Security Design Principles

Keith A. Watson, CISSP, CISA
GLSP February 2011 Meeting

Overview

- Introduction to information security
  - Definitions and terms
- Software security
- The C-I-A and other concepts
- Security and the SDLC
- Software Security Principles

Objectives

1. Have a fundamental understanding of information security as it applies to the security of applications
2. Learn how security can be integrated into the software development life cycle
3. Have a understanding of security principles applied in the design and implementation process

Introduction to Software Security

- What is Information Security?
- How does it apply to Software Security?
- Trends that affect the security of software
- Attacker’s Advantage
- Defender’s Dilemma
- Security versus Reliability
- Secure Development in the SDLC

What is Information Security?

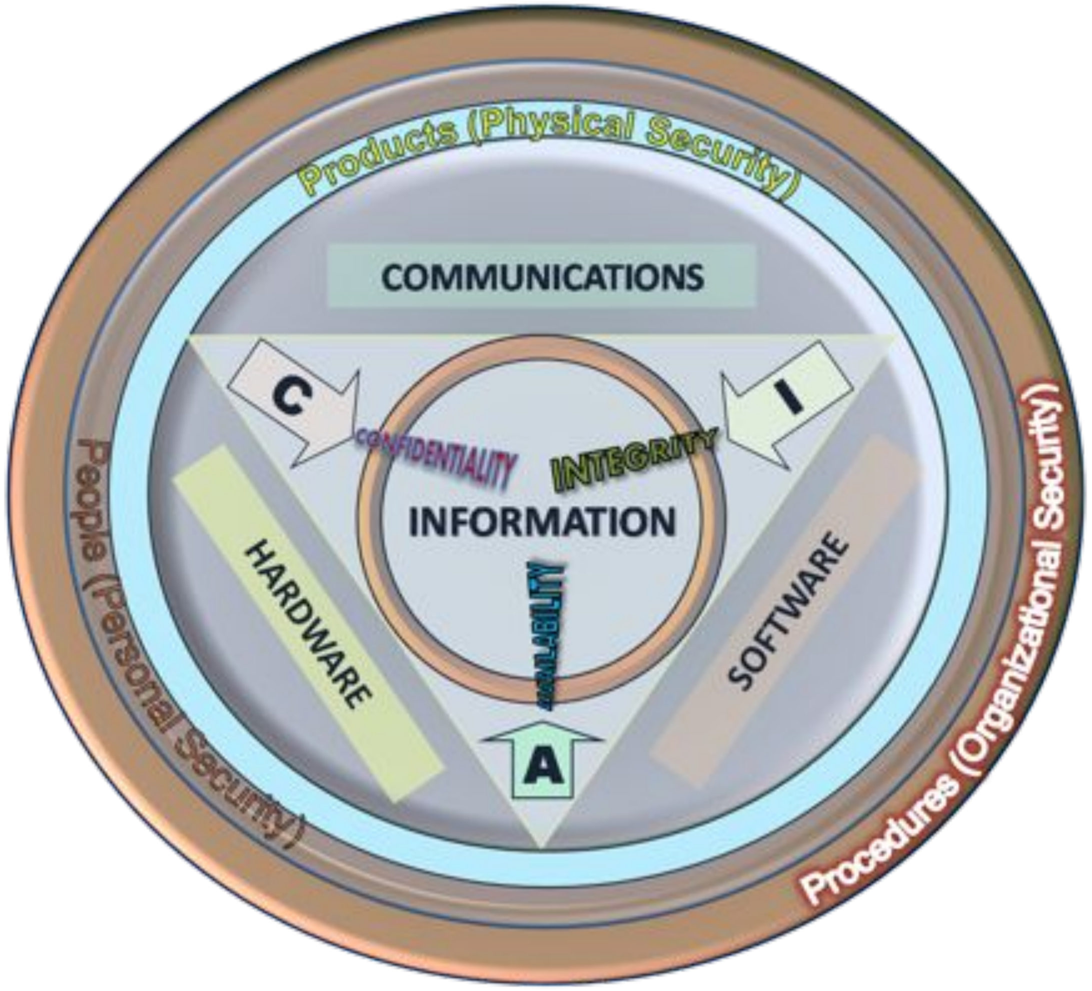
- The process of protecting information in all of its forms
  - Virtual (customer data, internal information, proprietary designs, source code, etc)
  - Physical (written, verbal, prototypes, printouts, disks, devices, backup tapes, etc)
- Computer security is a subset of information security

U.S. Definition of Information Security

"…protecting information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction in order to provide…integrity…confidentiality…availability."

Title 44, Chapter 35, Subchapter III, § 3542 (b)(1)
U.S. Code

Information Security



Products (Physical Security)
COMMUNICATIONS
C
I
CONFIDENTIALITY
INTEGRITY
INFORMATION
AVAILABILITY
HARDWARE
SOFTWARE
A
People (Personal Security)
Procedures (Organizational Security)

Courtesy of John Manuel

What is Software Security?

- Also called “Application Security”
  - Shortened to “app security” or “appsec”
- Area of information security focused on software systems, applications, software components

Why is Software Vulnerable?

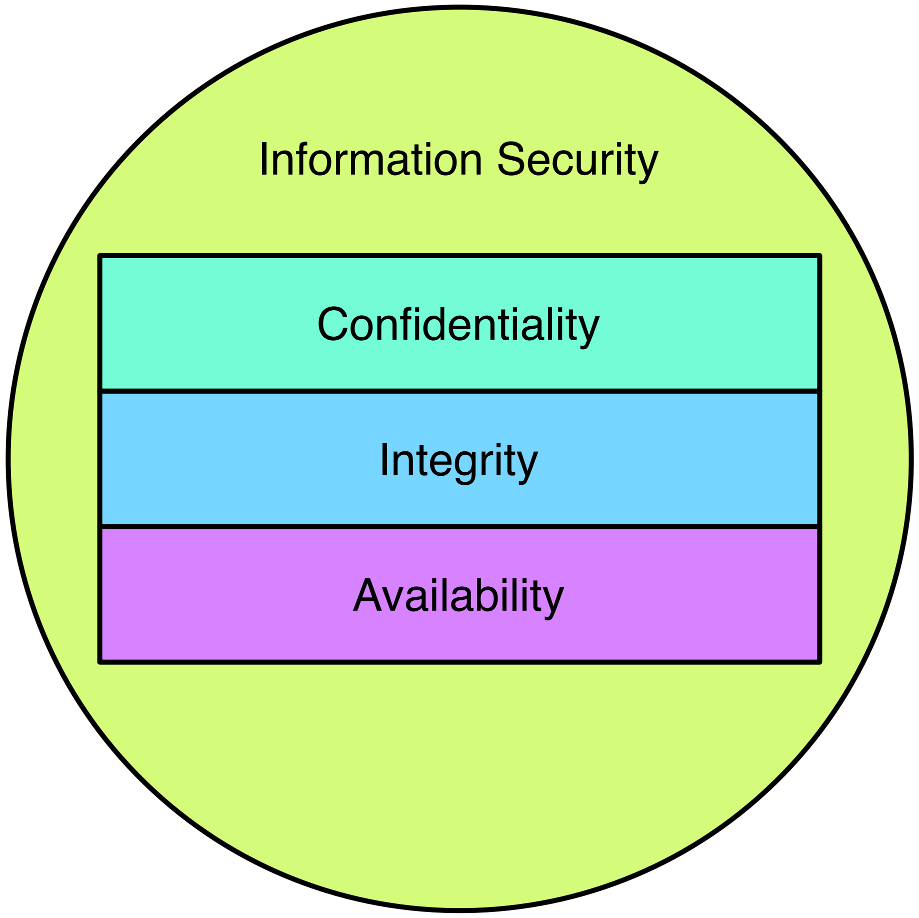
- Andy Ozment, Cambridge software security researcher, suggested:
  - Lack of Developer Motivation
  - Lack of Developer Knowledge
  - Lack of Tools
- The PITAC reports the software development is not yet a science or a rigorous discipline
  - The process is not controlled to prevent vulnerabilities

DoD's Definition of Software Assurance

"…the level of *confidence* that software functions *as intended* and is free of vulnerabilities, either *intentionally* or unintentionally designed or inserted as part of the software."

Komaroff and Baldwin, DoD Software Assurance Initiative, September 2005.

The CIA


Information Security
Confidentiality
Integrity
Availability

Confidentiality

- Keeping information secret
- Prevention of intentional or unintentional disclosure of sensitive information to unauthorized individuals
- Disclosure can occur through cryptanalysis, inference, traffic analysis, covert channels, bugs, insiders, attackers, etc.

Integrity

- Ensuring that information has not been modified or deleted
- Prevent:
  - Modifications by *unauthorized* individuals or processes
  - Unauthorized modifications by *authorized* individuals or processes
- Data must maintain consistency

Availability

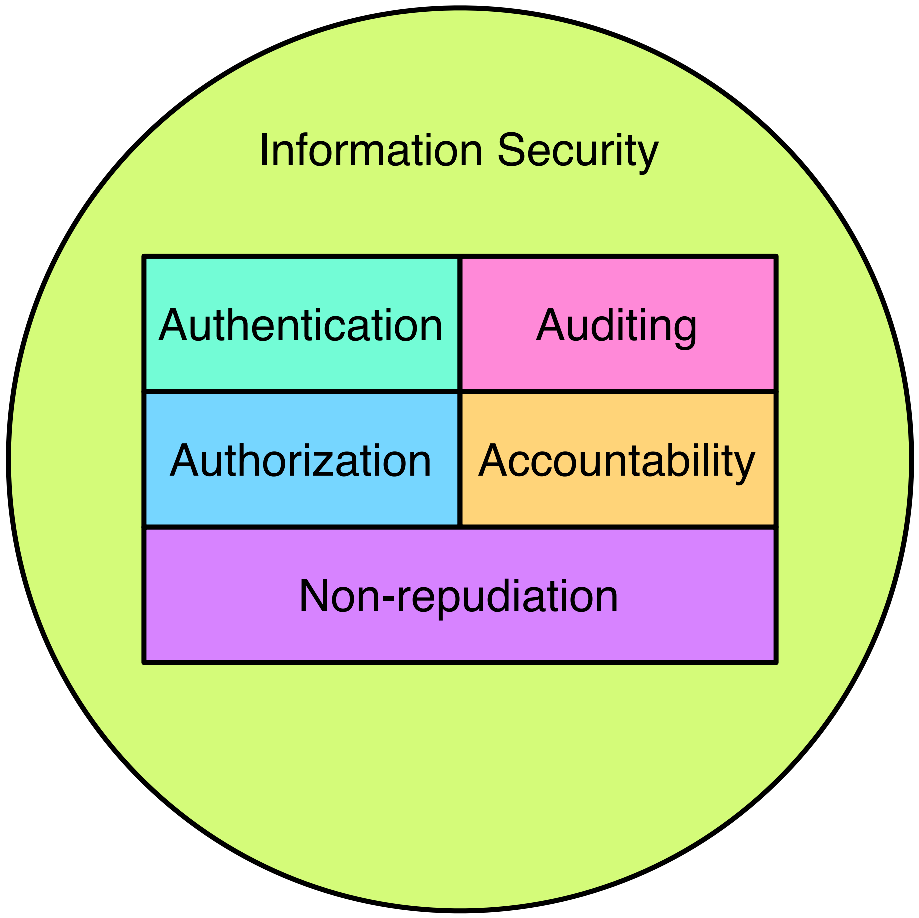
- Avoid “Why can’t I read my email?!”
- Ensuring the timely and reliable access to information and information systems by the appropriate personnel
- Maintaining the proper working order of systems to reduce likelihood of downtime
- Preventing the misuse of resources that could lead to a denial of service

The CIA of Information Security

- Confidentiality, Integrity, and Availability are often referred to as the C-I-A triad of information systems security
- Guiding principles

- The opposite of C-I-A is D-A-D: Disclosure, Alteration, and Destruction

Other Key Concepts



Authentication

- Act of verifying someone's identity
- Something you know
  - A secret: password, passphrase, single-use PW
- Something you have
  - A token: smart card, OTP card, RFID card
- Something you are
  - A biometric: fingerprint, iris pattern, vein pattern
- Two-factor authentication is more effective

Authorization

- Act of checking that a user is allowed to do something
- Verification of user’s authority
- Access Control Lists are a general mechanism
- Access Control Models:
  - Discretionary Access Control (DAC)
  - Mandatory Access Control (MAC)
  - Role-based Access Control (RBAC)

Authorization

- Bell-LaPadula Model
  - Access control model used by governments
  - Users and Resources are classified by sensitivity
    - Top Secret, Secret, Unclassified, for example
  - Users cannot read to a higher level (no read up)
  - Users cannot write to a lower level (no write down)

Auditing

- Periodic review of the security of a system
- Ongoing monitoring of activities on a system
- Auditors can review the security controls
  - Determine threats, vulnerabilities, lapses
  - Assign a risk value and provide recommendations
- Audit trails can provide detailed information about user activities and transactions

Accountability

- Act of determining what happened
- Recalling the actions an attacker or user took during a transaction
- Techniques used to ensure accountability:
  - Audit trails and logs correlation and analysis
  - Digital Forensics
  - Detailed postmortem analysis

Non-repudiation

- Act of ensuring undeniability of a transaction
- Using a variety of evidence to prove that the actions took place by a specific individual
- Cryptographic protocols exists to provide non-repudiation
  - Trusted third parties
  - Digital Signatures
- Non-repudiation is difficult and expensive

Trends Affecting Software Security

- Moving from centralized to decentralized
  - Mainframes to workstations
  - Central administration to user administration
- Moving from fixed to portable to "take it everywhere" devices
  - Desktops to laptops
  - Laptops to mobile devices

Trends Affecting Software Security

- Moving from custom software to commodity software
  - Mainframe applications to desktop applications
  - Desktop applications to web applications
  - “Standard” software to open standards
- Moving from open source to close source back to open source
  - Open source versus Closed source vulnerabilities

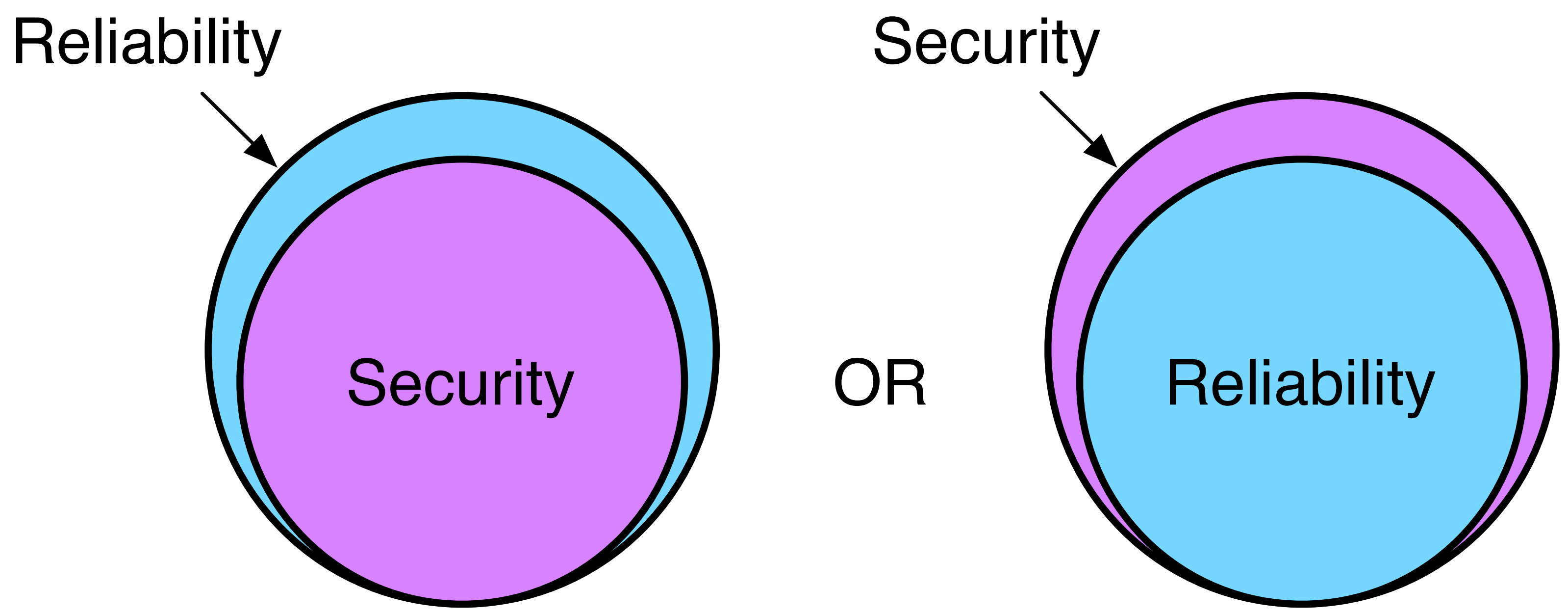
Attacker's Advantage
Defender's Dilemma

- Lack of central control over software
- Software built for a variety of environments
- Programmers embed secrets in code
- Programmers invent crypto algorithms
- Poor or lack of formal testing processes
- Poor design and architecture processes
- Constant break/fix cycle

Security versus Reliability

- Security is a part of reliability
  - Security addresses correctness
  - A security failure can lead to reliability problems
- Reliability is a part of security
  - Reliability addresses availability
  - A reliability problem can impact operations

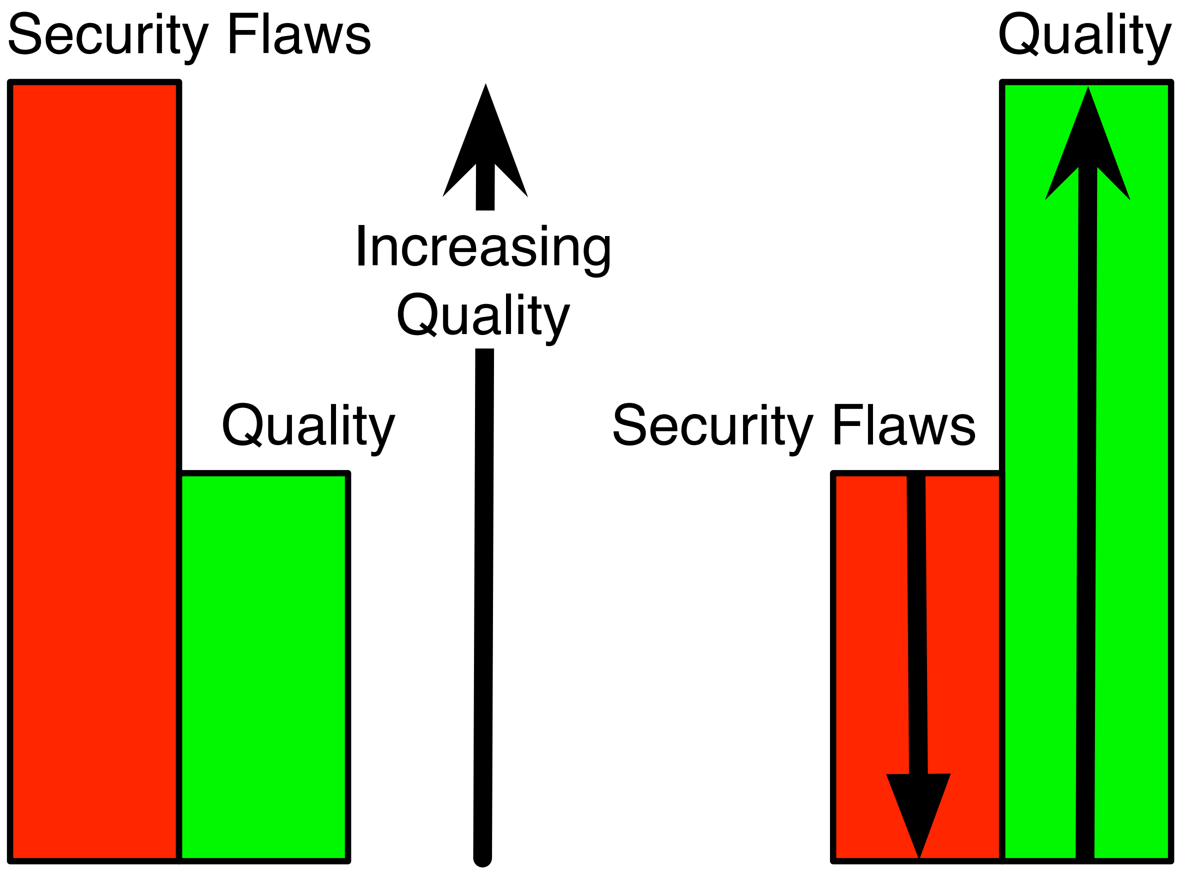
Security versus Reliability



Quality and Security

- Increasing the investment in software quality leads to fewer security problems
  - Most effective quality measure is a code walk-through or code review
  - Programmers are more careful when their code will be scrutinized by others
- Testing and review processes are improved
- Impacts to quality/security addressed early

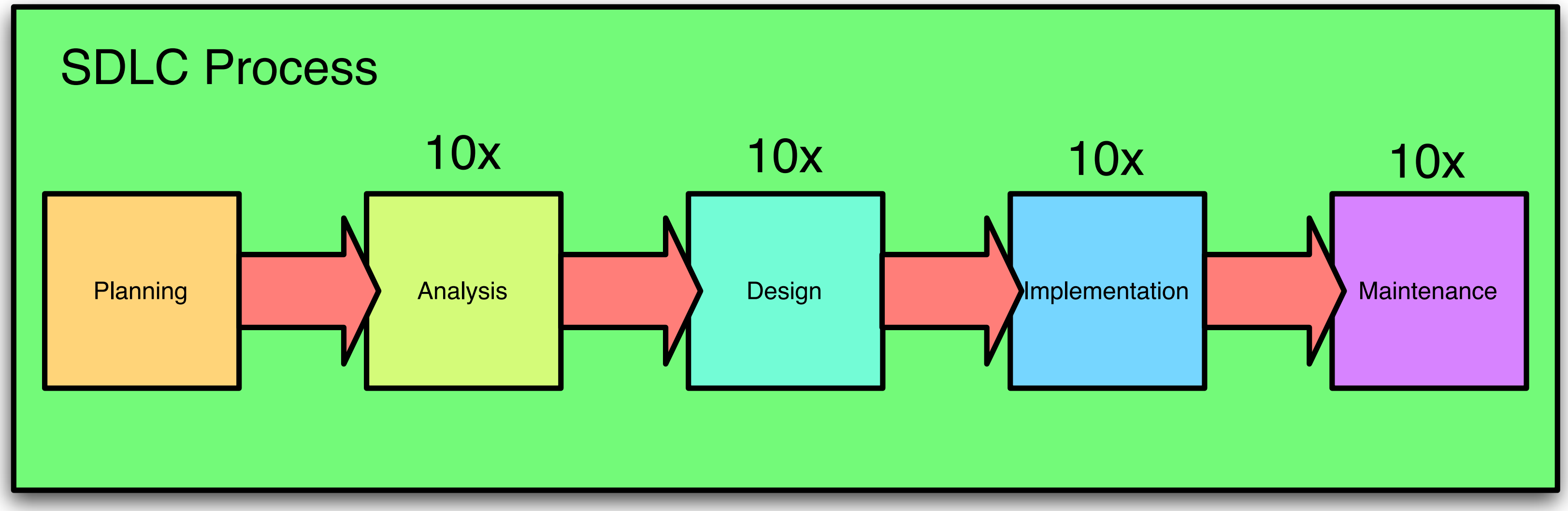
Quality and Security



Secure Development in the SDLC

- The software development life cycle is an appropriate place to define security efforts
- “best practices” and checklists lead to small improvements in app security
- Security is better addressed early
  - 10x money, time, effort to correct at each higher stage

SDLC Bug Fix Costs

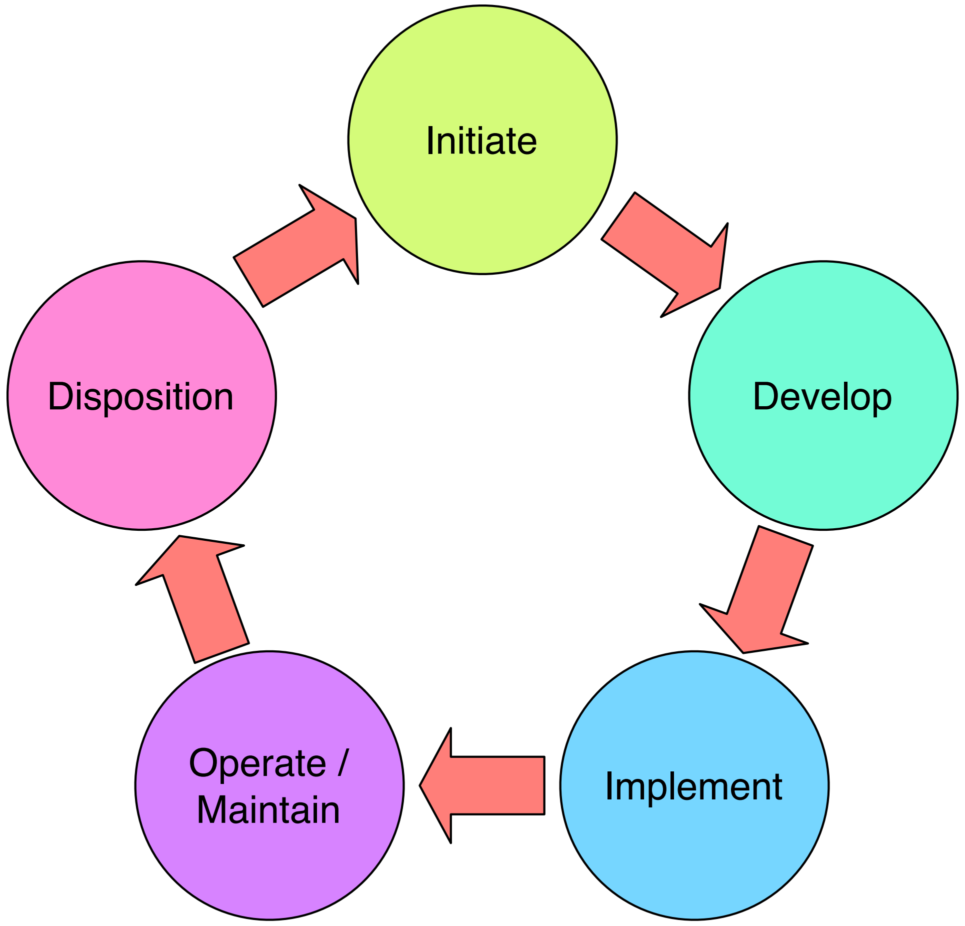


SDLC Phases
(as defined by NIST)

- Phase 1: Initiation
- Phase 2: Development (or Acquisition)
- Phase 3: Implementation (or Assessment)
- Phase 4: Operations and Maintenance
- Phase 5: Disposition

- Note: Your organization's process may vary. For example, DoJ's SDLC has ten steps.

SDLC Phases


Initiate
Develop
Implement
Operate / Maintain
Disposition

Phase 1: Initiation

- Identify or specify applicable policies or laws
- Classify information involved
  - Public, Internal Use, Sensitive
- Develop confidentiality, integrity, availability objectives
- Gather and address security requirements
- Preliminary risk assessment

Phase 2: Development

- Risk assessment
- Select initial baseline of security controls
- Refinement of security controls
- Security control design
- Security operation planning
- Security test planning

Phase 3: Implementation

- Component inspection and acceptance
- Security control integration
- User and administrator training
- Security test and acceptance
- Statement of residual risk

Phase 4: Operations and Maintenance

- Change management control and auditing
- Security monitoring
- Incident response and handling
- Auditing
- Intrusion detection
- Contingency plans and testing
- Periodic assessments and penetration tests

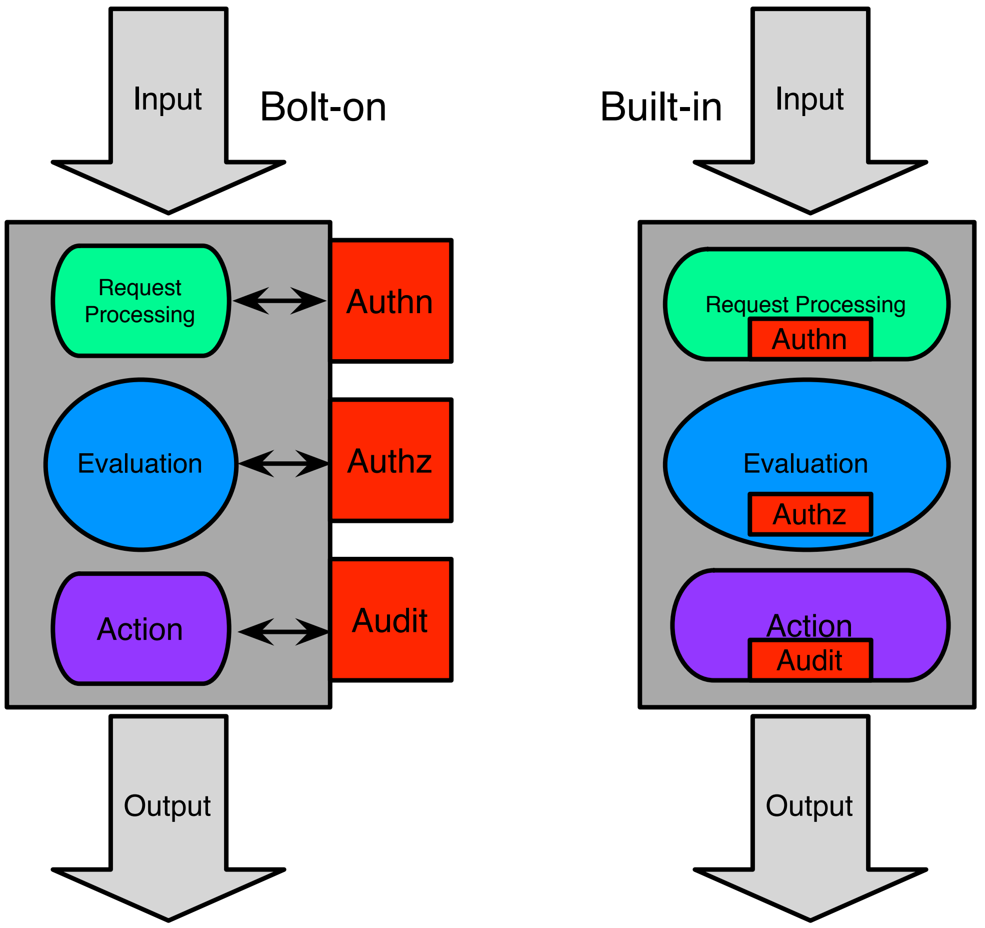
Phase 5: Disposition

- Transitioning of sensitive data
- Component disposal
- Media sanitization
- Information archiving

Security Features

- Security Features are not the same thing as Secure Features
- Security Features are often “bolted on” instead of “built in”
  - Afterthoughts in the design process
  - User demanded after security lapses
  - Premium features that cost extra
- Having security features means that was not a priority in your design, implementation

Security Bolted-On or Built-In



Software Security Principles

- There are several sources of sound principles for software security:
  - Software Security Principles, **Building Secure Software** (John Viega and Gary McGraw)
  - Secure Design Principles, **Foundations of Security: What Every Programmer Needs to Know** (Neil Daswani, Christoph Kern, Anita Kesavan)
  - Saltzer Schroeder Principles, "The Protection of Information in Computer Systems," 1974
- The following principles provide coverage for 90% of potential problems

Software Security Principles from Building Secure Software

1. Secure the Weakest Link
2. Practice Defense in Depth
3. Fail Securely
4. Use Least Privilege
5. Compartmentalize
6. Keep It Simple
7. Promote Privacy
8. Hiding Secrets is Hard
9. Be Reluctant to Trust
10. Use Community Resources

Principle #1: Secure the Weakest Link

- Attackers target the weakest component
  - These are most likely to be broken
  - High reward for lowest amount of effort
- Attackers will avoid or bypass system security
  - Why attack a firewall when you attack the web server behind the firewall?
  - Why attack the authentication system when you can pose as a legitimate user and call support?

Principle #2: Practice Defense in Depth

- Defense in Depth involves diversification
  - Multiple layers, different types
  - Increases likelihood of detection and response
  - Decreases likelihood of complete access
- Compare banks to convenience stores
  - Banks: guards, cameras, vaults, silent alarms
  - Stores: cameras, little cash
  - Think about rewards. Think about risk to robbers.

Principle #3: Fail Securely

- Complexity is the enemy of security
- Involves designing a system to withstand the failure of components while ensuring security
- A web server access control module cannot read the ACL file. What should happen?
- A payroll service system cannot connect to the employee database. What should happen?
- The payroll system is given a corrupted file. What should happen?

Principle #4:
Use Least Privilege

- A user or program is given the minimum amount of privilege needed to accomplish a specific task
- Attackers may take advantage of additional privileges to access sensitive information
- UNIX web servers need system privileges to acquire port 80/tcp, then drop to user privs
- Other applications use mediated privileges

Principle #5: Compartmentalize

- Break up the system into units and isolate security privileges
  - Minimize damage if attacked
  - Avoids "all-or-nothing" architecture approach
- Postfix (sendmail alternative) uses multiple processes running in "compartments"
  - Each process runs with different privileges
  - Each process is contained in a directory (chroot())

Principle #6: Keep it Simple

- Design and implementation should be simple
- Complex designs are hard to understand and implement correctly
- Complex implementations are hard to understand and debug quickly
- Security operations should pass through a small number of “choke points”
  - There must be no way around these

Principle #7: Promote Privacy

- Avoid compromising the privacy of the user
- Be diligent in protecting personal information
  - In some industries, regulations define privacy requirements (HIPAA, GLBA, COPPA, etc)
- Some trade offs made with usability
  - Storing personal information for easy recall
- Systems and applications should reveal less
  - Removing or altering version information

Principle #8:
Hiding Secrets is Hard

- Secrets are difficult to keep sometimes
- Complex binaries do not hide secrets well
- Attackers are surprisingly good at reverse engineering
  - Copy protection schemes fall victim to R.E.
  - DVD DeCSS is perhaps the best example R.E.
- Insiders are another significant issue
  - SIPRnet has 500k users; Wikileaks source

Principle #9: Be Reluctant to Trust

- Trust is often extended too frequently
  - Clients and servers can be compromised
  - Trust relationships can be exploited
- Compromised web servers can still provide secure connections to clients
  - Serve malware files and false data
- rsh, rlogin use simple host/port access control
  - Source host and port can be spoofed

Principle #10: Use Community Resources

- The security and programming communities are open and sharing
- The crypto community is open because algorithms must be reviewed and proven correct
- Use libraries and code that has been well-tested and “abused” by the community
- Beware that more eyes do not always find bugs

Software Design Principles from Foundations of Security

Foundations of Security contains most of the previous principles

- Secure By Default
- Usability
- Security Features Do Not Imply Security

Software Design Principles

- Secure By Default
  - Design security in instead of adding secure options
  - Make secure options the default configuration
- Usability
  - Complex systems do not encourage proper use
  - Users don’t read documentation
  - Stringent security configurations will force users to bypass security or find less than secure methods

Software Design Principles

- Security Features Do Not Imply Security
  - Security Application Developers are regular programmers and they make mistakes too
  - A significant number of security products have had major security issues over the years
  - Enabling some security features does ensure that a system is secure

Saltzer Schroeder Principles

Saltzer and Schroeder published a paper at the Fourth ACM Symposium on Operating Systems in 1974 that highlighted 11 security design principles

- Complete Mediation
- Open Design
- Psychological Acceptability

Saltzer Schroeder Principles

- Complete Mediation
  - Every access should be checked for authority
  - Access control is system-wide and consistent
  - Changes in authority should be promulgated
- Open Design
  - The security of the system should not be a secret
  - The security should lie in the protection of password and keys, not the internal operations
  - Open design encourage review and critique

Saltzer Schroeder Principles

- Psychological Acceptability
  - Design the user interface so that users operate with the appropriate protection consistently
  - Use the user’s mental model of security as a guide so that mistakes are minimized

Review

- Introduced information security concepts and core principles
  - C-I-A and AAAA and Non-repudiation
- Software security
  - Quality and Reliability
- How security fits into the SDLC
- Software Security Principles