universidade de aveiro

Critical software

Secure Software Design Principles

Nuno Silva, PhD, Critical Software SA
E.: 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

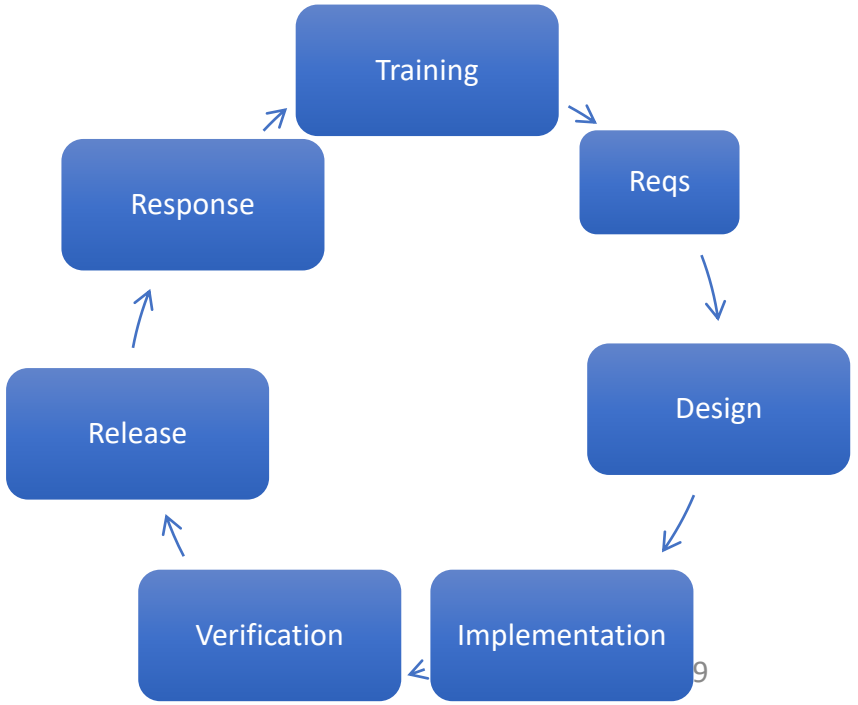
- Characteristics:
  - “Good” Requirements
  - Eliciting Nonfunctional Requirements
  - Documenting Nonfunctional Requirements
  - Verifying, Validating (eventually qualifying or certifying)
  - Identifying Restrictions, and
  - Documenting…

- 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

universidade de aveiro

Critical software

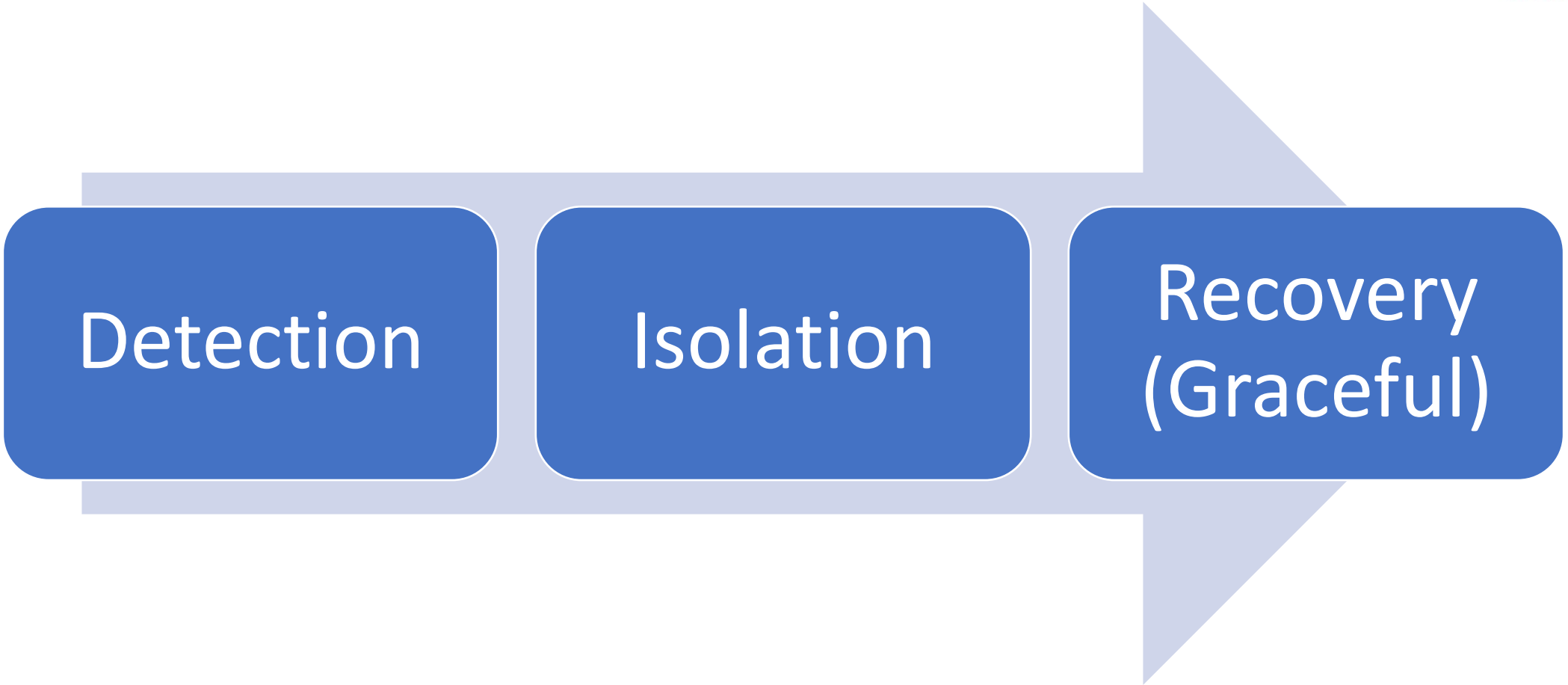
1. Apply Defense in Depth
2. Use a Positive Security Model
3. Fail Securely
4. Run with Least Privilege
5. Avoid Security by Obscurity
6. Keep Security Simple
7. Detect Intrusions
    1. Log All Security-Relevant Information
    2. Ensure That the Logs Are Monitored Regularly
    3. Respond to Intrusions
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



Architecting for the Web/Cloud

universidade de aveiro

Critical software

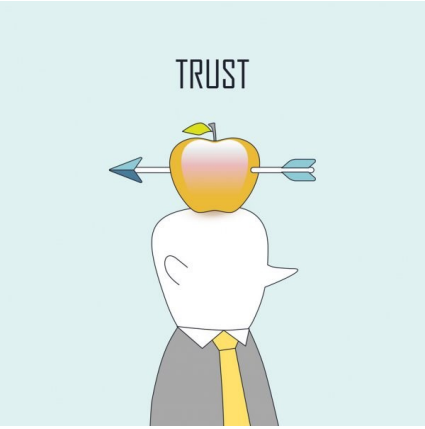
- Why Design for Failure when Nothing Fails? (everything fails…)
- **Build Security in all layers** (do not trust)
- Leverage alternative processing/storage (redundancy pays off)
- Implement elasticity (flexibility, scalability, easy restart)
- Think parallel (decoupling data from computation, load balancing, 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

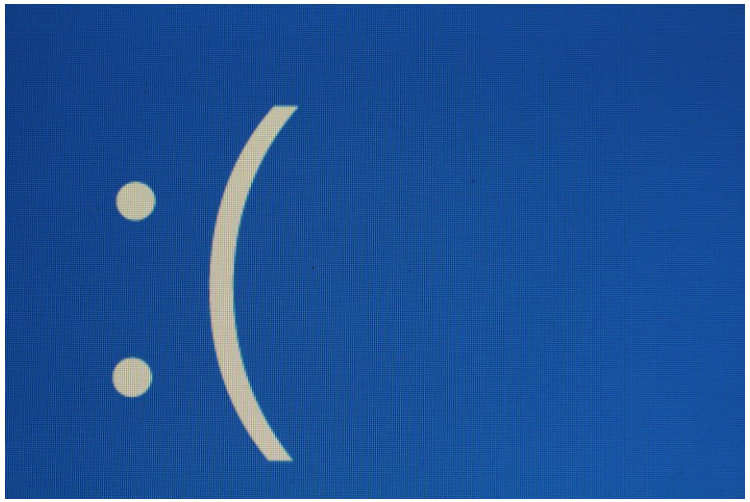
Design Best Practices – Web/Cloud

- 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.
- XSRF 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



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

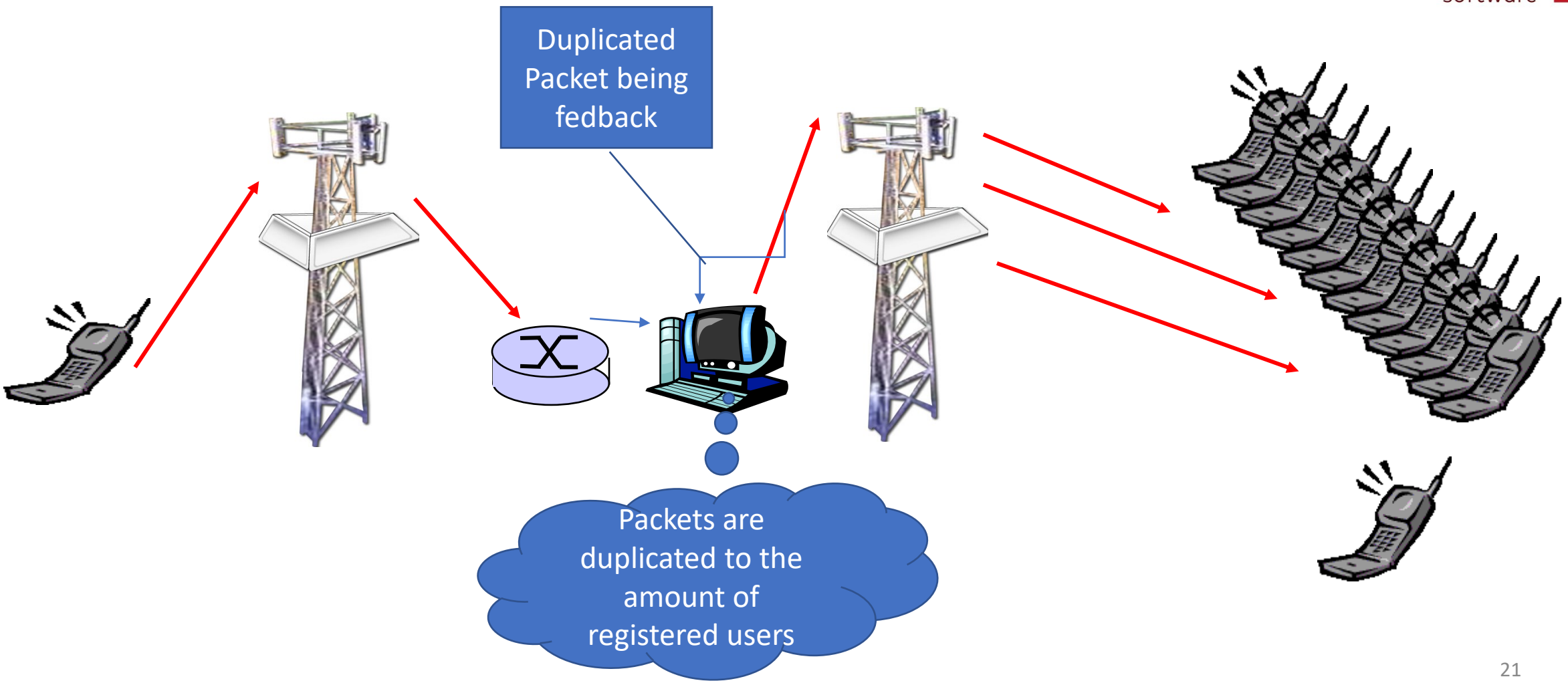
Design Best Practices - Security

- Data packets (from calls) where 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

universidade de aveiro

Critical software

- https://cheatsheetseries.owasp.org

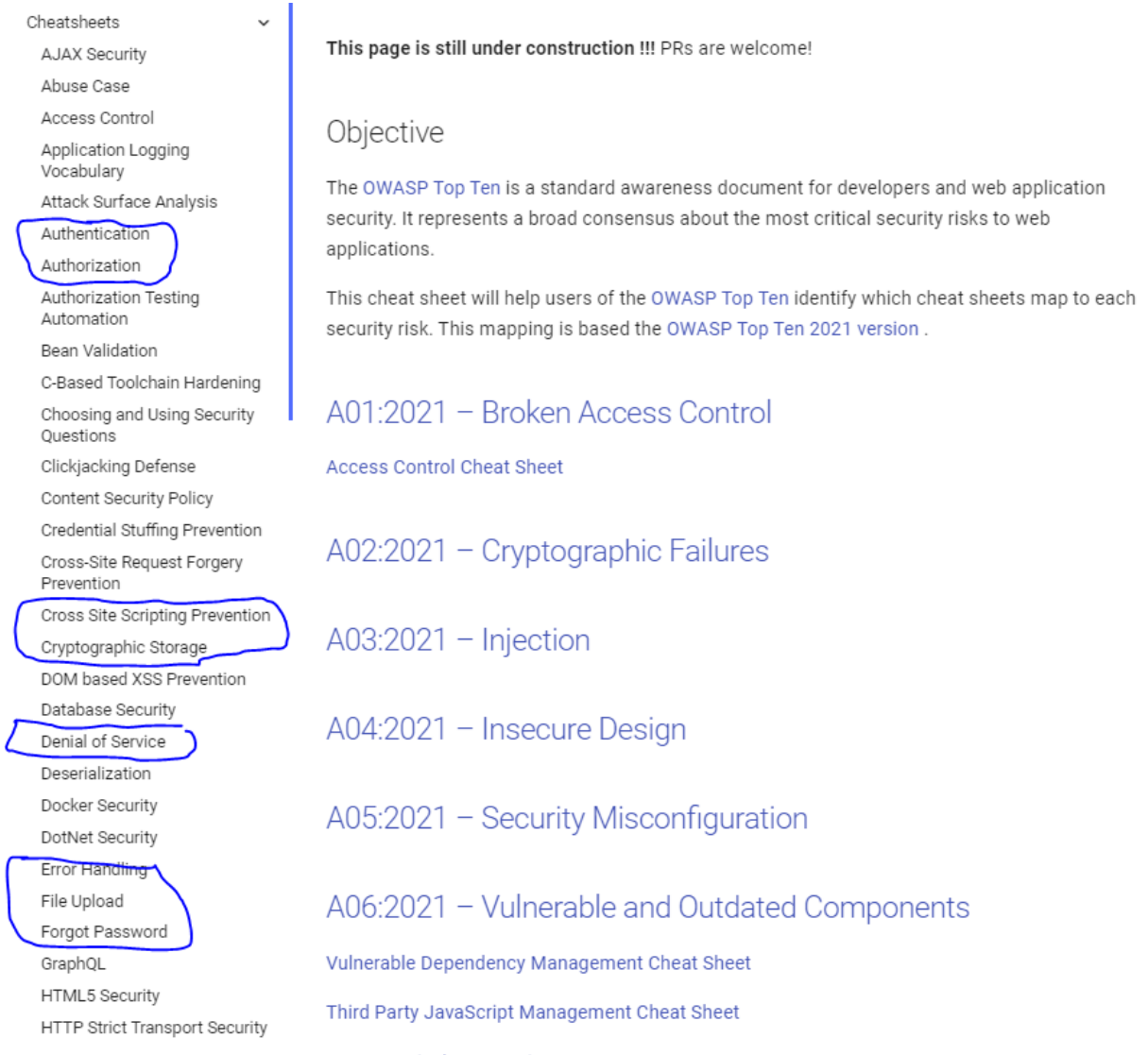
One Resource to explore

universidade de aveiro

Critical software

- 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



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

universidade de aveiro

Critical software

- Next up: Software security lifecycle