Segurança de Software

1. Introdução

A segurança de software é uma disciplina crítica no desenvolvimento de aplicações modernas, sendo essencial para proteger sistemas, dados e utilizadores contra ameaças internas e externas. A sua relevância é reforçada pelo aumento das vulnerabilidades em software, que exploram falhas como SQL Injection, Cross-Site Scripting (XSS) e Cross-Site Request Forgency (CSRF), entre outras. Estas vulnerabilidades são frequentemente o resultado de práticas de programação inadequadas ou da ausência de validações rigorosas nos sistemas.

Os princípios de segurança em desenvolvimento de software, como a validação de entradas, o controlo de acessos e a proteção de dados sensíveis, são pilares fundamentais para mitigar riscos. Estes princípios são complementados pelo uso de ferramentas de análise de segurança que permitem identificar e corrigir vulnerabilidades de forma eficiente e sistemática. Ferramentas como o OWASP ZAP e o SonarQube destacam-se pela capacidade de realizar auditorias detalhadas e melhorar a qualidade e a segurança do código.

Para além de identificar vulnerabilidades, a segurança de software exige uma abordagem proativa, onde a remediação eficaz e a implementação de boas práticas de programação segura asseguram a criação de sistemas mais robustos. Compreender e aplicar estes conceitos é indispensável para desenvolver soluções que estejam alinhadas com os padrões globais de segurança e com as necessidades crescentes de proteção em ambientes globais.

2. Objetivos e competências a desenvolver

Ao concluir esta atividade laboratorial, os estudantes deverão ser capazes de:

- Compreender os princípios fundamentais de segurança no desenvolvimento de software, reconhecendo a sua importância em todas as fases do ciclo de desenvolvimento.
- Identificar e analisar vulnerabilidades comuns, como SQL Injection, XSS e CSRF, avaliando o seu impacto em aplicações e sistemas.
- Aplicar práticas seguras de programação, implementando validações robustas e mecanismos de proteção contra injeções de código e outras ameaças.
- Utilizar ferramentas de análise de segurança para realizar auditorias de segurança e interpretar os respetivos relatórios.
- Propor e implementar ações de remediação para corrigir vulnerabilidades detetadas, assegurando a conformidade com padrões de segurança.
- Analisar a evolução das vulnerabilidades ao longo do tempo, utilizando informações do CWE Top 25 para compreender tendências e mudanças no panorama de segurança de software.

3. Instruções

A atividade será dividida em 4 etapas:

Etapa 1: Análise da evolução das vulnerabilidades comuns

1. Aceda ao website do CWE (https://cwe.mitre.org/) e consulte as listas anuais “CWE Top 25 Most Dangereous

Software Weaknesses" desde 2019 até 2024.
2. Registe as vulnerabilidades identificadas em cada ano, criando uma tabela comparativa que inclua:
   - Nome da vulnerabilidade.
   - Código CWE.
   - Breve descrição.
   - Alterações de posição na classificação ao longo dos anos.
3. Analise as tendências, focando-se nas vulnerabilidades mais recorrentes e naquelas que emergiram ou desapareceram da lista.
4. A análise não deverá exceder as 2 páginas e deverá incluir gráficos (p.ex., evolução de posições de cada vulnerabilidade), tabelas e uma análise textual das suas observações.

Etapa 2: Auditoria manual de segurança

1. Analise o código fornecido, focando-se nos seguintes aspetos:
   - Validação e sanitização de entradas do utilizador.
   - Controlo de acesso a recursos.
   - Gestão de sessões e cookies.
   - Armazenamento e exposição de dados sensíveis.
2. Identifique as vulnerabilidades presentes, justificando cada deteção com base nas práticas recomendadas de segurança e nas vulnerabilidades da lista CWE.
3. Classifique cada vulnerabilidade com base no impacto (alto, médio, baixo) e na probabilidade de exploração (alta, média, baixa).
4. Documente os resultados numa tabela que inclua:
   - Identificação da vulnerabilidade.
   - Localização no código (ficheiro e linha).
   - Classificação de impacto e probabilidade.
   - Proposta inicial de remediação.

Etapa 3: Auditoria automática com o OWASP ZAP e SonarQube

1. Instale e configure as ferramentas OWASP ZAP e SonarQube.
2. Com o OWASP ZAP (https://www.zaproxy.org/; https://github.com/The-DevSec-Blueprint/dast-scanning-with-zap):
   - Realize um scan de segurança na aplicação fornecida, analisando as interações entre cliente e servidor.
   - Exporte e analise o relatório gerado, identificando as vulnerabilidades encontradas.
3. Com o SonarQube (https://www.sonarsource.com/; https://github.com/emad-zaamout/SonarQube-Dockerized):
   - Realize uma análise estática do código-fonte fornecido.
   - Analise o relatório gerado, identificando problemas de segurança e qualidade do código.
4. Compare os resultados das auditorias manuais e automáticas, evidenciando:
   - Vulnerabilidades identificadas em comum.
   - Vulnerabilidades exclusivas de cada método.
   - Falsos positivos ou falsos negativos detetados pelas ferramentas.

Etapa 4: Ações de remediação

1. Utilize as melhores práticas de programação segura para corrigir as vulnerabilidades detetadas. Assegure-se de:
   - Implementar validação robusta das entradas do utilizador.
   - Proteger dados sensíveis (p.ex., hashing de passwords, utilização de variáveis de ambiente para chaves).
   - Corrigir problemas de controlo de acessos e gestão de sessões.
2. Documente detalhadamente cada correção implementada, incluindo:
   - O problema identificado.
   - A solução aplicada.
   - A secção de código antes e depois da alteração.
3. Reexecute as ferramentas OWASP ZAP e SonarQube para verificar se as vulnerabilidades foram efetivamente eliminadas.

Bom trabalho!