universidade de aveiro

Critical software

Last Topics & Test Preparation

Robust Software – Nuno Silva

Mestrado em Cibersegurança

Agenda

- Side-channels
- DevSecOps
- Course Summary
- Test Preparation
- References

Side Channels

- A side-channel attack is an attack based on information gained from knowledge of the **physical implementation** of a system (process), rather than theoretical weaknesses in the algorithms

- Similar to covert channels, but information is leaked **unintentionally**
  - Example, access to a **shared resource**

- Covert channel is a mechanism used to transmit info using methods not originally intended for data transmission (unauthorized and hidden) - https://www.youtube.com/watch?v=YgZJ5z7Hmhw

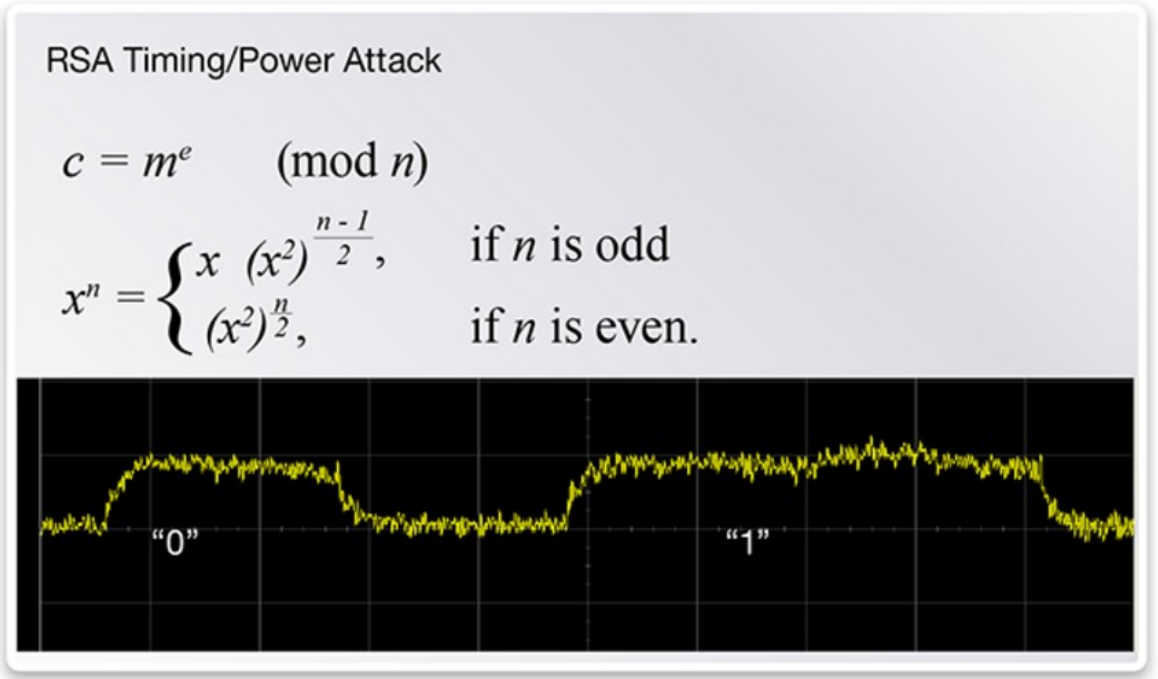
Side Channels

System and Network channels (unintentional or provoked by):

- Heat, Cold, Low Power, Microwaves …

Examples:

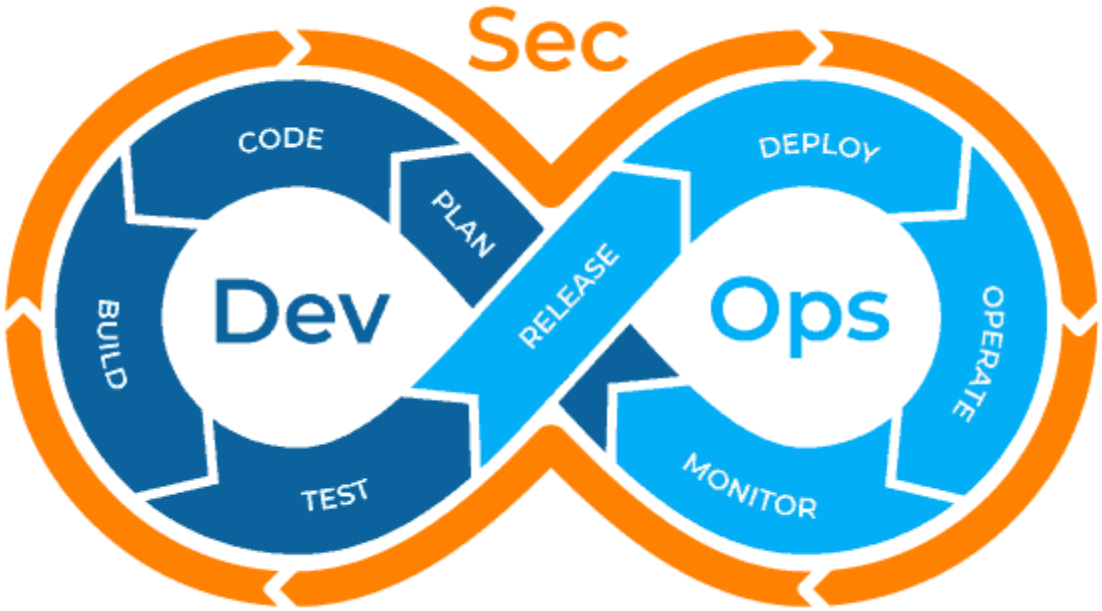
- Fault Attacks
- Timing Attacks
- Cache Attacks
- Power Analysis
- Electromagnetic Emissions
- Acoustic Emissions
- Information Disclosure



Side Channels

- Side channels allow an attacker to infer information about a secret by observing nonfunctional characteristics of a program, such as execution time or memory consumed. Recall that a program can be viewed as a communication channel where information is transmitted from a source H to a sink O. For side-channel analysis, the sink O is not necessary an output variable but rather a nonfunctional characteristic of program execution, such as running time, power consumption, number of memory accessed or packets transmitted over a network.

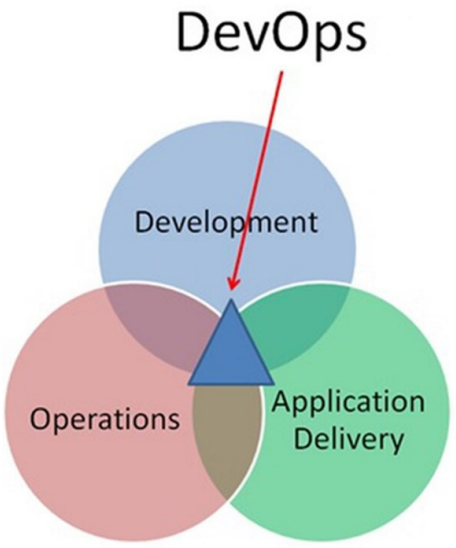
DevSecOps

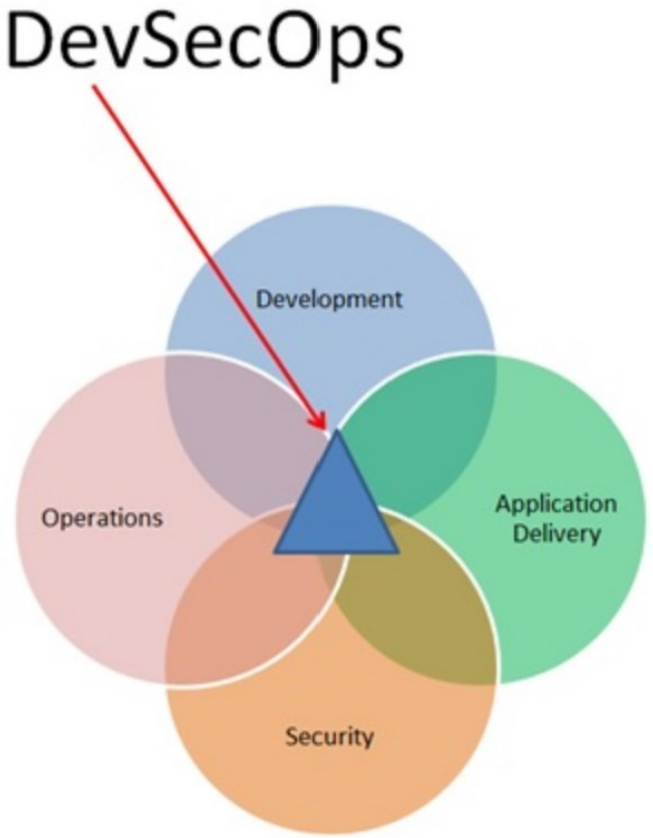


Everyone is responsible for security

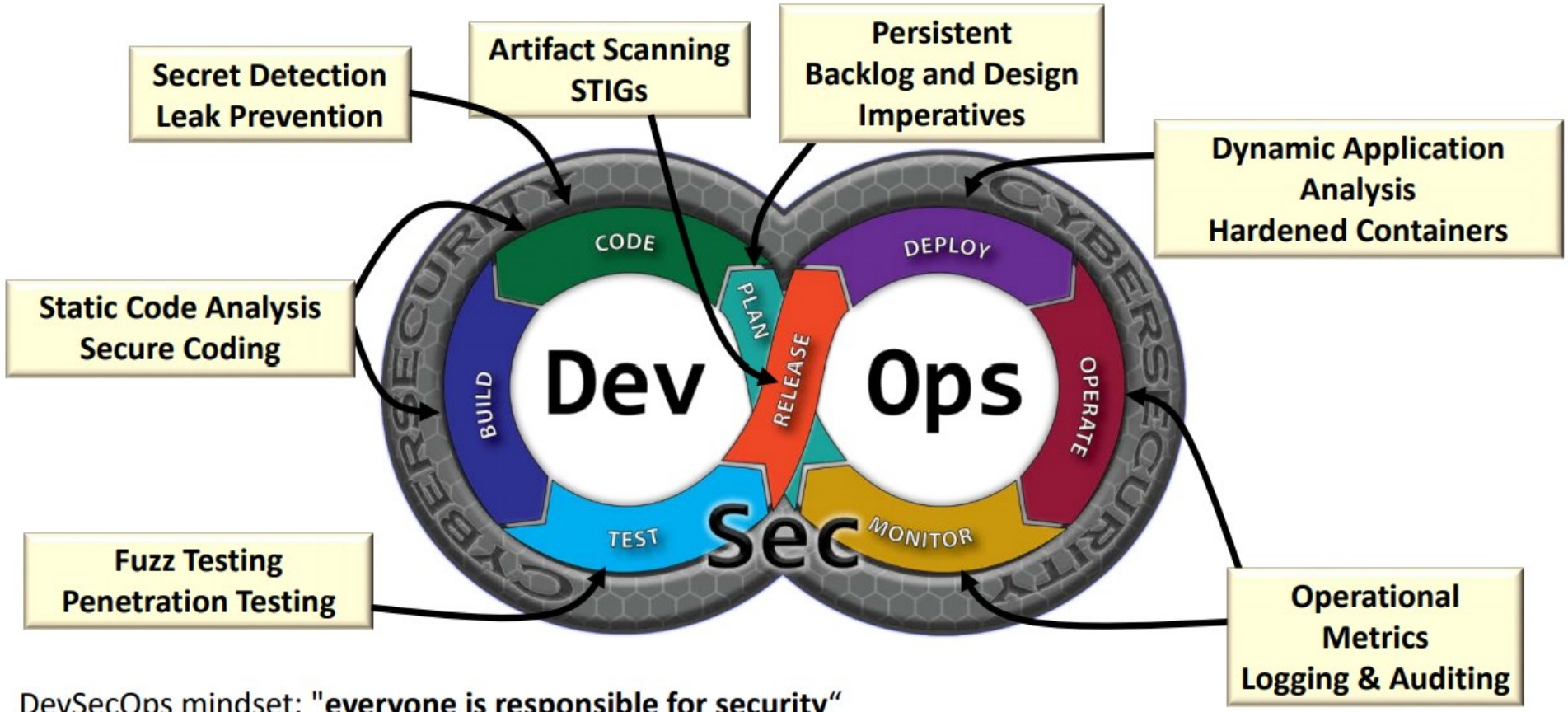
DevSecOps


DevOps
PLAN
CODE
BUILD
TEST
RELEASE
DEPLOY
OPERATE
MONITOR


DevOps
Development
Operations
Application Delivery

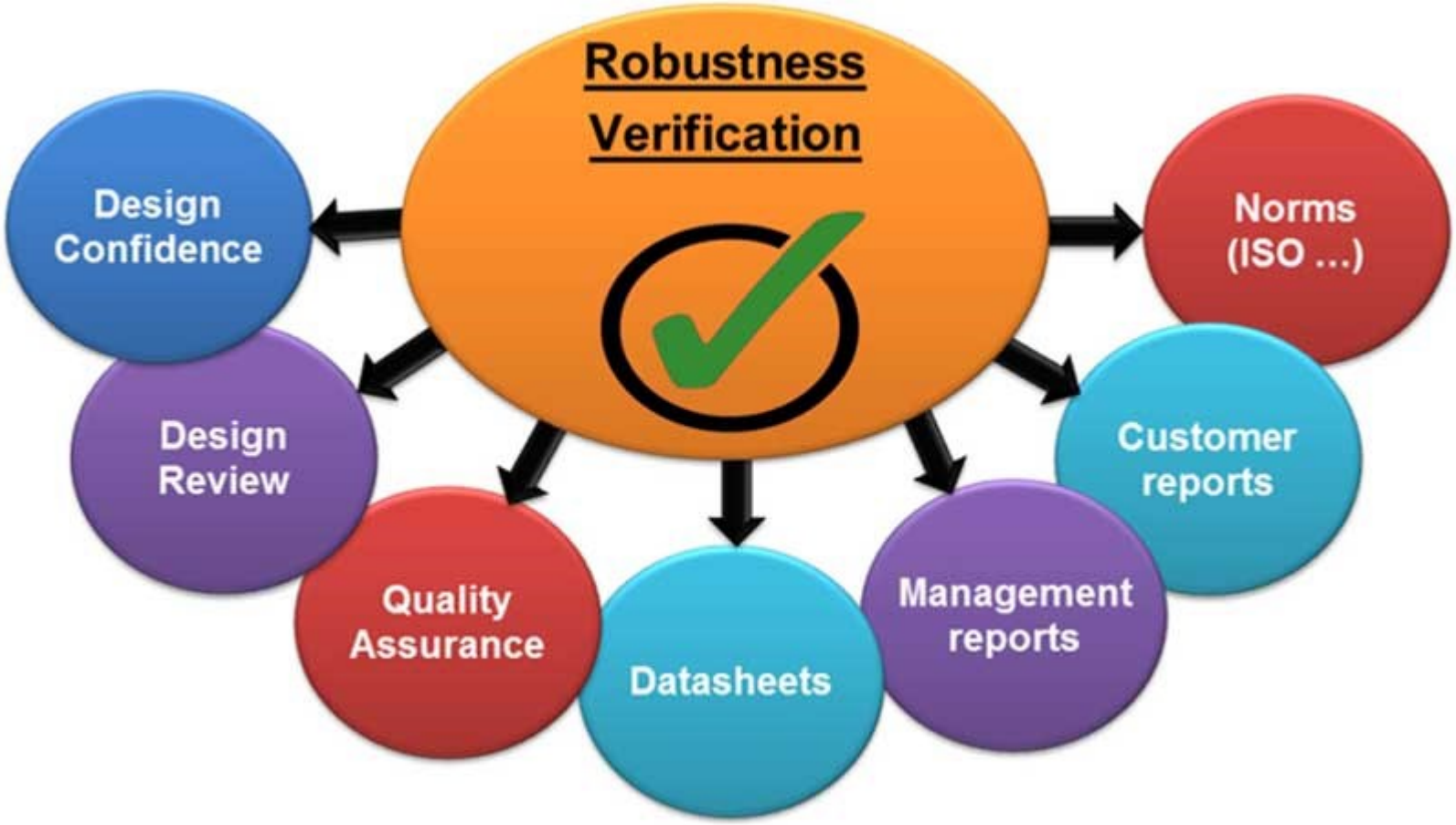

DevSecOps
Development
Operations
Application Delivery
Security

DevSecOps



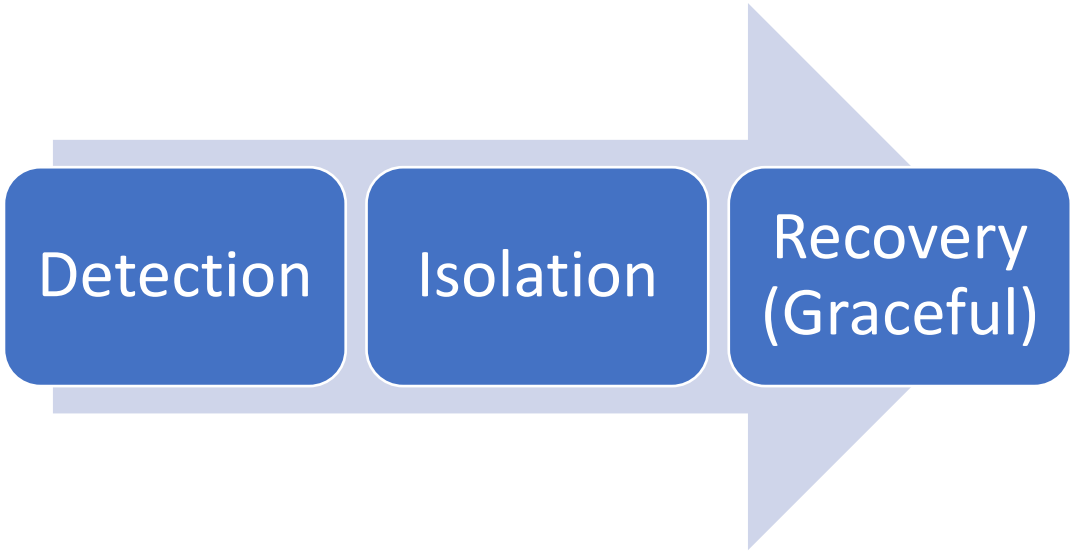
DevSecOps mindset: "**everyone is responsible for security**"

Course Summary



Design Principles

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



Secure SW Lifecycle

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

Software Quality Attributes

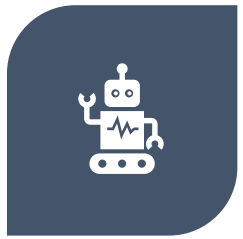
Source: ISO/IEC CD 25010 Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Quality model and guide, 2011.

Security Requirements

S – SPECIFIC

M – MEASURABLE

A – ACHIEVABLE / ATTAINABLE



R – RELEVANT

T – TIME-BOUND

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

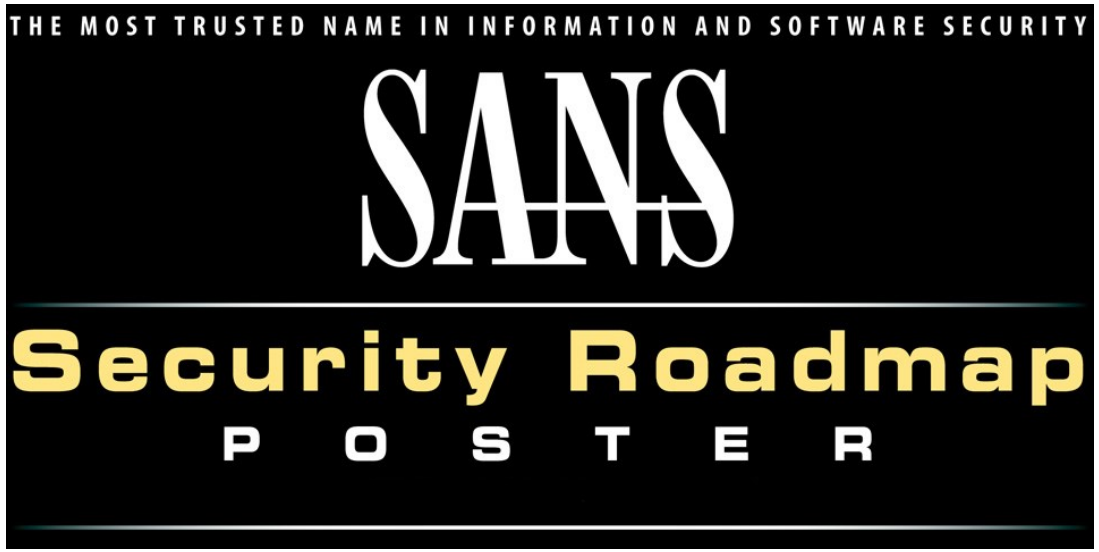
SANS SWAT Checklist

- 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



Secure Coding Practices

universidade de aveiro

Critical software

Practice	Description
1. Validate input.	Validate input from all untrusted data sources. Proper input validation can eliminate the vast majority of software vulnerabilities. Be suspicious of most external data sources, including command line arguments, network interfaces, environmental variables, and user controlled files [Seacord 05].
2. Heed compiler warnings.	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.
3. Architect and design for security policies.	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.
4. Keep it simple.	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.
5. Default deny.	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
6. Adhere to the principle of least privilege.	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].
7. Sanitize data sent to other systems.	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.
8. Practice defense in depth.	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].
9. Use effective quality assurance techniques.	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].
10. Adopt a secure coding standard.	Develop and/or apply a secure coding standard for your target development language and platform

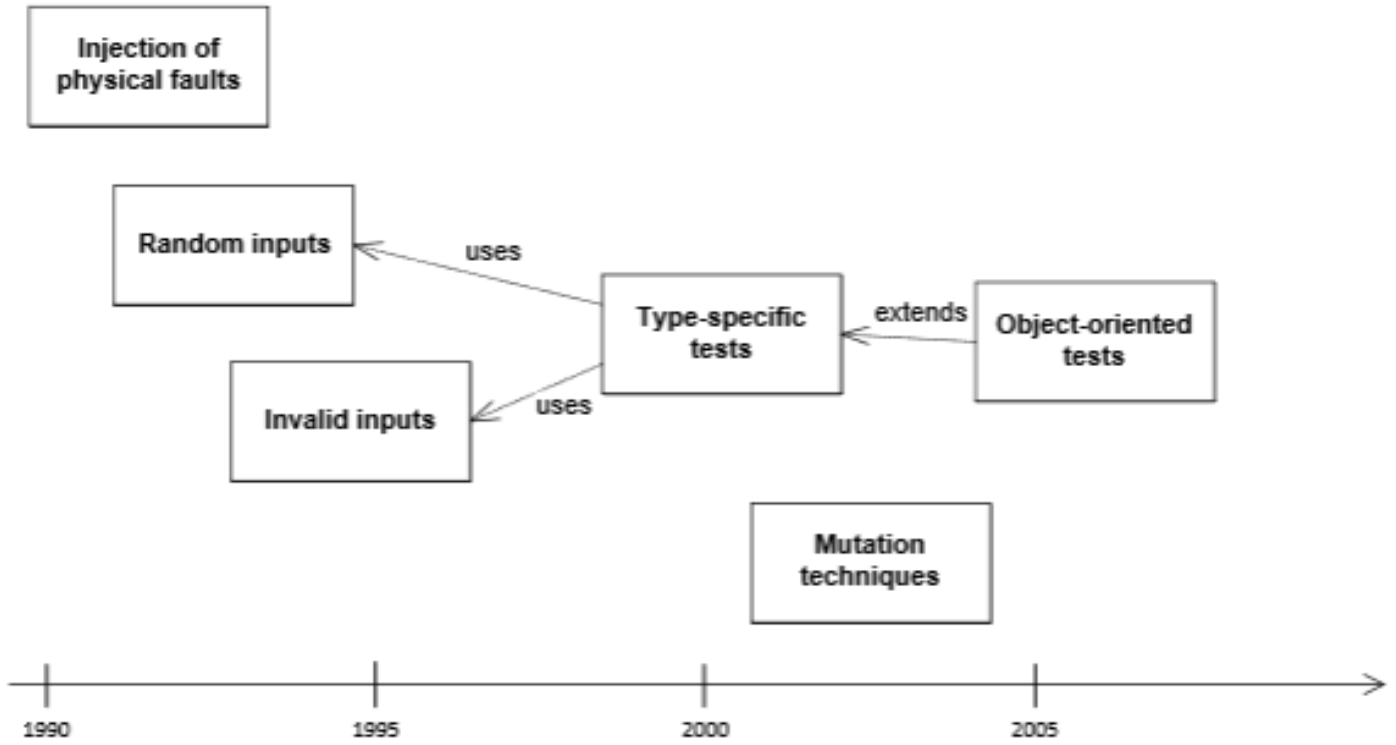
7 Pernitious Kingdoms

- 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: Tsipenyuk, Chess, and McGraw, "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors", *Proceedings SSATTM,* 2005

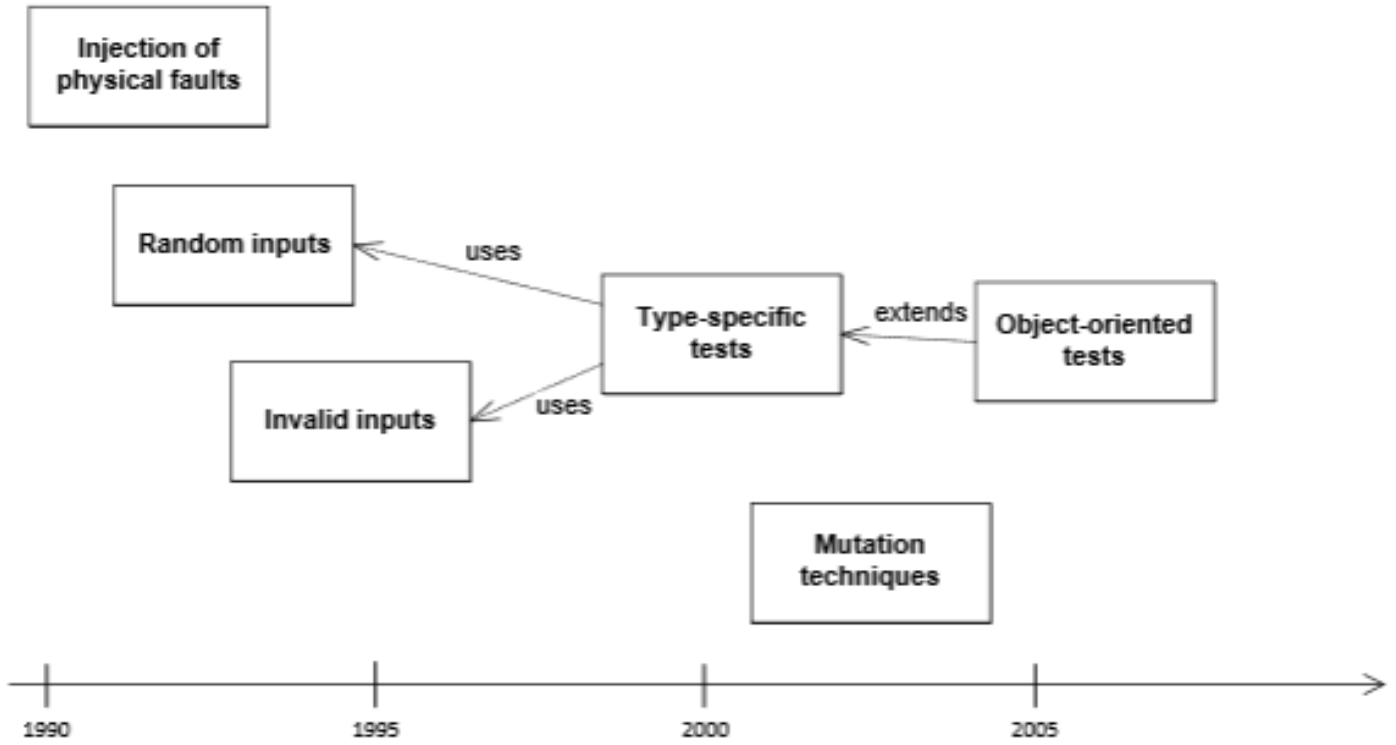
Robustness Test

- 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



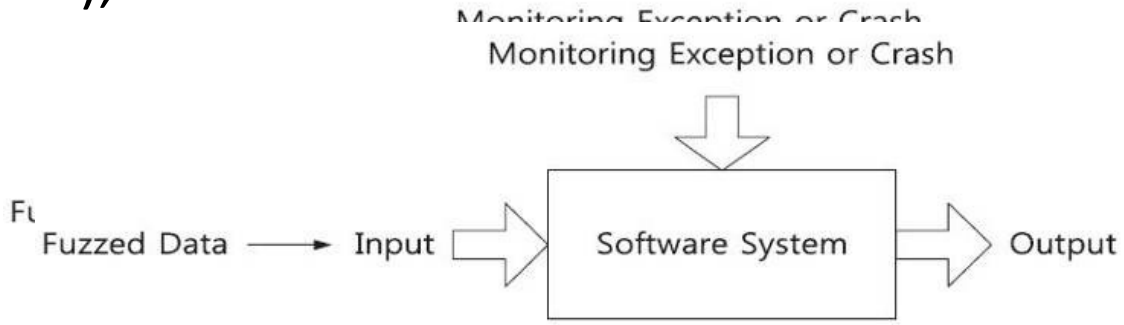
Pen Test

- 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



Fuzzy Test

- Automaticaly 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())



A safety Lifecycle Example

universidade de aveiro

Critical software

1. Concept	2. System Definition and Application Conditions	3. Risk Analysis	4. System Requirements	5. Apportionment of System Requirements
• **Develop a level of understanding of the system** sufficient to enable all subsequent RAMS tasks • Review previously achieved safety performance	• Define the mission profile of the system and its boundaries • Establish the application conditions influencing the characteristics of the system • **Define the scope of the hazard analysis** • Establish the Safety Plan	• **Perform System Hazard & Safety Risk Analysis** • Set-up Hazard Log	• **Specify the overall System Safety requirements** • Establishment of Safety Management	• Define the RAMS acceptance criteria for the designated sub-systems and components • **Apportion System Safety targets and requirements** to designated sub-systems and components

A safety Lifecycle Example

universidade de aveiro

Critical software

6. Design and Implementation	7. Manufacturing	8. Installation	9. System Validation	10. System Acceptance
• **Create** sub-systems and components • **Demonstrate that sub-systems and components conform to RAMS requirements** • Implement Safety Plan • Prepare Generic Safety Case	• Implement a process which produces RAMS-validated sub-systems and components • Use Hazard Log	• Assemble and install the total combination of sub-systems and components • Initiate system support arrangements • Establish Installation Programme	• **Validate that the total combination of sub-systems, components and external risk reduction measures comply with the RAMS requirements for the system** • Commission the total combination of sub-systems, components risk reduction measures • Prepare, and if appropriate accept, the Application Specific Safety Case	• Assess compliance of the total combination of sub-systems and components with the overall RAMS requirements of the complete system • **Accept the system for entry into service** • Assess Application Specific Safety Case

A safety Lifecycle Example

11. Operation and Maintenance	12. Performance Monitoring	13. Modification and Retrofit	14. Decommissioning an Disposal
• **Operate** maintain and support the total combination of sub-systems and components such that compliance with system RAMS requirements is maintained	• Maintain confidence in the RAMS performance of the system • **Collect, analyse, evaluate and use performance and Safety statistics**	• Control system modification and retrofit tasks to maintain system RAMS requirements • **Consider safety implications for modification and retrofit**	• Control system decommissioning and disposal tasks • Perform hazard analysis and risk assessment

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

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	10
Likely	4	5	6	7	8	9
Possible	3	4	5	6	7	8
Improbable	2	3	4	5	6	7
Highly improbable	1	2	3	4	5	6

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

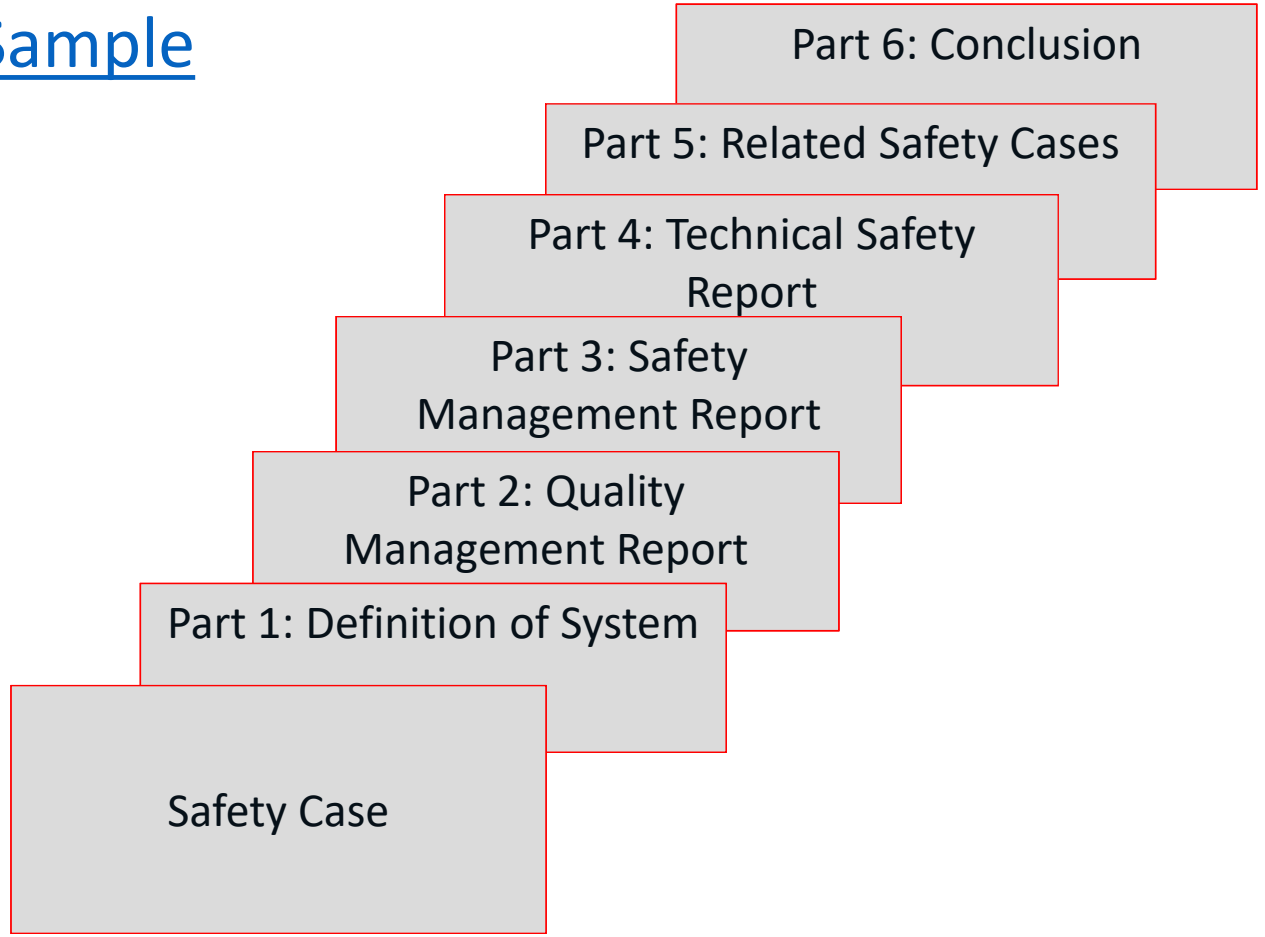
Safety & Security

- **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, Space, Railway	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 Case

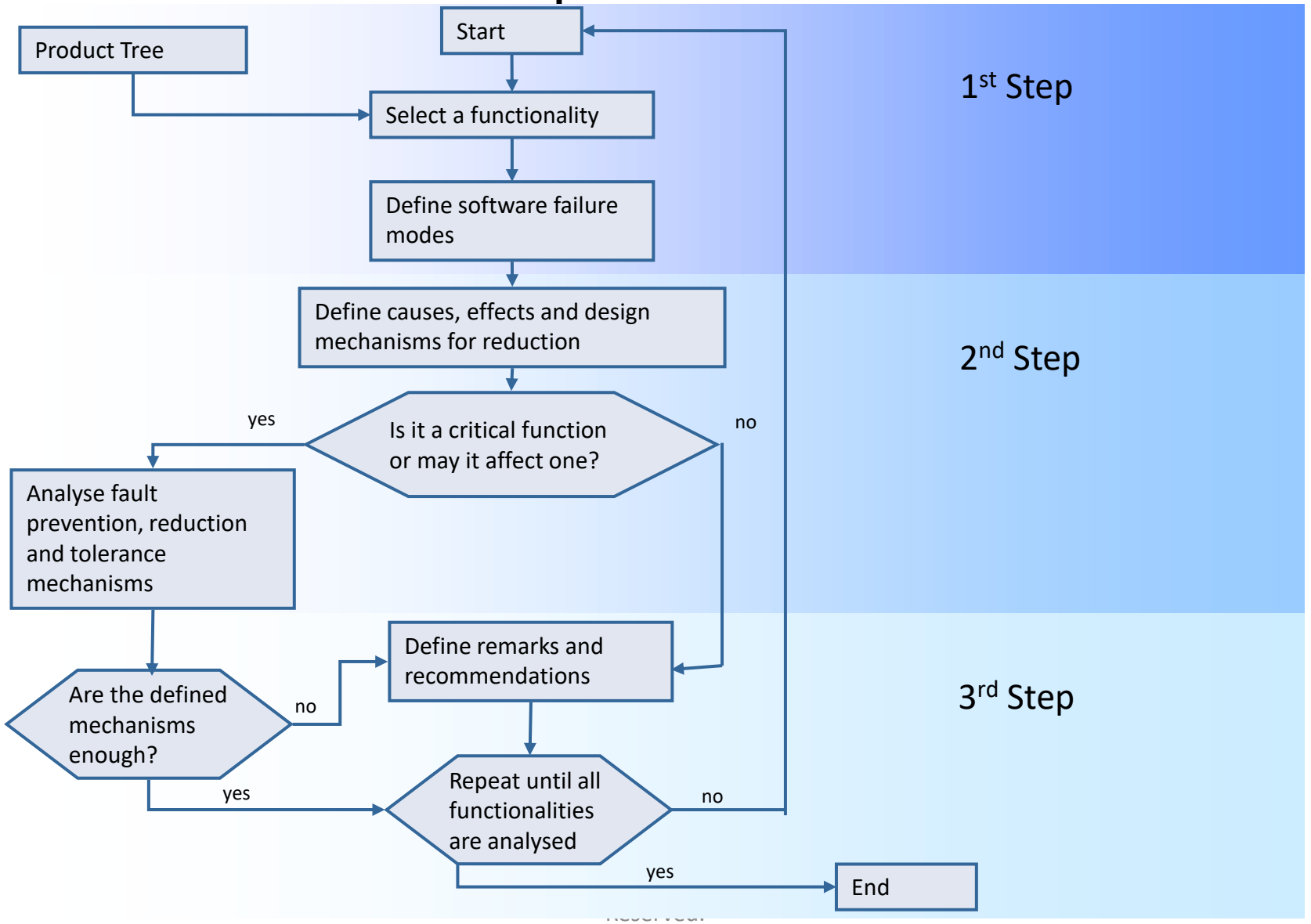
- Safety Case Sample



FMEA Example

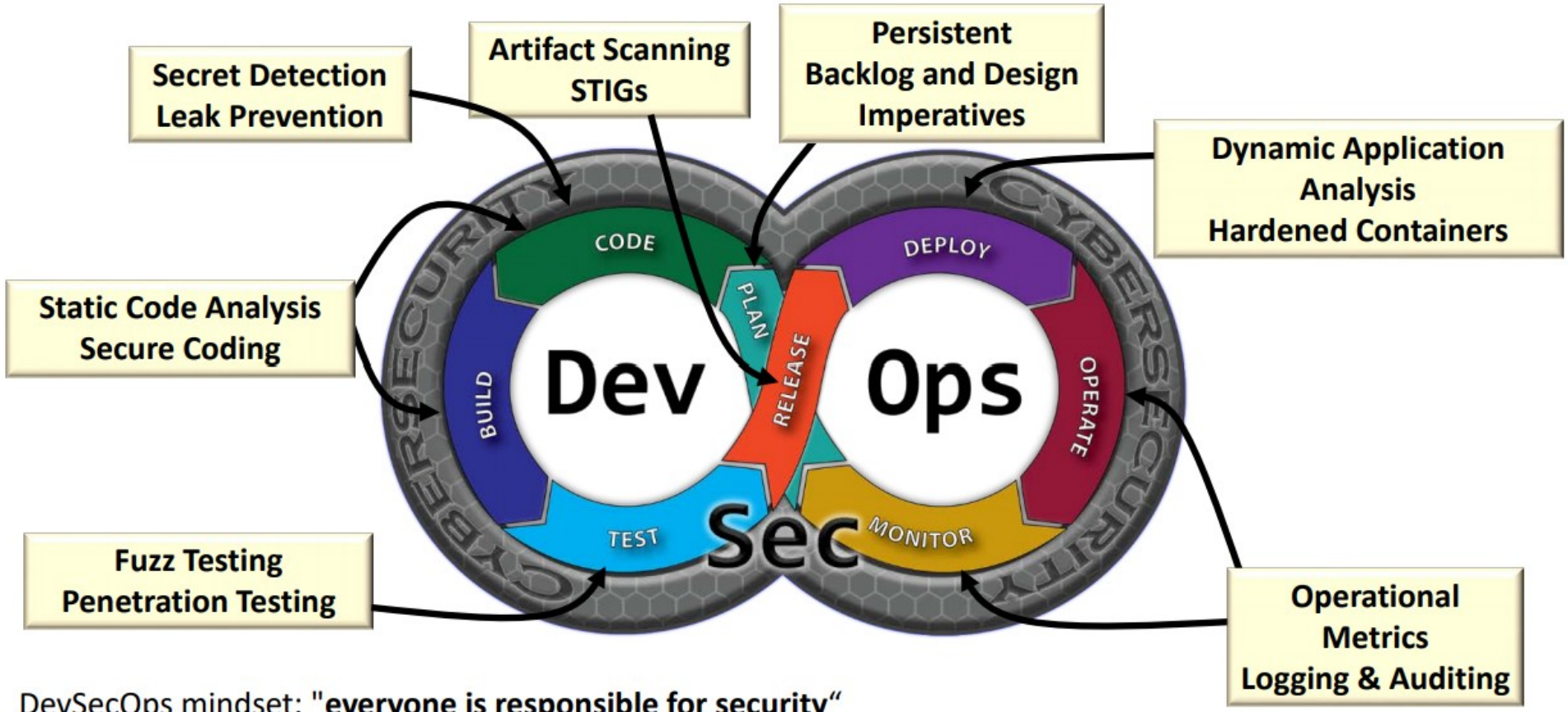
- Failure Modes and Effects Analysis (FMEA)
- Start from the foreseen functions (at system or software level) and ends up at system level with “odd” situations.
- Can be combined with a Fault Tree Analysis
- Generic (but not necessarily the only ones) failure modes:
  - No function
  - Incorrect Function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

FMEA Example



- FMEA Sheet Sample

DevSecOps



DevSecOps mindset: "**everyone is responsible for security**"

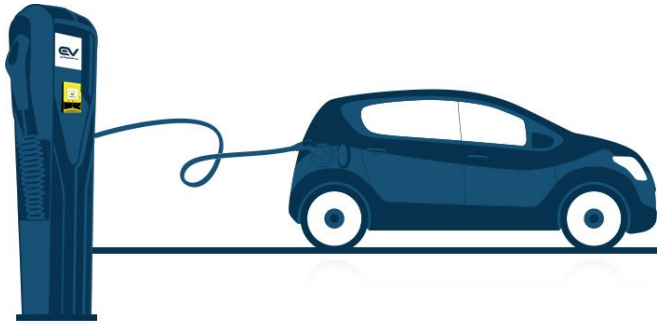
The End

universidade de aveiro

Critical software

Exam Example

- Considere uma aplicação/solução para sistemas de carregamento de viaturas eléctricas (quiosque com monitor táctil).
- Antes de iniciar o carregamento o utilizador deve autenticar-se (no ponto, por ligação wifi ou Bluetooth, através de app específica)
- Acesso através de registo prévio
- Na sua conta o utilizador consegue ver e alterar: Nome, Email, Morada, dados da viatura, dados do CC, nº de contribuinte, histórico dos consumos, histórico das facturas (mensais), cartão de crédito associado para pagameno directo, e outras configurações de conta (tipo de pagamento, Código de desconto, etc.)



Exam Example

- 1) Descrever um plano genérico cobrindo, pelo menos, a parte da especificação do sistema (não da app), o desenvolvimento, testes e aceitação/instalação. [10 pts]

- 2) À luz de uma análise de ameaças de segurança (threats analysis) aplicável a este sistema [15 pts]:
  - Identificar os principais atributos de segurança que se devem ter em conta, e explicar brevemente cada um deles com alguns exemplos (3 atributos no mínimo, 6, no máximo).

- 3) Escrever entre 5 e 10 requisitos relacionados com a segurança funcional do sistema, cobrindo tópicos relacionados com os atributos apresentados na questão 2. Fazer esse mapeamento, mesmo que alguns requisitos mapeiem com mais do que um atributo. [15 pts]

Exam Example

- 4) Definir um conjunto de testes para confirmar a correcta implementação dos requisitos escritos na questão 3. Mapear cada caso de teste ao(s) respectivo(s) requisito(s). [15 pts]

- 5) Para testar a robustez da solução, fornecer os seguintes dados [20 pts]:
  - Lista de todas as interfaces internas e externas da solução.
  - Nomear e descrever (até 100 palavras cada) dois ataques de segurança que poderiam ser descobertos ao aplicar uma metodologia de penetration testing.
  - Apresentar 3 soluções para evitar ataques de DOS (Denial-of-Service).

- 6) O cliente identificou 4 ameaças que pretende ver resolvidas com a própria solução (sem intervenções externas, nem interrupções no serviço), e estas são [20 pts]:
  - A) Instalação de vírus ou malware através do terminal
  - B) Acesso indevido a funções de configuração do terminal (deve ser permitido apenas com o utilizador “maintenance”)
  - C) Obtenção de dados confidenciais de outro utilizador
  - D) Utilização de uma conta alheia para carregamento

  No âmbito de uma análise de ameaças (hazard analysis) fornecer, para cada ameaça listada pelo cliente, a seguinte informação:
  - Possíveis consequências;
  - Possíveis causas;
  - Algumas medidas a implementar/adoptar para eliminar a ameaça.

- 7) Qual dos seguintes componentes não está relacionado com a tríade CIA? [5 pts]
  - A) Integridade
  - B) Disponibilidade
  - C) Fiabilidade
  - D) Confidencialidade
  - E) Não Repúdio