systems where more demanding security is needed, a more complicated system such as the Kerberos system may be employed with its encrypted secrets.

In all cases, authentication systems depend on some unique bit of information known only to the party being authenticated and the authentication system, a shared secret established during the identification phase. To protect this shared secret, in most cases, it is represented by a token such as a salted hash. This enables manipulation without risk of disclosure.

The concept of tokenization can be extended to other shared secrets, such as biometrics. A biometric can be converted to a specific representative code, which can then be treated like any other secret. Another form of tokenization is in the use of a security token or smart card system.

In the end, the requirement for authentication depends upon the level of security needed, the scale of users, and the management of users, coupled with the specific uses of the authentication. For a given system, multiple methods may be used; for example, high-value customers may use two-factor authentication, whereas visitors may have only a simple single factor. System administrators may require stronger authentication based on their higher access levels. The specifics for all of these individual circumstances can be decomposed from policy-driven requirements.

EXAM TIP Most authentication involves one party authenticating the other, as in a bank authenticating an account holder. Mutual authentication involves both parties authenticating each other at the same time, an essential process to prevent some forms of man-in-the-middle attacks.

Authentication systems come in a variety of sizes and types. Several different elements can be used as secrets as part of the authentication process. Passwords, tokens, biometrics, smart cards—the list can be long. The types can be categorized as something you know, something you have, or something you are. The application of one or more of these factors simultaneously for identity verification is a standard process in virtually all computing systems.

The underlying mechanism has some best-practice safeguards that should be included in a system. Mechanisms such as an escalating time lock-out after a given number of

successive failures, logging of all attempts (both successful and failed), and integration with the authorization system once authentication is successful are common protections. Password/token reset, account recovery, periodic changes, and password strength issues are just some of the myriad of functionalities that need to be encapsulated in an authentication system.

NOTE Passwords and other verification credentials, such as PINs, passphrases, token values, etc., are secrets and should never be accessible by anyone, including system administrators. Cryptography allows secrets to remain secret and still be used. If a system can e-mail you your password, it is not stored properly; disclosure should be impossible.

Numerous technologies are in use for authentication and authorization. Federated ID systems allow users to connect to systems through known systems. The ability to use your Facebook account to log in to another system is a useful convenience for many users. A known user experience (UX) interface and simple-to-use method for users to have a single sign-on environment, a federated ID system can use best-practice authentication systems. There are two main parties in these systems: a relying party (RP) and an identity provider (IdP). The user wants access to an RP and has credentials established on an IdP. As shown in Figure 1-1, a trust relationship exists between the RP and the IdP, and through data transfers between these parties, access can be granted.

Two of the more prevalent systems are OpenID and OAuth. OpenID was created for federated authentication, specifically to allow a third party to authenticate your users for you by using accounts that users already have. The OpenID protocol enables websites or applications (consumers) to grant access to their own applications by using another service or application (provider) for authentication. This can be done without requiring users to maintain a separate account/profile with the consumers.

OAuth was created to eliminate the need for users to share their passwords with third-party applications. The OAuth protocol enables websites or applications (consumers)

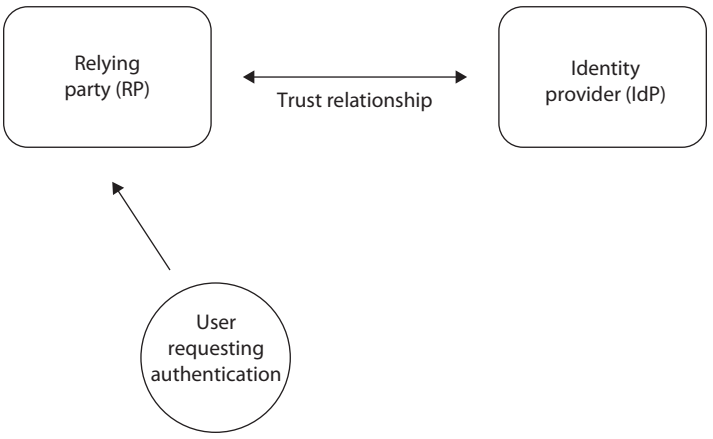

Figure 1-1 RP and IdP relationships

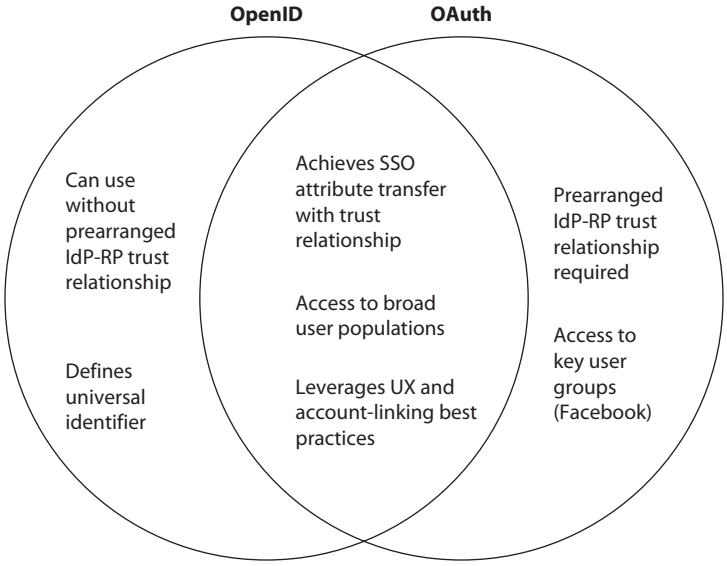

Figure 1-2 OpenID vs. OAuth

to access protected resources from a web service (service provider) via an application programming interface (API), without requiring users to disclose their service provider credentials to the consumers. Figure 1-2 highlights some of the differences and similarities between OpenID and OAuth.

Both OpenID (for authentication) and OAuth (for authorization) accomplish many of the same things. Each protocol provides a different set of features, which are required by their primary objective, but essentially, they are interchangeable. At their core, both protocols have an assertion verification method. They differ in that OpenID is limited to the "this is who I am" assertion, while OAuth provides an "access token" that can be exchanged for any supported assertion via an API.

Credential Management

There are numerous methods of authentication, and each has its own set of credentials that require management. The identifying information that is provided by a user as part of their claim to be an authorized user is sensitive data and requires significant protection. The identifying information is frequently referred to as *credentials*. These credentials can be in the form of a passed secret, typically a password. Other common forms include digital strings that are held by hardware tokens or devices, biometrics, and certificates. Each of these forms has advantages and disadvantages.

Each set of credentials, regardless of the source, requires safekeeping on the part of the receiving entity. Managing these credentials includes tasks such as credential generation, storage, synchronization, reset, and revocation. Because of the sensitive nature of manipulating credentials, all of these activities should be logged.

X.509 Credentials

X.509 refers to a series of standards associated with the manipulation of certificates used to transfer asymmetric keys between parties in a verifiable manner. A digital certificate binds an individual's identity to a public key, and it contains all the information a receiver needs to be assured of the identity of the public key owner. After a registration authority (RA) verifies an individual's identity, the certificate authority (CA) generates the digital certificate. The digital certificate can contain the information necessary to facilitate authentication.

X.509 Digital Certificate Fields

The following fields are included within an X.509 digital certificate:

- **Version number** Identifies the version of the X.509 standard that was followed to create the certificate; indicates the format and fields that can be used.
- **Serial number** Provides a unique number identifying this one specific certificate issued by a particular CA.
- **Signature algorithm** Specifies the hashing and digital signature algorithms used to digitally sign the certificate.
- **Issuer** Identifies the CA that generated and digitally signed the certificate.
- **Validity** Specifies the dates through which the certificate is valid for use.
- **Subject** Specifies the owner of the certificate.
- **Public key** Identifies the public key being bound to the certified subject; also identifies the algorithm used to create the private/public key pair.
- **Certificate usage** Specifies the approved use of the certificate, which dictates intended use of this public key.
- **Extensions** Allow additional data to be encoded into the certificate to expand its functionality. Companies can customize the use of certificates within their environments by using these extensions. X.509 version 3 has expanded the extension possibilities.

Certificates are created and formatted based on the X.509 standard, which outlines the necessary fields of a certificate and the possible values that can be inserted into the fields. As of this writing, X.509 version 3 is the most current version of the standard. X.509 is a standard of the International Telecommunication Union (www.itu.int). The IETF's Public-Key Infrastructure (X.509), or PKIX, working group has adapted the X.509 standard to the more flexible organization of the Internet, as specified in RFC 3280, and is commonly referred to as PKIX for Public Key Infrastructure X.509.

The public key infrastructure (PKI) associated with certificates enables the passing and verification of these digital elements between firms. Because certificates are

cryptographically signed, elements within them are protected from unauthorized alteration and can have their source verified. Building out a complete PKI infrastructure is a complex endeavor, requiring many different levels of protection to ensure that only authorized entities are permitted to make changes to the certification.

Setting up a functioning and secure PKI solution involves many parts, including certificate authorities, registration authorities, and certificate revocation mechanisms, meaning either certificate revocation lists (CRLs) or the Online Certificate Status Protocol (OCSP).

Figure 1-3 shows the actual values of the different certificate fields for a particular certificate in a web browser. The version of this certificate is V3 (X.509 v3), and the serial number is also listed—this number is unique for each certificate that is created by a specific CA. The CA used the SHA1 hashing algorithm to create the message digest value, and it then signed it using the CA's private key, which used the RSA algorithm.

X.509 certificates provide a wide range of benefits to any application that needs to work with public key cryptography. Certificates provide a standard means of passing keys, a standard that is accepted by virtually every provider and consumer of public keys. This makes X.509 a widely used and proven technology.

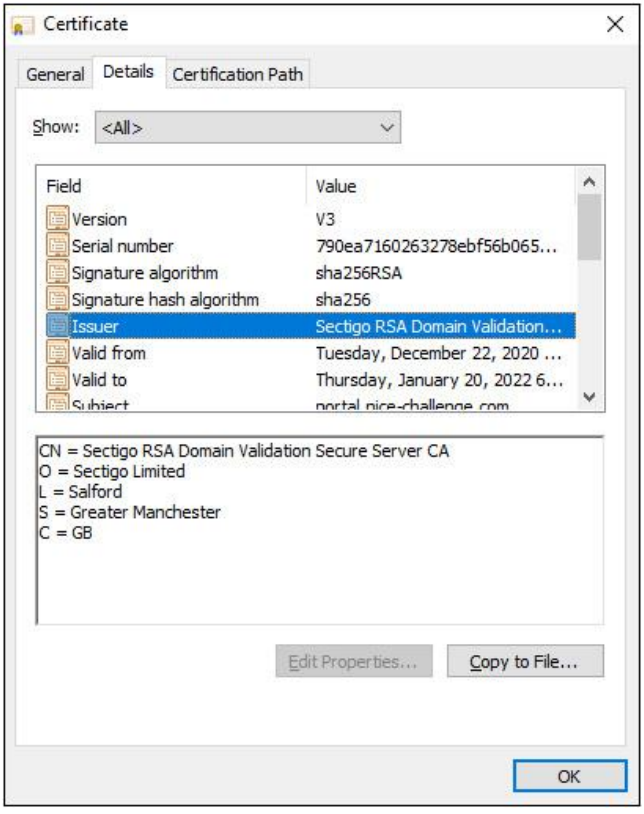

Figure 1-3
Digital certificate

Single Sign-On

Single sign-on makes it possible for a user, after authentication, to have their credentials reused on other applications without the user re-entering the secret. To achieve this, it is necessary to store the credentials outside of the application and then reuse the credentials against another system. There are a number of ways that this can be accomplished, but two of the most popular and accepted methods for sharing authentication information are Kerberos and Security Assertion Markup Language. The OpenID protocol has proven to be a well-vetted and secure protocol for SSO. However, as with all technologies, security vulnerabilities can still occur due to misuse or misunderstanding of the technology.

The key concept with respect to SSO is federation. In a federated authentication system, users can log in to one site and access another or affiliated site without re-entering credentials. The primary objective of federation is user convenience. Authentication is all about trust, and federated trust is difficult to establish. SSO can be challenging to implement, and because of trust issues, it is not an authentication panacea. As in all risk-based transactions, a balance must be achieved between the objectives and the risks. SSO-based systems can create single-point-of-failure scenarios, so for certain high-risk implementations, their use is not recommended.

Authorization

After the authentication system identifies a user, the authorization system takes over and applies the predetermined access levels to the user. *Authorization* is the process of applying access control rules to a user process and determining whether a particular user process can access an object. There are numerous forms of access control systems, which are covered later in the chapter. Three elements are used in the discussion of authorization: a requestor (sometimes referred to as the *subject*), the object, and the type or level of access to be granted. The authentication system identifies the subject to be one of a known set of subjects associated with a system. When a subject requests access to an object, be it a file, a program, an item of data, or any other resource, the authorization system makes the access determination as to grant or deny access. A third element is the type of access requested, with the common forms being read, write, create, delete, or the right to grant access rights to other subjects.

Once a user is positively identified by the authentication system, an authorization determination still needs to be made. The best method of decomposing authorization follows a subject-object-activity model. A subject is a user that has been authenticated. An object is the item that a user wants to perform an action on—it can be a file, a system, a database entry, a web page, basically any resource whatsoever. The activity is the desired action that the subject wants to invoke with the object. For file systems, this can be read, write, execute, delete, and so on. With database entries, there are basically the same options.

At this point, the loop back to the requirements associated with CIA is complete. The instantiation of the CIA requirements is performed by the authorization system. Whether a simple access control mechanism or a more sophisticated granular control such as a database access lattice, the concept is the same. The authorization system implements the control of access.

Access Control Mechanisms

The term *access control* has been used to describe a variety of protection schemes. It sometimes refers to all security features used to prevent unauthorized access to a computer system or network—or even a network resource such as a printer. In this sense, it may be confused with authentication. More properly, access is the ability of a subject (such as an individual or a process running on a computer system) to interact with an object (such as a file or hardware device). Once the individual has verified their identity, access controls regulate what the individual can actually do on the system. Just because a person is granted entry to the system does not mean they should have access to all the data the system contains.

NOTE It may seem that access control and authentication are two ways to describe the same protection mechanism. This, however, is not the case. Authentication provides a way to verify to the computer who the user is. Once the user has been authenticated, the access controls decide what operations the user can perform. The two go hand in hand but are not the same thing.

Access control systems exist to implement access control models. Different access control models are used based on the scale and scope of the elements of the subject-object-activity relationship.

NOTE Common access control models include mandatory access control (MAC), discretionary access control (DAC), role-based access control (RBAC), and rule-based access control (RBAC). These are covered in the next chapter.

Accountability (Auditing and Logging)

Accounting is a means of measuring activity. *Accountability* is the recording of actions and the users performing them. In IT systems, this can be done by logging crucial elements of activity as they occur. With respect to data elements, accounting is needed when activity is determined to be crucial to the degree that it may be audited at a later date and time. Management has a responsibility for ensuring work processes are occurring as designed. Should there be a disconnect between planned and actual operational performance metrics, then it is management's responsibility to initiate and ensure corrective actions are taken and effective. Auditing is management's lens to observe the operation in a nonpartisan manner. Auditing is the verification of what actually happened on a system. Security-level auditing can be performed at several levels, from an analysis of the logging function that logs specific activities of a system to the management verification of the existence and operation of specific controls on a system.

Auditing can be seen as a form of recording historical events in a system. Operating systems have the ability to create audit structures, typically in the form of logs that allow

management to review activities at a later point in time. One of the key security decisions is the extent and depth of audit log creation. Auditing takes resources, so by default it is typically set to a minimal level. It is up to a system operator to determine the correct level of auditing required based on a system's criticality. The system criticality is defined by the information criticality associated with the information manipulated or stored within it. Determination and establishment of audit functionality must occur prior to an incident, as the recording of the system's actions cannot be accomplished after the fact.

Audit logs are a kind of balancing act. They require resources to create, store, and review. The audit logs in and of themselves do not create security; it is only through the active use of the information contained within them that security functionality can be enabled and enhanced. As a general rule, all critical transactions should be logged, including when they occurred and which authorized user is associated with the event. Additional metadata that can support subsequent investigation of a problem is also frequently recorded.

NOTE A key element in audit logs is the employment of a monitoring, detection, and response process. Without mechanisms or processes to "trigger" alerts or notifications to admins based on specific logged events, the value of logging is diminished or isolated to a post-incident resource instead of contributing to an alerting or incident prevention resource.

The information system security policy decomposition for auditing activities should consider both risk-based and organizational characteristics. The risk-based issues can be examined as cases of three forms of audit-related risk (also known as *residual risk*):

- Inherent risk
- Detection risk
- Control risk

Inherent risks are those associated with the process and its inherent error rate, assuming no internal controls exist to handle the potential errors. Detection risk is the risk that an audit will not detect an issue that can result in material error. Control risk is the risk that controls will not detect or prevent material errors in a timely fashion.

Organizational characteristics that can drive auditing include items such as organizational history, the business environment, and supervisory issues. When decomposing policies, the following organizational security elements should be explored for auditing:

- Roles and responsibilities
- Separation of duties
- Training and qualifications
- Change management
- Control management

Logging

An important element in any security system is the presence of security logs. Logs enable personnel to examine information from a wide variety of sources after the fact, providing information about what actions transpired, with which accounts, on which servers, and with what specific outcomes. Many compliance programs require some form of logging and log management. The challenges in designing log programs are what to log and where to store it.

What needs to be logged is a function of several criteria. First, numerous compliance programs—HIPAA, SOX, PCI DSS, EOC, and others—have logging requirements, and these need to be met. The next criterion is one associated with incident response. What information would investigators want or need to know to research failures and issues? This is a question for the development team—what is available that can be logged that would provide useful information for investigators, either to the cause of the issue or to the impact?

The "where to log it" question also has several options, each with advantages and disadvantages. Local logging can be simple and quick for the development team. But it has the disadvantage of being yet another log to secure and integrate into the enterprise log management system. Logs by themselves are not terribly useful. What makes individual logs useful is the combination of events across other logs, detailing the activities of a particular user at a given point in time. This requires a coordination function, one that is supported by many third-party software vendors through their security information and event management (SIEM) tool offerings. These tools provide a rich analytical environment to sift through and find correlations in large datasets of security information.

To Log or Not to Log

One of the challenging questions is to what extent items should be logged. If all goes right and things never fail, there is still a need for some logging, if for no other reason than to administratively track what has occurred on the system. Logs provide a historical record of transactions through a system.

Logs can also provide information that can be critical in debugging problems encountered by the software. Logs generated as part of error processing routines can provide great insight not only into individual program issues but also into the overall system state at the time of the error.

When deciding what to log, there are two central questions. First, when would we need the information, and what information would be needed in that use case? This defines the core foundation of a log entry. As each log entry takes time and space, both to record and later to sift through when using, an economic decision on "does this deserve logging?" must occur.

The second foundational element to consider when logging is the security of the information being logged. Details, such as system state, user IDs, critical variable values, including things such as passwords—all of these are subject to misuse or abuse in the wrong hands and thus must be protected. Logs require security. The level of security is a key element to determine in the creation of logging requirements.

Syslog

Syslog is an Internet Engineering Task Force (IETF)–approved protocol for log messaging. It was designed and built around UNIX and provides a UNIX-centric format for sending log information across an Internet Protocol (IP) network. Although in its native form it uses User Datagram Protocol (UDP) and transmits information in the clear, wrappers are available that provide Transport Layer Security (TLS)–based security and Transmission Control Protocol (TCP)–based communication guarantees. While syslog is the de facto standard for logging management in Linux and UNIX environments, there is no equivalent in the Microsoft sphere of influence. Microsoft systems log locally, and there are some Microsoft solutions for aggregating logs to a central server, but these solutions are not as mature as syslog. Part of the reason for this is the myriad of third-party logging and log management solutions that provide superior business-level analytical packages that are focused on log data.

Nonrepudiation

Nonrepudiation is the concept of preventing a subject from denying a previous action with an object in a system. When authentication, authorization, and auditing are properly configured, the ability to prevent repudiation by a specific subject with respect to an action and an object is ensured. In simple terms, there is a system in place to prevent a user from saying they did not do something; this system can prove, in fact, whether an event took place or not. Nonrepudiation is a general concept, so security requirements must specify the subject, objects, and events for which nonrepudiation is desired, as this will affect the level of audit logging required. If complete nonrepudiation is desired, then every action by every subject on every object must be logged, and this could be a very large log dataset.

Secure Development Lifecycle

The term *secure development lifecycle* (SDL) comes from adding security to a software development lifecycle to reduce the number of security bugs in the software being produced. There are a wide range of different software development lifecycle models in use today across software development firms. To create a secure development lifecycle model, all one has to do is add a series of process checks to enable the development process to include the necessary security elements in the development process. The elements that are added are the same, although the location and methodology of adding the new elements to the process are dependent upon the original process.

Security vs. Quality

Quality has a long history in manufacturing and can be defined as fitness for use. Quality can also be seen as absence of defects. Although this may seem to be the same thing as security, it is not, but they are related. Software can be of high quality and free of defects and still not be secure. The converse is the important issue; if software is not of high quality and has defects, then it is not going to be secure. Because a significant percentage of software

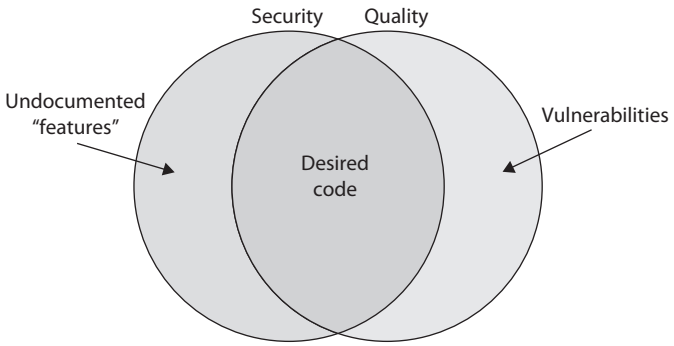

Figure 1-4
Software security vs. quality

security issues in practice are due to basic mistakes where the designer or developer should have known better, software quality does play a role in the security of the final product. Ross Anderson, a renowned expert in security, has stated that investments in software quality will result in a reduction of security issues, whether the quality program targets security issues or not.

Figure 1-4 illustrates some of the common issues associated with software quality versus security issues. If the product has quality but lacks security, then the result is a set of vulnerabilities. If the software is secure but is lacking in quality, then undocumented features may exist that can result in improper or undesired behaviors. The objective is to have both quality and security in the final output of the SDL process.

Security Features != Secure Software

A common misconception is when someone confuses security features with secure software. Security features are elements of a program specifically designed to provide some aspect of security. Adding encryption, authentication, or other security features can improve the usability of software and thus are commonly sought after elements to a program. This is not what secure development is about. Secure software development is about ensuring that all elements of a software package are constructed in a fashion where they operate securely. Another way of looking at this is the idea that secure software does what it is designed to do and only what it is designed to do. Adding security features may make software more marketable from a features perspective, but this is not developing secure software. If the software is not developed to be secure, then even the security features cannot be relied upon to function as desired. An example of this was the case of the Debian Linux random-number bug, where a design flaw resulted in a flawed random-number function, which, in turn, resulted in cryptographic failures.

Secure Development Lifecycle Components

SDLs contain a set of common components that enable operationalization of security design principles. The first of these components is a current team-awareness and education program. Having a team that is qualified to do the task in an SDL environment includes security knowledge. The next component is the use of security gates as a point to

check compliance with security requirements and objectives. These gates offer a chance to ensure that the security elements of the process are indeed being used and are functioning to achieve desired outcomes. Three sets of tools—bug tracking, threat modeling, and fuzzing—are used to perform security-specific tasks as part of the development process. The final element is a security review, where the results of the SDL process are reviewed to ensure that all of the required activities have been performed and completed to an appropriate level.

Software Team Awareness and Education

All team members should have appropriate training and refresher training throughout their careers. This training should be focused on the roles and responsibilities associated with each team member. As the issues and trends associated with both development and security are ever changing, it is important for team members to stay current in their specific knowledge so that they can appropriately apply it in their work.

Security training can come in two forms: basic knowledge and advanced topics. Basic security knowledge, including how it is specifically employed and supported as part of the SDL effort, is an all-hands issue, and all team members need to have a functioning knowledge of this material. Advanced topics can range from new threats to tools, techniques, etc., and are typically aimed at a specific type of team member (i.e., designer, developer, tester, project manager). The key element of team awareness and education is to ensure that all members are properly equipped with the correct knowledge before they begin to engage in the development process.

Gates and Security Requirements

As part of the development process, periodic reviews are conducted. In an SDL, these are referred to as *gates*. The term *gate* is used to signify a condition that one must pass through. To pass the security gate, a review of the appropriate security requirements is conducted. Missing or incomplete elements can prevent the project from advancing to the next development phase until these elements or issues are addressed. This form of discipline, if conducted in a firm and uniform manner, results in eventual behavior by the development team where the gates are successfully negotiated as a part of normal business. This is the ultimate objective; the inclusion of security is a part of the business process.

Bug Tracking

Bug tracking is a basic part of software development. As code is developed, bugs are discovered. Bugs are elements of code that have issues that result in undesired behaviors. Sometimes, the behavior results in something that can be exploited, and this makes it a potential security bug. Bugs need to be fixed, and hence, they are tracked to determine their status. Some bugs may be obscure, impossible to exploit, and expensive to fix; thus, the best economic decision may be to leave them until the next major