1 Security and Software Development

Ibéria Medeiros

Departamento de Informática

Faculdade de Ciências da Universidade de Lisboa

Android Supply Chain Infecting Mobiles with Malware Before Delivery (March 2017)

Certain Android smartphones are being infected with malware somewhere along the supply chain. The affected brands include Samsung, Xiaomi, Asus, LG, ZTE, and Lenovo. Several types of malware have been found on the devices: *data stealers, malicious advertisement displayers, and ransomware*.

Basic Concepts

Basic concepts

Three (plus one) security attributes

- Confidentiality
- Integrity
- Availability
- *Authenticity*

Also, called the CIA attributes

- "non-authorized" requires a *security policy*
  - ☞ defined in an explicit or implicit way

Vulnerabilities

- **Vulnerability:** a system (hw/sw) defect relevant security-wise
  - ☞ can be exploited by an attacker to subvert the security policy

- They are defects but

  "But some team leaders were not playing along. They were unwilling to relax schedules or use the productivity factor to offset the increased time needed to fix identified vulnerabilities [using static analysis tools]. They preferred to allocate time and budget to creating functionality rather than improving the vulnerability risk score of the their team's code. In effect, the team leaders conveniently assumed that security vulnerabilities were **not** defects and could be deferred for future enhancements or projects."

  – Jim Routh in "Beautiful Security", pg 189, 2010

Types of vulnerabilities

- **Design vulnerability**: inserted during the software design (e.g., weak authentication mechanism, forget that communication can be listened in the network)
- **Coding vulnerability**: a bug introduced during coding with security implications (e.g., missing end of buffer verification)
- **Operational vulnerability**: caused by the environment in which the software is executed or its configuration (e.g., database accounts with empty password)

Attacks and Others

- **Attack**: malicious action that attempts to exploit one or more vulnerabilities

(Attack + Vulnerability) → Intrusion

- **Exploit**
  - ☞ an attack directed to a certain vulnerability **or**
    a piece of code that can be used to activate a vulnerability

- **0-day vulnerability**: a vulnerability not publicly known (only a group of people has the knowledge that it exists; sold in dark-web marketplaces)

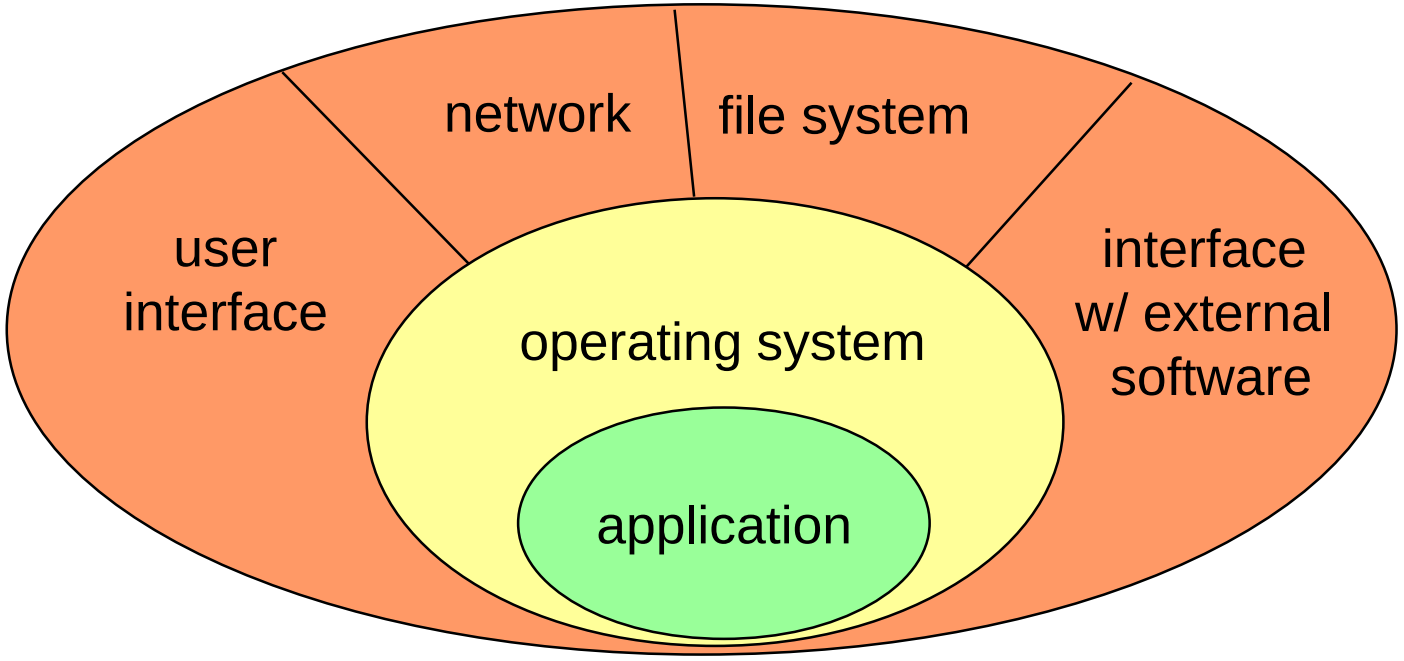
- **Patch**: a software that fixes a defect and therefore should remove the vulnerability

Databases about Vulnerabilities

- Classification: organize vulnerabilities in classes
  - Common Weakness Enumeration — cwe.mitre.org
- Catalog: identifies & describes vulnerabilities
  - Common Vulnerabilities and Exposures — cve.mitre.org
  - National Vulnerability Database — nvd.nist.gov
  - Bugtraq and others — securityfocus.com
  - Exploit database — www.exploit-db.com

Attack surface

- Attacks come through interfaces, which correspond to the attack surface
  - ☞ *Analogy*: think about a fort … how can you enter?



Attack Vector

- The way an attack is deployed, its features
- Can be used in several senses…
  - ☞ Type of vulnerability (BO, SQL injection,…)
  - ☞ How the attack is performed (virus, worm,…)
- Can be directed or not
  - ☞ e.g., against a specific company or the Internet in general
- Can be technical vs. social engineering
- Can be manual vs. automated
  - ☞ the separation is not so simple (they may initially be manual and then is made automatically)

Software Development

Software development

- The goals of security are often contradictory with other objectives of software development
  - ☞ *Functionality*: more is better
  - ☞ *Usability*: easier the better
  - ☞ *Performance*: higher the better
  - ☞ *Simplicity*: more is better
  - ☞ *Time-to-market*: faster the better
    - Windows Security Push:

      "*During February and March 2002, all feature development on Windows products at Microsoft stopped so that the complete Windows development team could analyze the product design, code, test plans, and documentation for security issues*."

Risk – The Ultimate Goal

- Typically, the objective is (not to achieve 100% security but) to have an acceptable risk

Probability of successful attack = Threat level x Vulnerability level

Risk = Probability of successful attack x Impact

- Risk depends on the system environment
  - ☞ all software is 100% secure in a computer locked in a safe with no Internet connection
  - ☞ basic arithmetic: T*0*I=0 0*V*I=0 T*V*0=0

Factors Affecting the Vulnerability Level

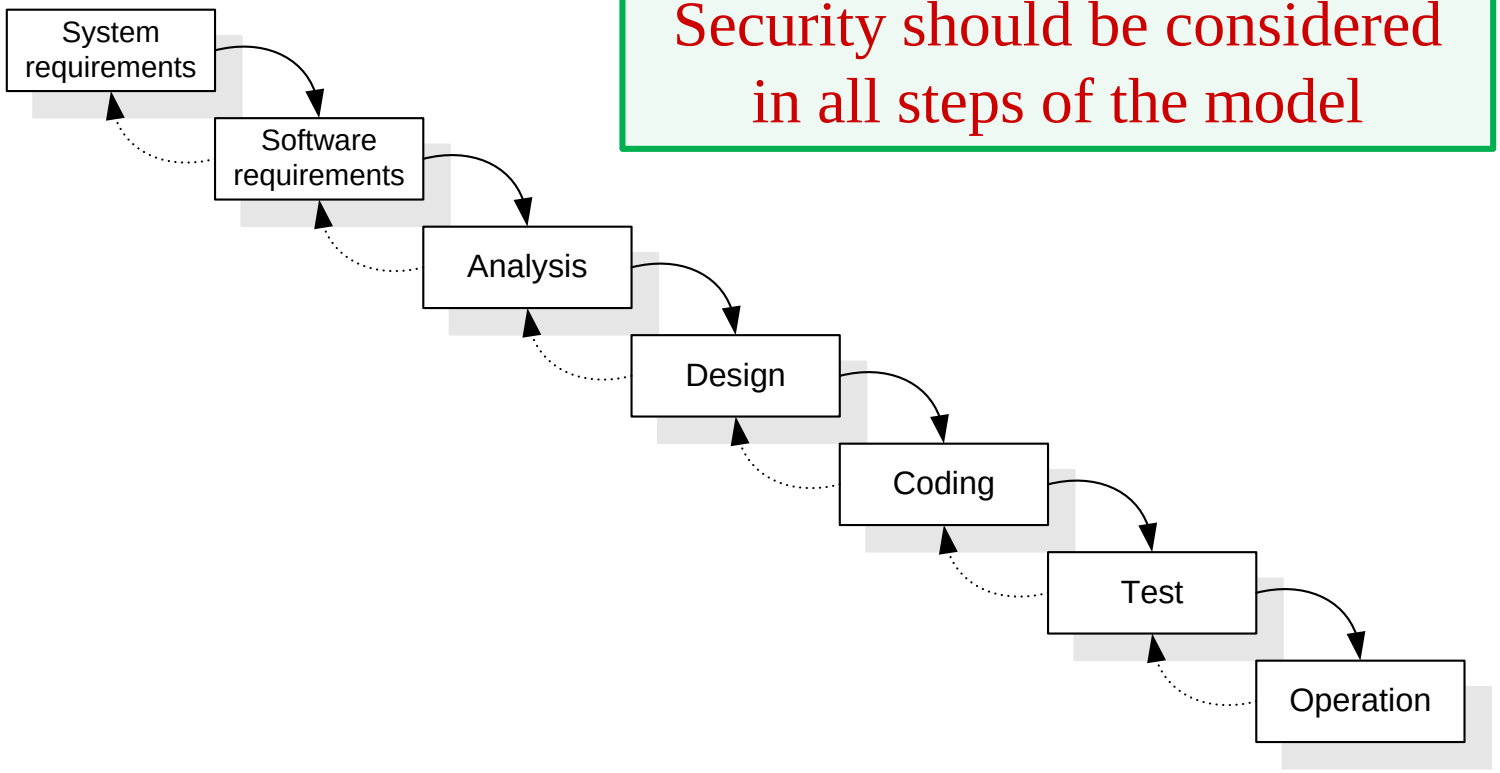
- The kinds of vulnerabilities that exist in the software (design/coding/operational)
- The implemented security controls
- Maturity level of the software development processes used in the company
- Uncertainty of the above factors

Software development life cycle

- How shall security be introduced in the software development cycle?

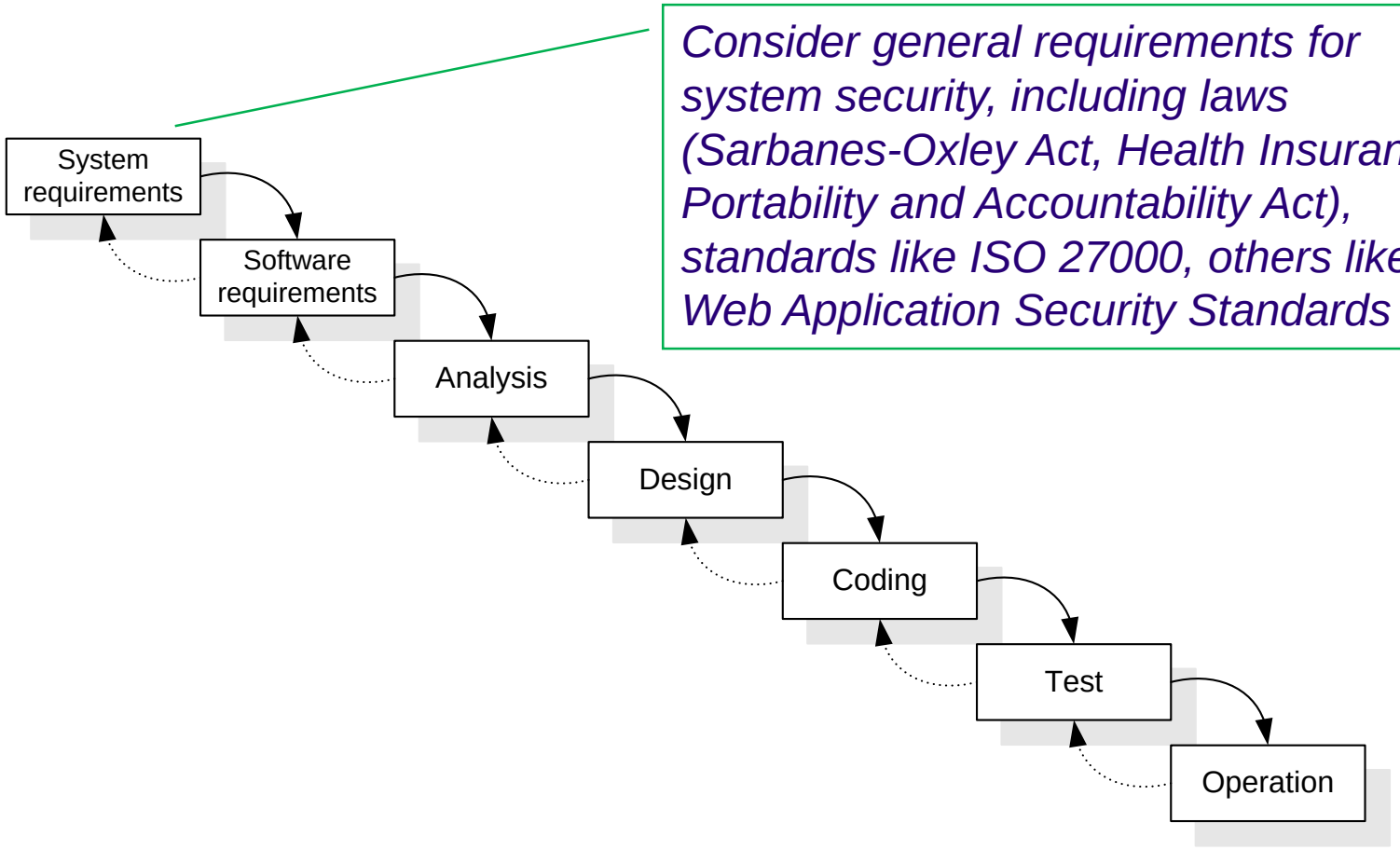
- Let us see some software development processes …
  - ☞ Waterfall model
  - ☞ Spiral model
  - ☞ Microsoft SDL

Waterfall model



Security should be considered in all steps of the model

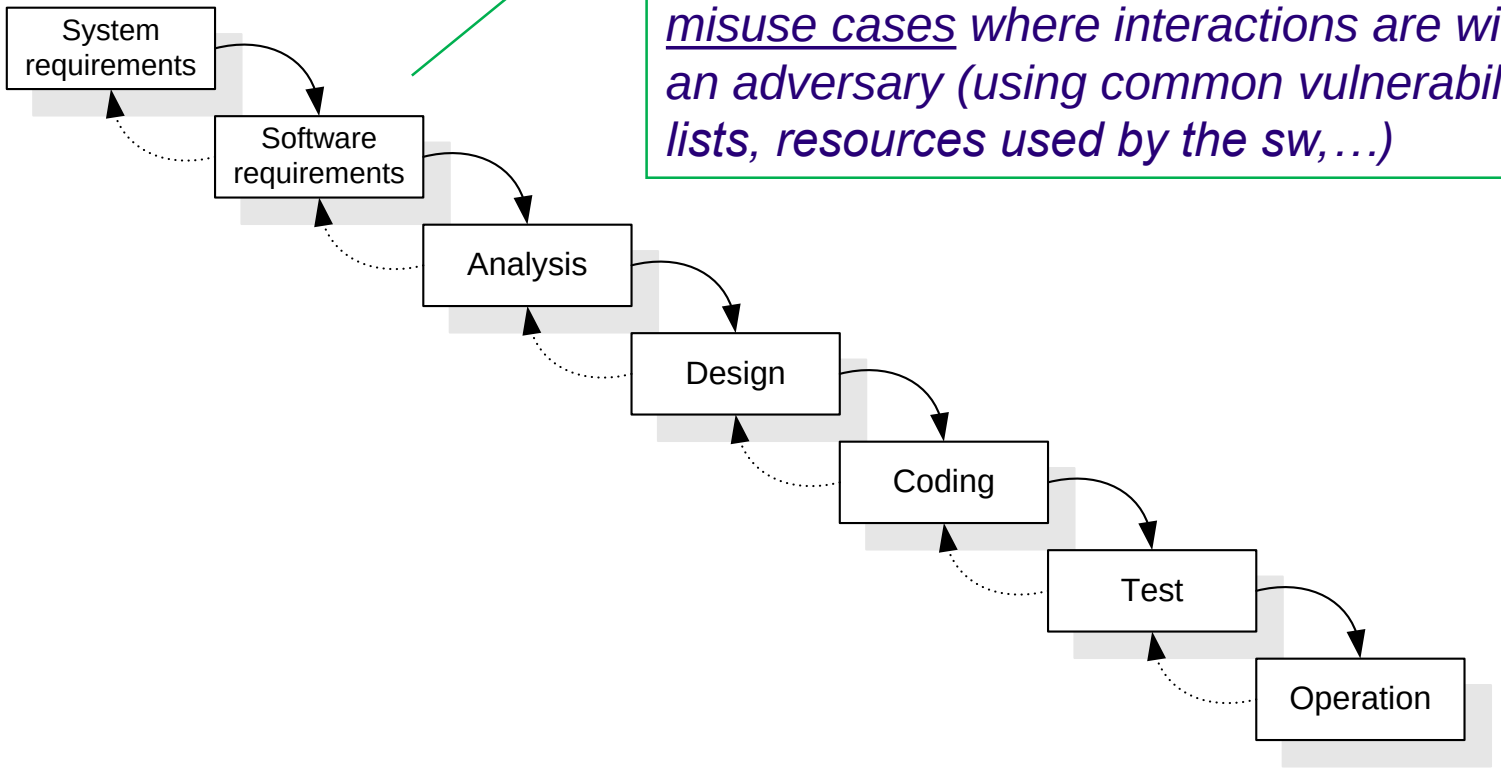
Waterfall model



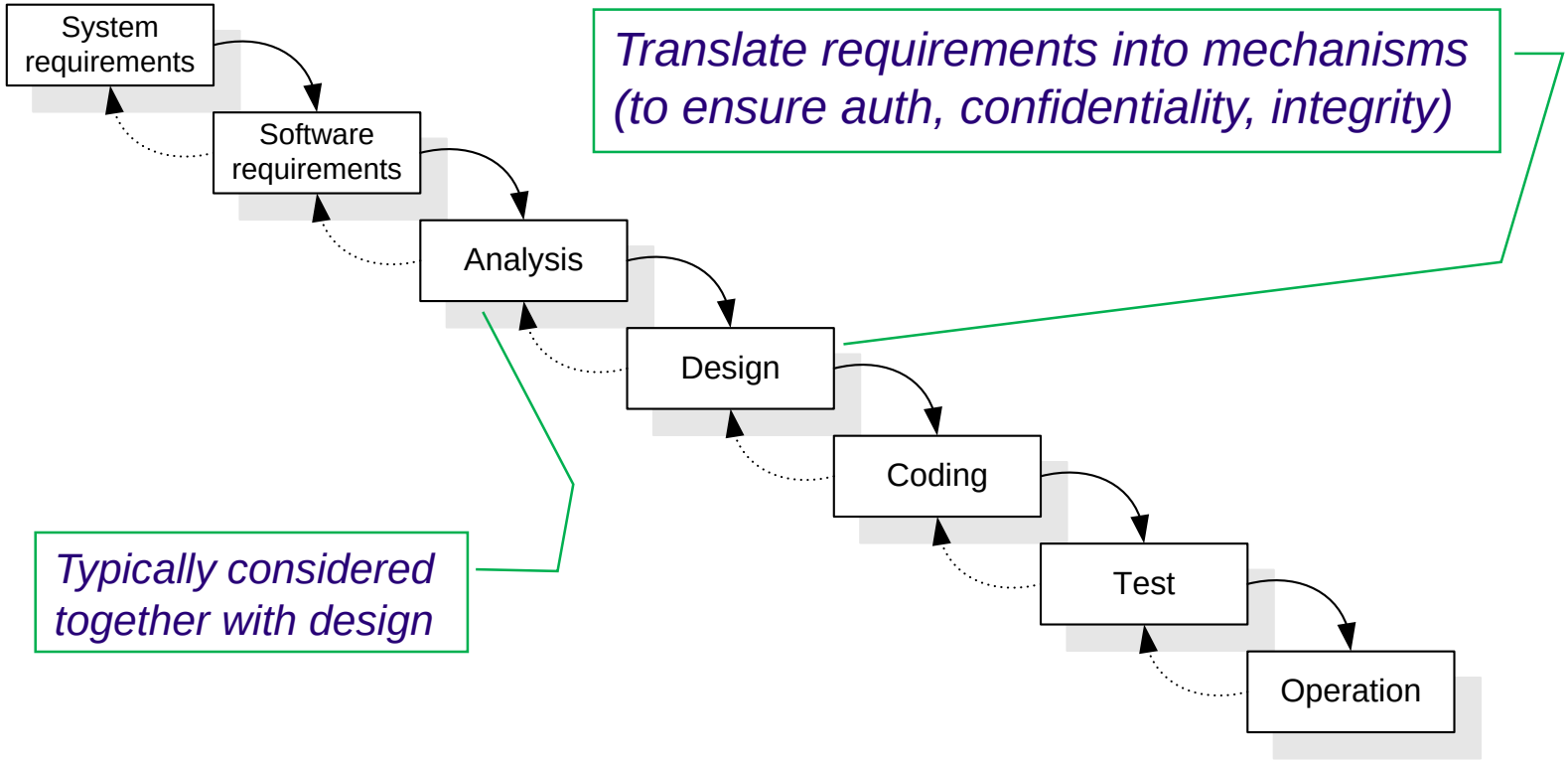
Consider general requirements for system security, including laws (Sarbanes-Oxley Act, Health Insurance Portability and Accountability Act), standards like ISO 27000, others like the Web Application Security Standards

Waterfall model

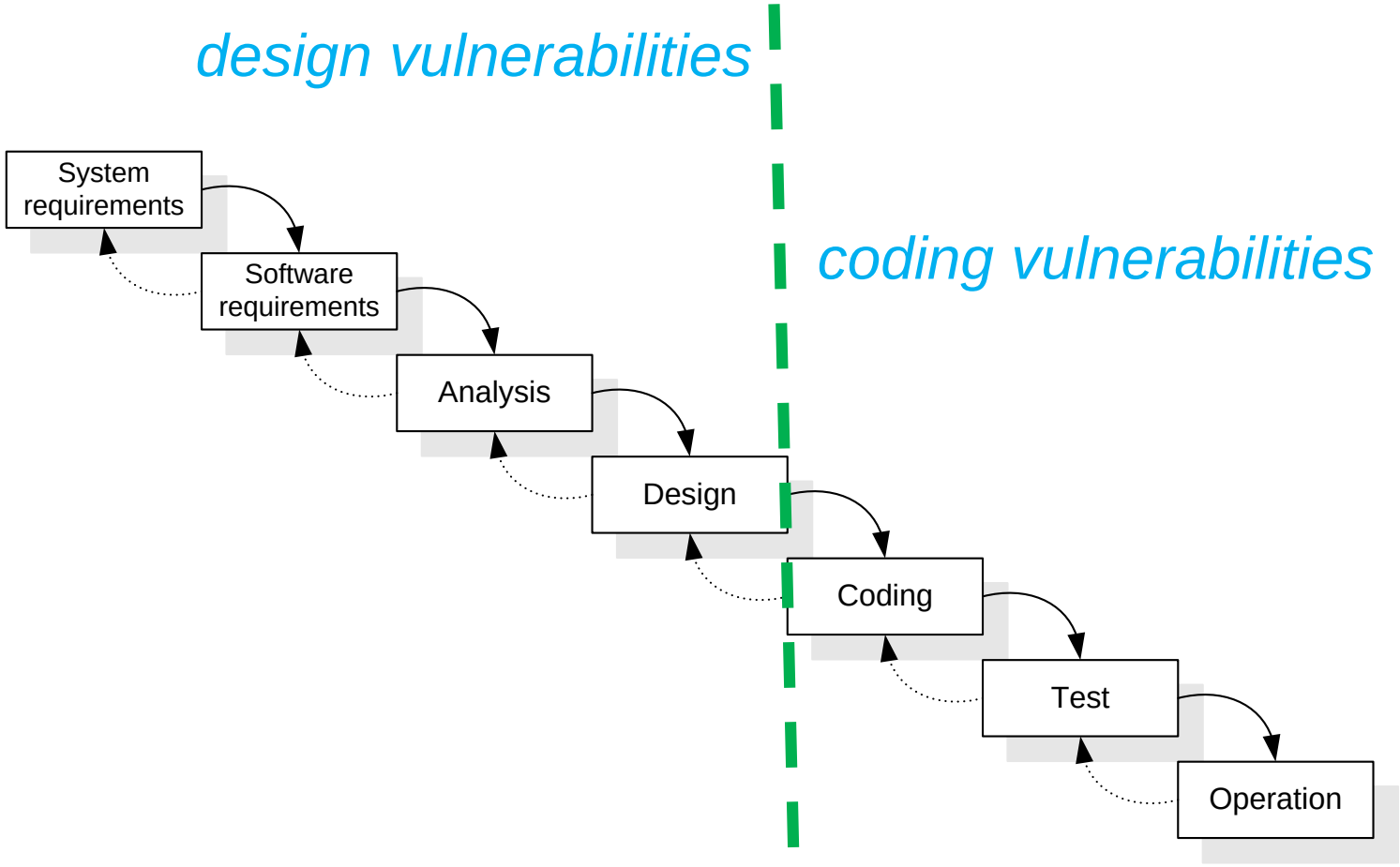
Translate generic requirements into specific software requirements. Can be done using misuse cases where interactions are with an adversary (using common vulnerability lists, resources used by the sw,…)



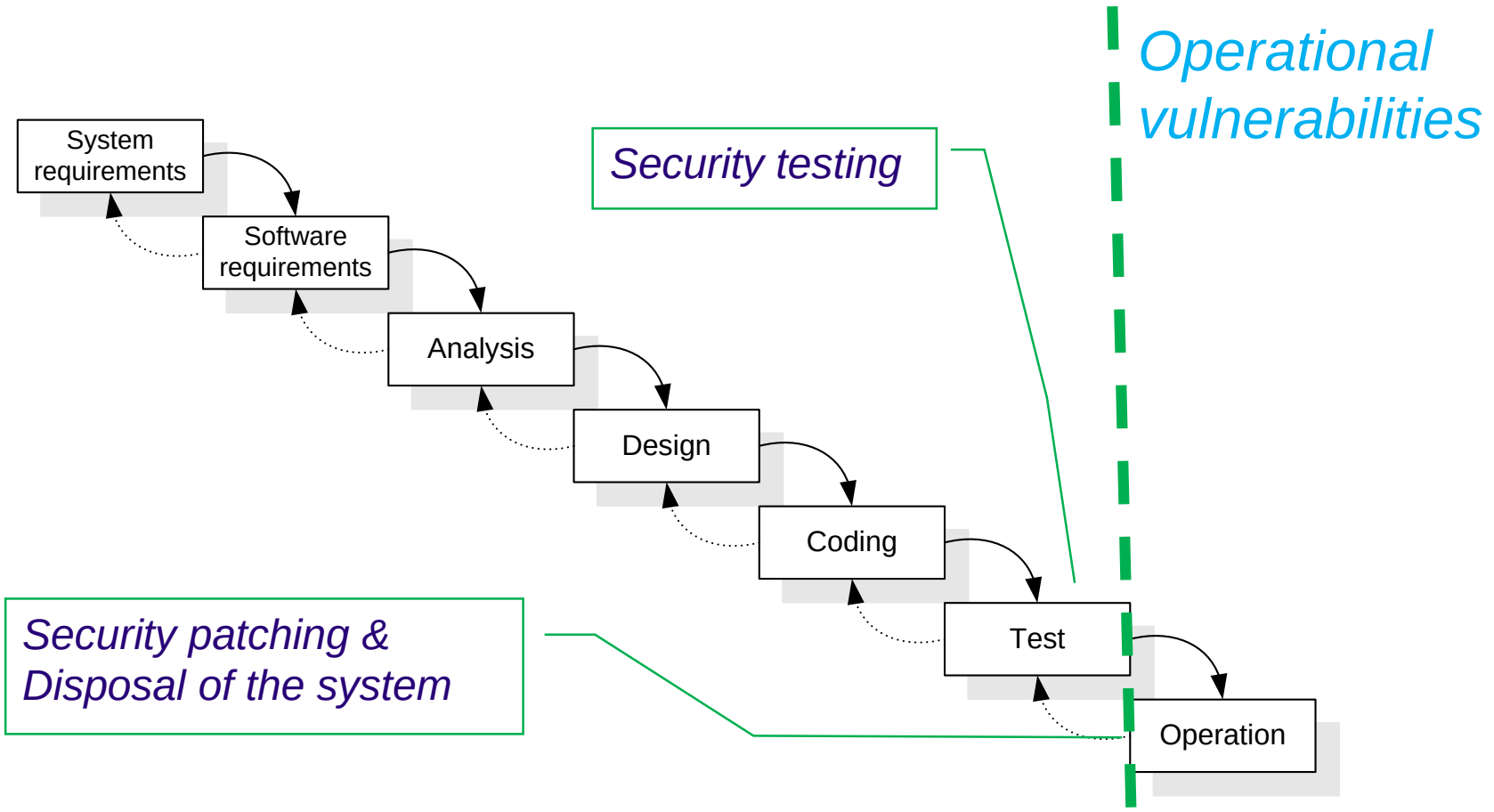
Waterfall model



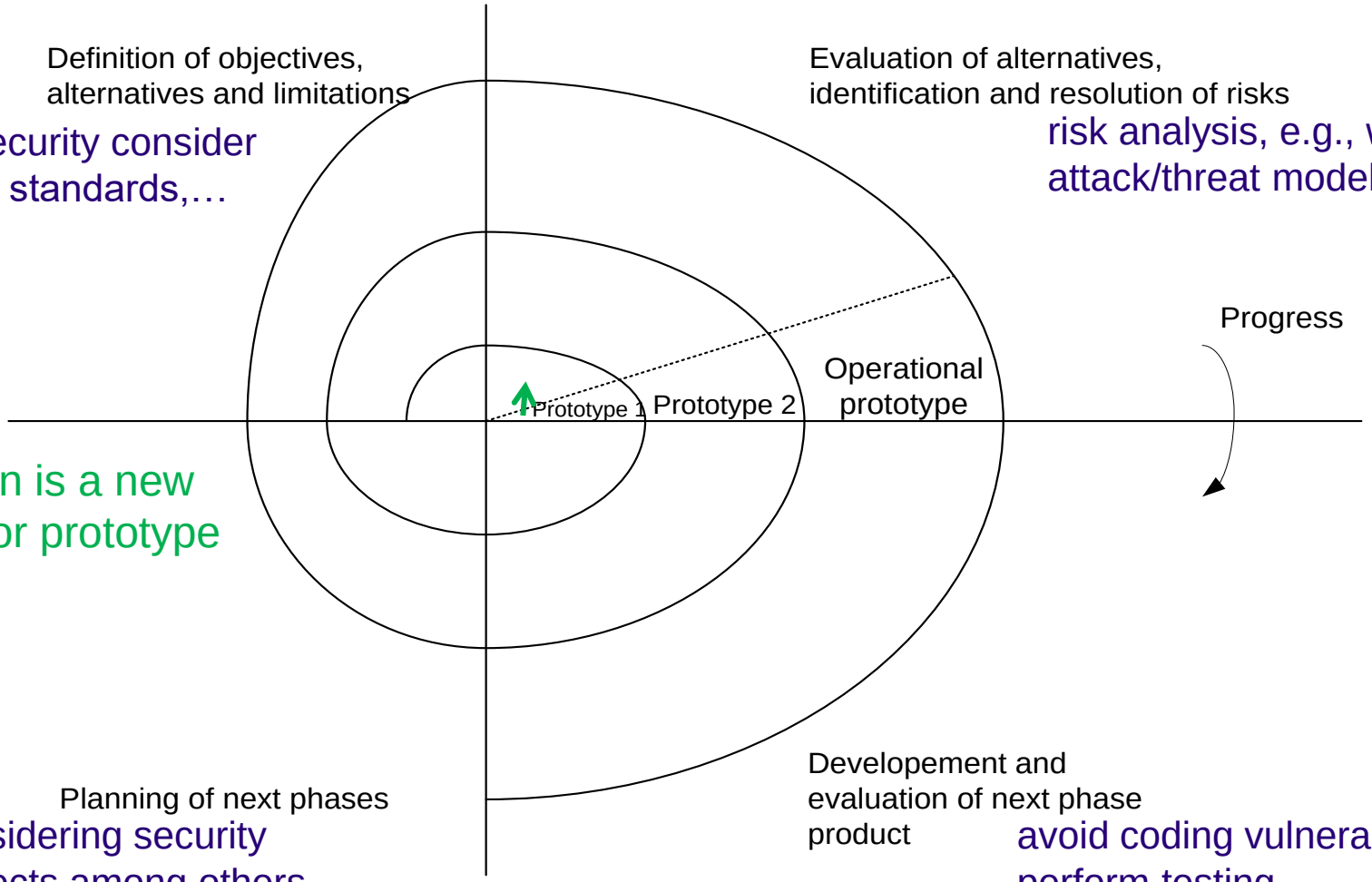
Waterfall model



Waterfall model



Spiral model



for security consider laws, standards,…

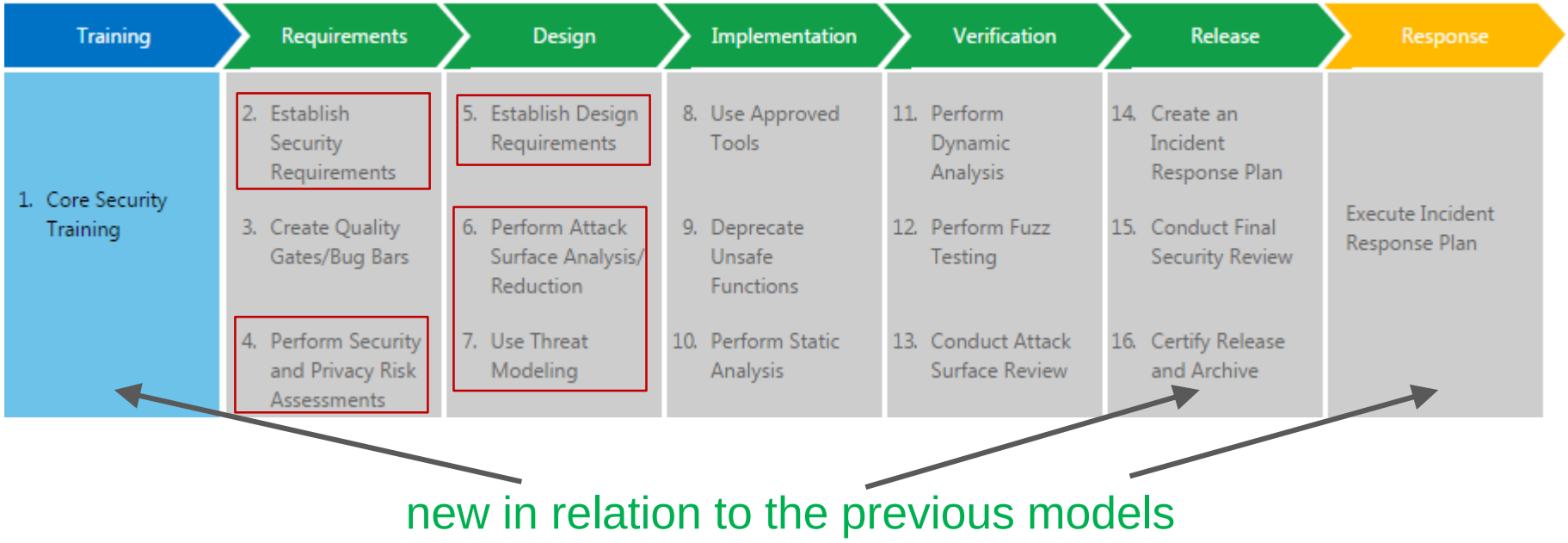
risk analysis, e.g., with attack/threat modeling

Each turn is a new version or prototype

considering security aspects among others

avoid coding vulnerabilities, perform testing

Microsoft Security Development Lifecycle



new in relation to the previous models

- 6- Final review is external and independent (e.g., by another team)
- 7- Response execution involves collecting information about security events, issuing reports and patches

Microsoft SDL-Agile

every 2-4 weeks

EVERY-SPRINT PRACTICES	BUCKET PRACTICES	ONE-TIME PRACTICES

1. TRAINING	2. REQUIREMENTS	3. DESIGN	4. IMPLEMENTATION	5. VERIFICATION	6. RELEASE	7. RESPONSE
1. Core Security Training	2. Establish Security Requirements	5. Establish Design Requirements	8. Use Approved Tools	11. Perform Dynamic Analysis	14. Create an Incident Response Plan	
	3. Create Quality Gates/Bug Bars	6. Perform Attack Surface Analysis/ Reduction	9. Deprecate Unsafe Functions	12. Perform Fuzz Testing	15. Conduct Final Security Review	17. Execute Incident Response Plan
	4. Perform Security and Privacy Risk Assessments	7. Use Threat Modelling	10. Perform Static Analysis	13. Conduct Attack Surface Review	16. Certify Release and Archive	

7 – apply a structured approach based on the analysis of threat scenarios for new functionalities;

8 - use approved tools configured with security in mind;

9 – ban unsafe APIs;

10 – analyze source code prior to compilation;

15 - review all security activities;

16 - certify that security requirements were meet and archive everything

Microsoft SDL-Agile

every bucket

EVERY-SPRINT PRACTICES	BUCKET PRACTICES	ONE-TIME PRACTICES

1. TRAINING	2. REQUIREMENTS	3. DESIGN	4. IMPLEMENTATION	5. VERIFICATION	6. RELEASE	7. RESPONSE
1. Core Security Training	2. Establish Security Requirements	5. Establish Design Requirements	8. Use Approved Tools	11. Perform Dynamic Analysis	14. Create an Incident Response Plan	
	3. Create Quality Gates/Bug Bars	6. Perform Attack Surface Analysis/ Reduction	9. Deprecate Unsafe Functions	12. Perform Fuzz Testing	15. Conduct Final Security Review	17. Execute Incident Response Plan
	4. Perform Security and Privacy Risk Assessments	7. Use Threat Modelling	10. Perform Static Analysis	**13. Conduct Attack Surface Review**	16. Certify Release and Archive	

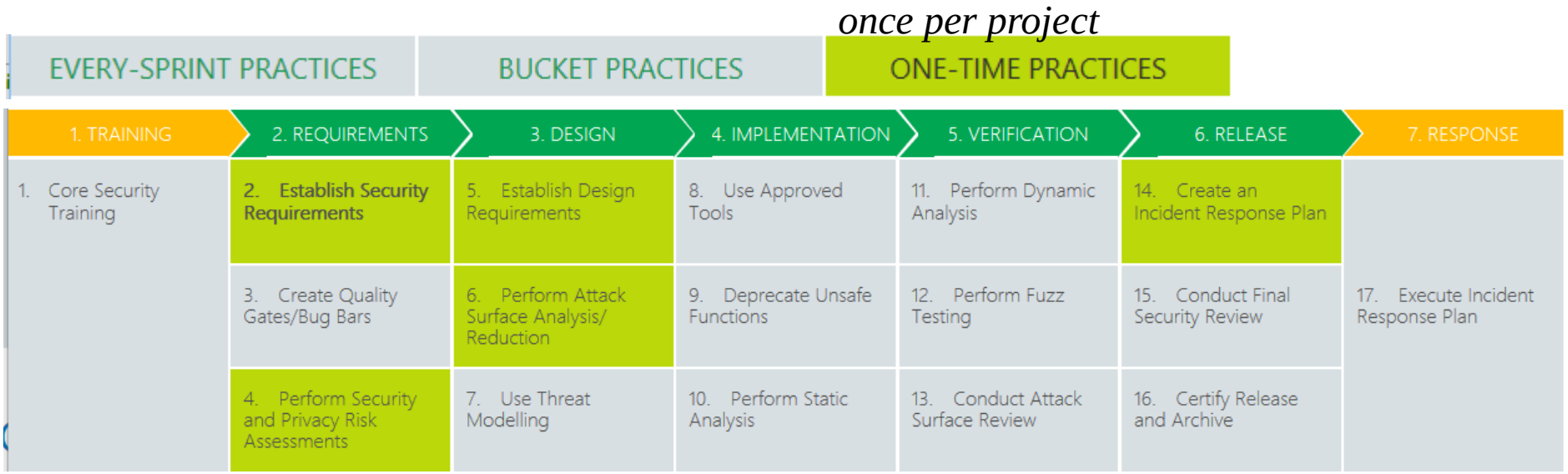
3 – define minimum acceptable levels of security and privacy quality;

11 – use tools to verify the execution of the code at run time (e.g., memory leaks);

12 – induce malicious data to the application;

13 - review attack surface and ensure that changes in the application have been taken into account

Microsoft SDL-Agile



2 – define security and privacy requirements early on;

4 – examine software design with respect to regulatory requirements and identify main risks;

5 – address security and privacy concerns during design;

6 – analyze and reduce the attack surface if possible, use least privilege and layered defenses;

14 – prepare a plan on how to deal with new threats, identify emergency contacts

Other Factors that affect security

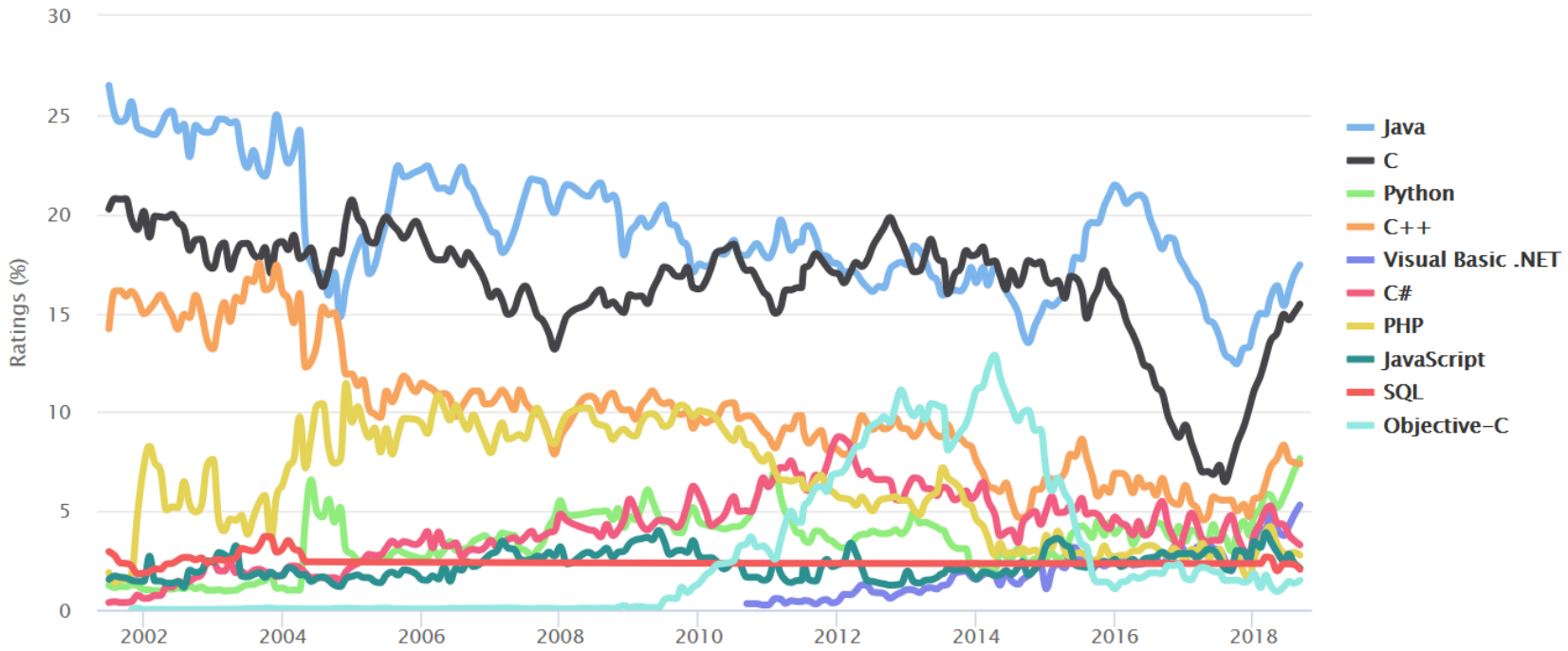
- Programming language
- Closed vs. Open source

Selecting a language

- The language has an impact on security so it should be selected taking this aspect into account
  - ☞ usually other factors have more weight: comfort with the language, performance, personal taste…
- C/C++ are fast but prone to vulnerabilities
  - ☞ pointers, lack of array bound verifications
- Java/C# are not perfect but limits these problems
  - ☞ no pointers, array bounds checked in runtime
  - ☞ programs run in a sandbox (enforces access control policies)
  - ☞ but there are also vulnerabilities, default policies, etc
- Perl is slower but has interesting security features
  - ☞ *taint mode* monitors variables in runtime to see if untrusted user input leads to a security violation

More popular, more secure?

- TIOBE Programming Community Index
  - gives an indication of the popularity of programming languages
  - based on the world-wide availability of skilled engineers, courses and third party vendors
  - popular search engines Google, MSN, and Yahoo! are used to calculate the ratings
  - not about best programming language or language in which most lines of code have been written



Open source vs closed source

- An eternal debate… and maybe it is not really important!
- *"Open source is much better because many-eyeballs see the code and detect vulnerabilities"*
  - ☞ but do they look?
  - ☞ can they see?
- *"Closed source is much better because it is harder for the attacker to find vulnerabilities"*
  - ☞ that's why so few vulnerabilities are found in commercial applications ☺
  - ☞ security by obscurity is not a good idea

Design Principles to Avoid Vulnerabilities

Design principles

- Objective is to avoid introducing design vulnerabilities
- Saltzer and Schroeder in 1975 listed 8 design principles for secure (access control) systems
  - ☞ they remain entirely valid
  - ☞ should always be kept in mind when designing sw systems
  - ☞ they are specially important for the design of protection mechanisms

- *There are others, more recent also ….*

  … so, lets have a look

From your book

1. Never Assume or Trust

- Systems built out of several components are vulnerable if one of them has weaknesses
- Trusting the client, by moving part of the functionality of the server to the client
  - ☞ authentication or access control decisions at the client
  - ☞ trust the client to perform the validation of user inputs
  - ☞ expecting ***thick clients*** to always operate correctly
- Place secrets or intellectual property relevant information at a remote entity (client or server)

2. Use Authentication Mechanisms that Cannot be Circumvented

- Ensure there are appropriate measures to authenticate users
  - ☞ multi factor authentication
- Prevent (authenticated) users from changing identity
  - ☞ if the authentication procedure produces a token associated to the user, it should be impossible to derive that token
  - ☞ it should not possible to leave behind the token for others
- Perform authorization only after authentication
  - ☞ different users have distinct access rights

3. Identify and Protect Sensitive Data

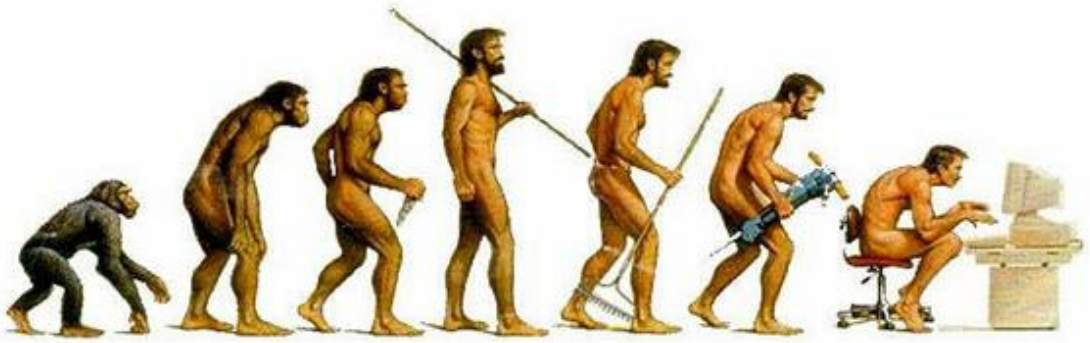
- Sensitive data depends on the context
  - ☞ regulations, contracts, user expectations
  - ☞ changes with time
  - ☞ can be different in diverse countries
- Steps to protect sensitive data
  - ☞ **Identify** – the sensitive data
  - ☞ **Discover** – location and accessibility
  - ☞ **Classify** – data accordingly to its value to the organization
  - ☞ **Secure** - employ appropriate controls to ensure integrity, availability, and confidentiality
  - ☞ **Monitor** – measure and evolve security practices

4. Understand External Components

- Systems are built out of components, and therefore they have an impact on the attack surface
  - ☞ the system inherits the weaknesses of the external components
- To minimize the impact
  - ☞ isolate the external components (e.g., in a sandbox)
  - ☞ activate only the necessary functionalities (e.g., in the configuration)
  - ☞ validate the integrity of the component (e.g., signatures)
  - ☞ maintain list of external components
  - ☞ authenticate / validate data coming / going to the components

5. Be Prepared for System Evolution

- Successful systems need to evolve over time
  - ☞ changes where it is executed, threats are different, functionality is added
- Areas that have to be considered
  - ☞ secure updates require, e.g., signed patches and their validation
  - ☞ capacity to activate or deactivate functionalities, e.g., that are perceived insecure
  - ☞ change the secrets, e.g., passwords, password recovery mechanisms, stored data
  - ☞ choose another external component, e.g., because cripto algorithm is no longer secure



Saltzer and Schroeder, 1975

1. Economy of mechanism

- "Keep the design [of the security mechanisms] as ***simple and small*** as possible."

  - ☞ systems fail because they are large and complex
  - ☞ security mechanisms should not fail so ...
  - ☞ assurance techniques (e.g., manual code auditing, static analysis) are not perfect, but are much more effective if applied to something simple and small

2.Fail-safe defaults

- "Base access decisions on [giving] permission rather than [defining the] exclusion."

  - ☞ default situation should be ***no access***
  - ☞ the protection mechanisms should decide on when to give access, not on when to deny access
  - ☞ if there is a mistake, at most someone will be refused access
  - ☞ examples:
    - huge number of vulnerabilities due to "default passwords" (easy to find in the web)
    - the access to resources should be provided based on an explicit argument that justifies the need, not by default

3.Complete mediation

- "Every access to every object must be checked for authority"
  - ☞ i.e., there should be no way to circumvent access control
  - ☞ a foolproof method of identifying the source of every request must be devised

4. Open design

- "The design should not be secret."
  - ☞ the achieved security should not be based on secrecy (except for small, special secrets like passwords and keys)
  - ☞ the examination by many reviewers does not compromise security
  - ☞ ***no security by obscurity***

- Obscurity does not work
  - ☞ there are many reverse-engineering tools
  - ☞ there are many automated ways for looking for vulnerabilities
  - ☞ break it once, exploit it many times

5.Separation of privilege

- "A protection mechanism that requires two keys to unlock is more robust and flexible than one that allows access to the presenter of only a single key."
  - ☞ a single compromise does not break security
  - ☞ analogies from normal life: two signatures in bank checks; two keys in safe-deposit boxes ...
  - ☞ for authentication, currently called multi-factor authentication
    - (username and password) + possession of a card
  - ☞ Somewhat related, ***security in depth***

6.Least privilege

- "Every program and every user of the system should operate using ***the least set of privileges*** necessary to complete the job."

  - ☞ limits the damage from accidental faults
  - ☞ limits the damage from malicious faults (intrusions, malicious sw,...)

  - ☞ The military security rule of "***need to know***" is an instantiation of this principle

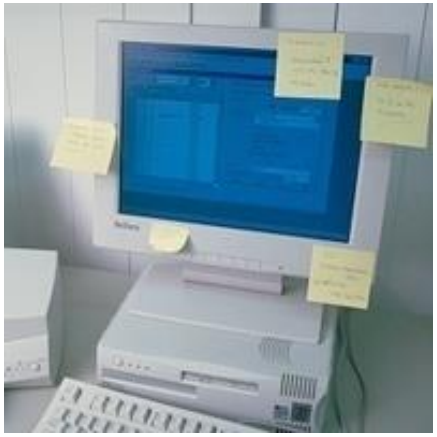
7.Least common mechanism

- "Minimize the amount of mechanisms common to more than one user **and** depended on by all users."

  - ☞ because every shared mechanism (e.g., shared variables, files, devices…) is a potential information path that may compromise security
  - ☞ considerable concern today due to multi-tenancy / cloud computing, where processes or virtual machines of different organizations run in the same physical machine

8.Psychological acceptability

- "It is essential that the human interface be designed for ease of use"
  - ☞ so that users routinely and automatically apply the protection mechanisms correctly
  - ☞ so that they understand that when the mechanism is properly used, they will benefit with an increased security
  - ☞ Corollary: security mechanisms should not make resources (much) harder to access than when they do not exist



Design Principles by Microsoft SD3+C

1. Secure by **D**esign
   - ☞ the overall system should be built to be secure
2. Secure by **D**efault
   - ☞ software shall be developed considering the possibility of intrusion
   - ☞ implies for instance enforcing the principle of least privilege and not running unnecessary services
3. Secure in **D**eployment
   - ☞ the software shall contain tools and documentation that allow proper security configuration and patching
4. **C**ommunication
   - ☞ between the developers and end-users about security aspects
   - ☞ patches must be provided and installed by the users

Other Principles: Consider Security From Start

- "build it first, secure it later" ☹
  - ☞ "We are very concerned about security. However the release deadline arrived and we didn't have time bla bla"
- Early design decisions have important security implications
  - ☞ two equally sound paths may lead to two very different systems in terms of security
  - ☞ add security later may have large costs and involve major redesign
  - ☞ it may also involve discarding functionality that had a cost to be developed

Secure the Weakest Link

- Security is a chain as strong as its weakest link…
- “low hanging fruit”
- part of risk assessment (e.g., attack modeling)
- protect everything usually not possible – prioritize!!!

Fail Securely

- Many systems ***become less secure when they fail***
  - ☞ multibanco has a maximum limit of cash that can be withdraw daily; however, if ATM is disconnected what should we do?
  - ☞ **downgrade attacks**: example … if the client does not support the latest version of the protocol, should the server use an older one?

- Always check for error conditions and handle them properly

Bibliography

- M. Correia, P. Sousa, Segurança no Software, FCA Editora, 2017 (see chapters 1-2)