OntarioTech Business & IT

Secure Software Development Life Cycle

By

Dr. Madani

Winter 2024

Learning Objectives

Upon completion of this material, you should be able to

- Describe the key security requirements of confidentiality, integrity, and availability (CIA).
- Identify most common threats and attacks against information systems.
- Describe fundamental security design principles.
- Define secure software development lifecycle fundamental components.

Optional Reading

- CSSLP by Conklin and Shoemaker: Chapters 1, 2, 14, and 16

Software Development Life Cycle (SDLC)

- Is a big-picture breakdown of all phases (i.e., steps) involved in software creation:
  - Planning, coding, testing, deploying, etc.
  - We use SDLC and “Software Development Process” interchangeably.
  - To manage degree of complexities involved, there are number of methodologies:
    - Waterfall
    - Spiral

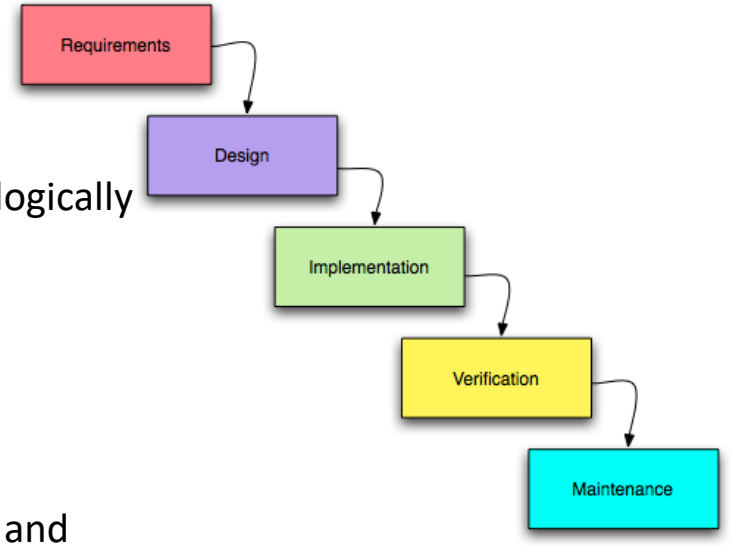
    Traditional Style
    - Agile Software Development (e.g., XP and Scrum)
    - Rapid Prototyping
    - Etc.

    More modern

[Credit: Wikipedia]

SDLC – cont'd

- **Waterfall** Software Development Process
  - Originally developed by Winston w. Royce in 1970.
    - Used in manufacturing and construction industries at first!
  - The original model to software development. Managers loved it because everything flow logically from the beginning to the end!
  - It is less "iterative and flexible" as progress flows in one direction.

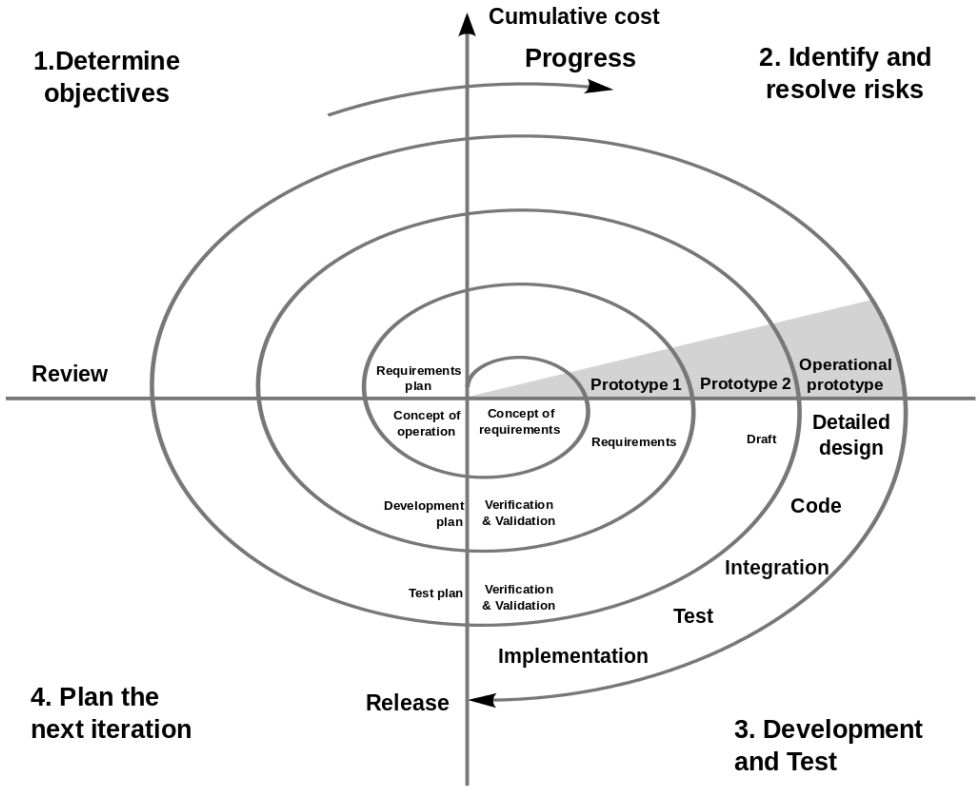


https://www.umsl.edu/~hugheyd/is6840/waterfall.html

- Pros:
  - Design errors are captured before any software is written!
  - Excellent technical documentation is created as part of deliverables from "Requirements" and "Design" phases.
  - Cost estimation can be very accurate!
- Cons:
  - Clients often can't express their requirements at the abstract level, early on! They tend to change their mind and realise about new requirements after they see a prototype! No change flexibility!

SDLC – cont'd

- Spiral Software Development Process
  - Risk-driven and incremental (dev&delivery) software development process, created by Barry Boehm in 1988.
  - Each loop in the spiral represents a phase of the software (creation) process .
  - The innermost loop might be concerned with system feasibility, the next loop with requirement definitions, the next loop with system design and so on.
  - The spiral model combines change avoidance with change tolerance.
    - It is assumed that changes are due to "risks" in the project and in order to avoid change is to model and mitigate project risks!

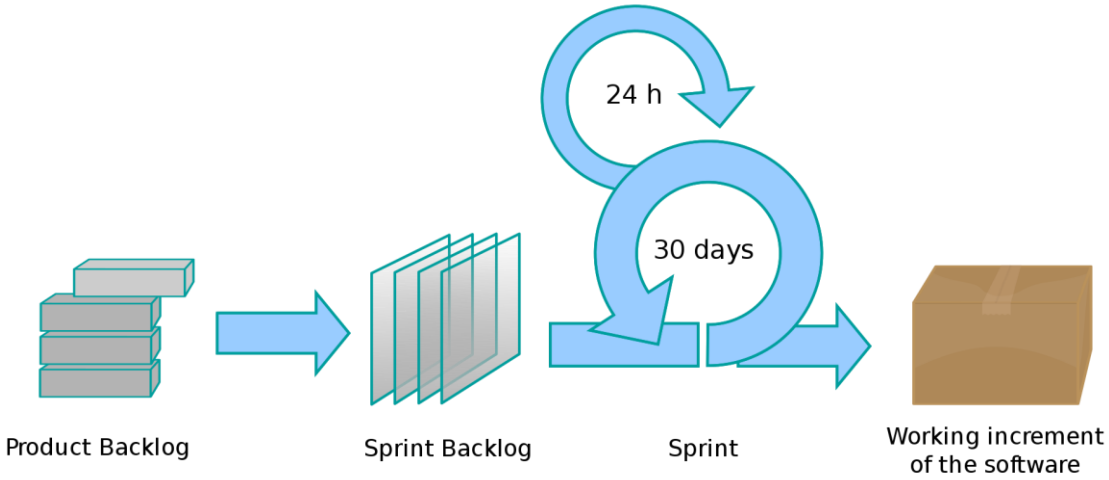


SDLC – cont'd

- **Agile** Methodology
  - Is a collection of different software development processes – is a philosophy.
  - It tries to address the issue of "lack of communication" with the customer beyond the introductory stages.
  - In today's day and age, "user requirements" change rapidly. Why? Because of evolving business needs in a very short period.
    - Rapid changes are needed for a business to remain competitive!
  - The "Agile Manifesto": http://agilemanifesto.org/
    - Individuals and interaction over processes and tools!
    - Working software over comprehensive documentation!
    - Customer collaboration over contract negotiation!
    - Responding to change over following a plan!

SDLC – cont'd

- **Scrum** Software Development Process
  - Is an agile methodology – commonly used in software development but can be used as a general project management framework.
  - Designed for smaller teams (ten or fewer)
  - Break their work into goals that can be completed within a time-boxed iterations also called "sprints".
    - No sprint can be longer than a month, usually two weeks!
    - Daily meetings are held to assess progress.
    - At the end of sprint, work is demonstrated to stakeholders and feedback is solicited – next sprint will be planned.



https://en.wikipedia.org/wiki/Scrum_(software_development)

SDLC – cont'd

- **Extreme Programming (XP)** Software Development Process
  - Is an agile methodology – embracing change is part of its core values.
  - XP starts with user stories (instead of traditional list of requirements).
  - Program tests are written before the code (this is also known as test driven development or TDD)
  - Pair programming is common: two programmers on a section of a code (often sitting side by side!)
    - "Pair Programming takes a task which is traditionally solitary and turns it into a team effort. The task consists of the driver, who sits at the computer entering code, and the navigator, who sits next to the driver, offering advice and assisting with decisions. The programmers switch roles regularly, although there is no set time interval between switches. Reported benefits of Pair Programming include improved product quality, improved developer morale and job satisfaction, fewer coding errors, and improved mentoring and training (Hanks, 2008; Drobka, Noftz, Raghu, 2004)." - https://www.umsl.edu/~hugheyd/is6840/extreme.html

Secure SDLC

- S-SDLC (also referred to as “Secure Development Lifecycle”)
  - Operationalization of common components that enable of security design principles in SDLC.
  - Can be adopted by all the existing software development processes.
- Common Components:
  - Software Team Awareness and Education: all team members are aware of common and advanced security threats.
  - Gates and Security Requirements: periodic review of security requirements of the project and making sure the requirements are actively met.
  - Bug Tracking: proper bug reporting and handling plan – some bugs may remain in the code just because its too expensive to solve them!
  - Thread Modelling: a design technique used to communication information associated with a threat throughout the development team.

Secure SDLC – cont'd

- Common Components – cont'd:
  - Fuzzing: a test technique where tester applies a series of input to an interface in an automated fashion and examines the outputs for undesired behaviours.
  - Security Reviews: examining the process and ensure that the security-related steps are being carried out and not being circumvented to expedite process.
  - Mitigations: not all risks are equal ($risk = probability \times impact$), some bugs need fixing more than lesser-risky bugs.
- Security Features != Secure Software
  - Security features are elements of a program specifically designed to provide some aspect of security.
  - Secure software development is about ensuring that all elements of a software package is constructed in a fashion where they operate securely.
  - *Example - bug in Debian's random-number generator resulted in many cryptographic failures!*

Secure SDLC – cont'd

- Release Software Securely
  - When software has finished the development process, it still needs to be delivered to the customer "securely".
    - How to handle application secrets (e.g., database passwords), digital certificates, and etc.
  - "DevOps" is a form of software development and maintenance where changes are introduced virtually directly into production.
  - Secure "DevOps" is a process where the development is also responsible for all aspects of code including security all the way to production.

Security Standards and Frameworks

- The objective of using standards and frameworks is to lead to lower numbers and severities of vulnerabilities in released software.
  - OWASP top 10 (owasp.org) for secure web application development
  - BSA (bsa.org) https://www.bsa.org/policy-issues/cybersecurity
  - SAFECode (safecode.org) https://safecode.org/wp-content/uploads/2018/03/SAFECode_Fundamental_Practices_for_Secure_Software_Development_March_2018.pdf

Decommission Software

- Is the process of safely terminate the existence of a system or a software entity.
  - Every software system requires continuous maintenance and vulnerability patching.
  - At some point, it is not economically viable to offer support!

- End-of-Life Policies
  - The plan defines schedules, actions, and resources that terminate the delivery of software services; transform the system into, or retain it in, a socially and physically acceptable state.
  - This plan stipulates the steps that will be taken by the operation and maintenance organization to remove active support.

https://fdiinc.com/application-decommissioning-101-what-it-is-and-why-your-it-department-should-care/

Fundamental Security Design Principles

- Despite years of research and development, systematic exclusion of security flaws and prevention of all unauthorized action has not been possible.
- In the absence of such foolproof techniques, we use the following design principles as a guide to develop more secure and robust information systems:

(a) Economy of mechanism	(i) Psychological acceptability
(b) Fail-safe defaults	(j) Isolation
(c) Complete mediation	(k) Encapsulation
(e) Open design	(l) Modularity
(f) Separation of privilege	(m) Layers
(g) Least privilege	(n) Least astonishment
(h) Least common mechanism	

Fundamental Security Design Principles – Cont'd

a) Economy of mechanism means the design of security measures embodied in both hardware and software should be as simple and small as possible. Simple and small design is easier to test and verify thoroughly.

b) Fail-safe defaults means access decision should be based on permission rather than exclusion. The default situation is lack of access, and the protection scheme identifies conditions under which access is permitted.

c) Complete mediation means every access must be checked against the access control mechanism and do not rely on access decision retrieved from a cache.

d) Open design means the design of a security mechanism should be open rather than secret – NO SECURITY BY OBSCURITY.

e) Separation of privilege is defined as a practice in which multiple privilege attributes are required to achieve access to a restricted resource (e.g., multi-factor authentication).

f) Least privilege means every process and every user of the system should operate using the least set of privilege necessary to perform the task.

Fundamental Security Design Principles – Cont'd

g) Least common mechanism means the design should minimize the functions shared by different users, providing mutual security.

h) Psychological acceptability implies the security mechanisms should not interfere unduly with the work of users, and at the same time meet the needs of those who authorize access.

i) Encapsulation provides protection by encapsulating a collection of procedure and data objects in a domain of its own so that internal structure of a data object is accessible only to the procedure of the protected subsystem.

j) Modularity refer both to the development of secure functions as separate, protected modules, and to the use of modular architecture for mechanism design and implementation.

k) Layering refer to use of multiple, overlapping protection approaches addressing the people, technology, and operational aspects of information systems.

l) Least astonishment means a program or user interface should always respond in the way that is least likely to astonish the user.

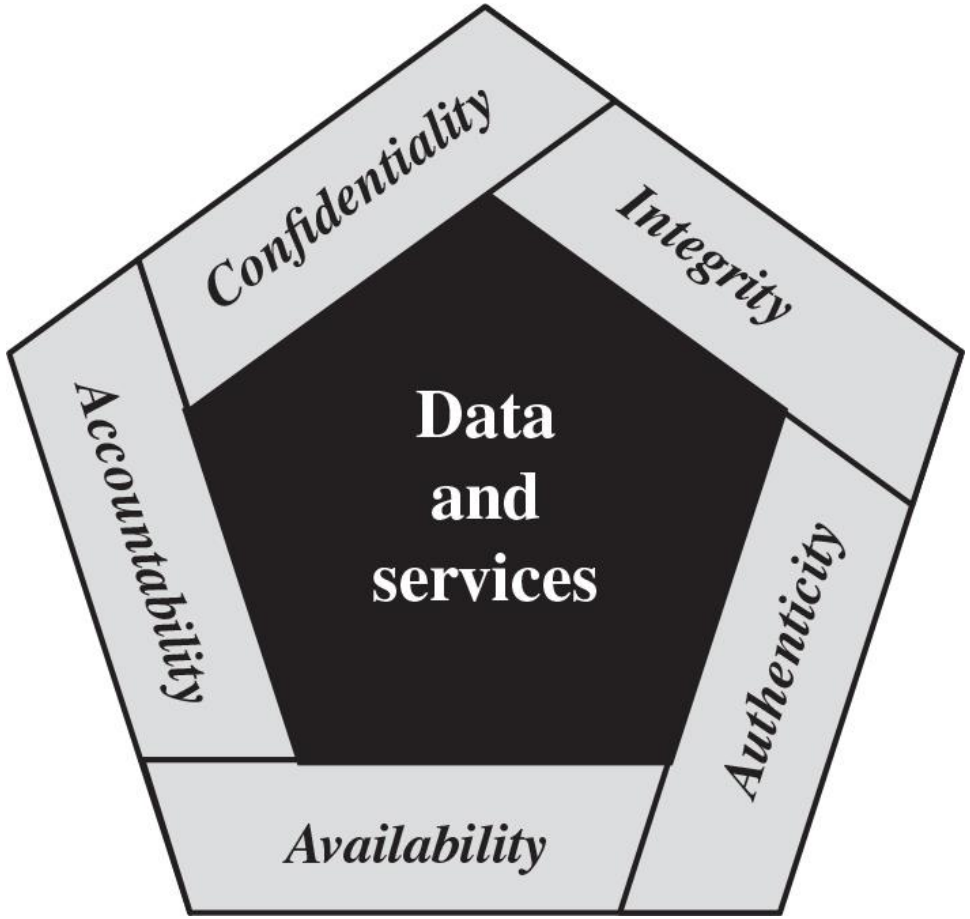
CIA Triad

- C.I.A. triangle represents 3 key characteristics of information that must be protected by information security and software systems involved:
  - **Confidentiality**
    - *Only authorized parties can view private/confidential information.*
  - **Integrity**
    - *Information is changed only in a specified and authorized manner.*
  - **Availability**
    - *Information is accessible to authorized users whenever needed.*

Extended CIA Triad

- Some security experts feel that additional concept need to be added to the CIA triad:
  - **Authenticity**
    - *Being genuine and being able to be verified and trusted. Verify identity of participating parties.*
  - **Accountability**
    - *Being able to trace actions of an entity uniquely to that entity.*

  ** Because truly secure systems are not yet an achievable goal, we must be able to trace a security breach to a responsible party.*



Data States

1. **Data at Rest**
   - Data being stored in memory or on disk.
     - The emphasis is on “stored” – you may also load data into the memory before “using” it!
     - How can you tell if a data in the volatile memory is in the “rest” state or “in use” state? [in class discussion]
     - Data at rest can/may be encrypted in order to ensure their confidentiality and integrity.
2. **Data in Transit**
   - Data being transferred between systems, in physical or electronic form.
3. **Data in Use**
   - Data being actively examined or modified (usually decrypted).

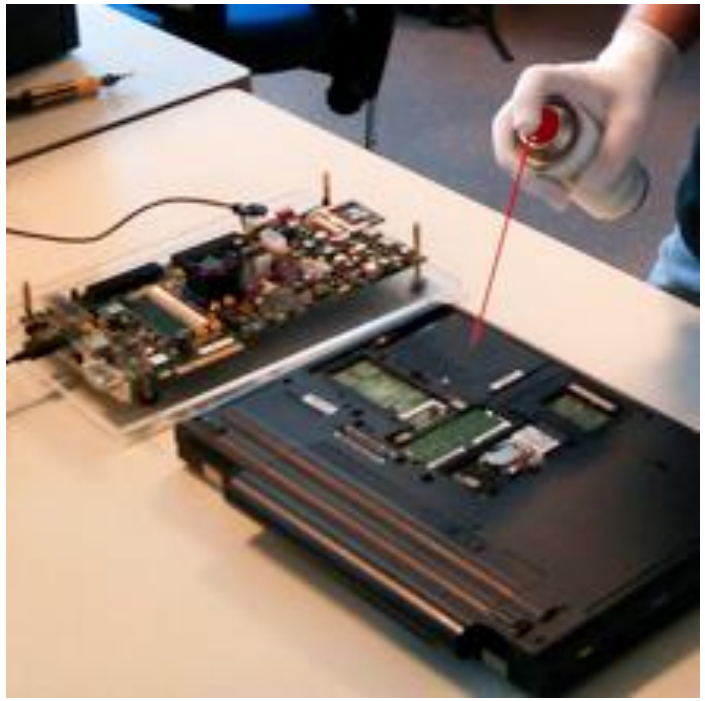
Data States – cont'd

- Data in-use is the most challenging to protect.
  - Unencrypted data can be read and understood by unauthorized parties.
  - Most of processing/computing algorithms can only run against unencrypted data.
  - Therefore, data in-use is in the most vulnerable state, i.e., must be decrypted before processing.

  - Homomorphic Encryption
    - Is a form of encryption that permits computing/processing on the encrypted data without first decrypting it.
    - Currently under development and in its current form is extremely slow (30 minutes per basic bit operation, e.g., single AND gate).
    - Outside of the scope of this course. Interested? Check this link out: https://palisade-crypto.org/

Data States – cont'd

- Coldboot attack is an example of sophisticated against data-in-use and why physical security is important. [In class discussion]



https://pc2.uni-paderborn.de/research/publications/open-source-projects/hardware-accelerated-cold-boot-attacks/

https://www.bleepingcomputer.com/news/security/cold-boot-attack-steals-passwords-in-under-two-minutes/

Attack Surfaces

- Consist of the reachable and exploitable vulnerabilities in a system
- Examples:
  - Open ports on outward facing Web and other servers, and code listening on those ports
  - Services available on the inside of a firewall
  - Code that processes incoming data, email, XML, office documents, and industry-specific custom data exchange formats
  - Interfaces, SQL, and Web forms
  - An employee with access to sensitive information vulnerable to a social engineering attack

Attack Surface Categories

- **Network Attack Surface**
  - Vulnerabilities over an enterprise network, wide-area network, or the Internet.
  - Included in this category are network protocol vulnerabilities, such as those used for a denial-of-service attack, disruption of communications links, and various forms of intruder attacks.
- **Software Attack Surface**
  - Vulnerabilities in application, utility, or operating system code.
  - Particular focus is Web server software.
- **Human Attack Surface**
  - Vulnerabilities created by personnel or outsiders, such as social engineering, human error, and trusted insiders.

Software Supply Chain Risk Management

- Like any other product, software has a supply chain.
  - Secure issues can arise in the supply chain and manifest as a risk in the software being produced.
  - You need t employ consistent, disciplined process that ensures measurable product integrity.

- Software Bill of Materials (SBOM)
  - is a listing of the provenance and lineage of all components in a software, including libraries, third-party code, and internally generated code.
  - Notable examples:
    - Heartbleed and OpenVPN [in class discussion]
    - SQL Slammer worm affecting any software using SQL Engine! [in class discussion]