Desenvolvimento Seguro de Sistemas Web: Uma Revisão Sistemática

Gustavo R. Malacarne[1], **Edson T. Camargo**[1]

Tecnologia em Sistemas para Internet – Universidade Tecnológica Federal do Paraná (UTFPR)
Toledo – PR – Brazil

gustavomalacarne@alunos.utfpr.edu.br, edson@utfpr.edu.br

Abstract. *The objective of this work is to present a systematic review on the development of Web applications from the main security vulnerabilities described in the OWASP TOP10 list. This systematic review aims to identify research, tools and methodologies that provide safe ways to develop Web systems. Initially, 165 works were found, 96 of which were selected as relevant. Results present the works that deal with the main vulnerabilities described in the OWASP TOP 10, and the main tools and methods for designing and developing secure software.*

Resumo. *O objetivo deste trabalho é apresentar uma revisão sistemática sobre o desenvolvimento de aplicações Web a partir das principais vulnerabilidades de segurança encontradas na lista OWASP TOP10. Esta revisão sistemática visa identificar pesquisas, ferramentas e metodologias que proporcionam formas seguras de desenvolver sistemas Web. Inicialmente foram encontrados 165 trabalhos, sendo desses 96 selecionados como relevantes. Resultados apresentam os trabalhos que tratam das principais vulnerabilidades descritas na OWASP TOP 10, e as principais ferramentas e métodos para projeto e desenvolvimento de um software seguro.*

1. Introdução

Softwares para a Internet são lançados em grande volume no mercado, vão desde simples aplicativos para celular até complexos sistemas para comércio eletrônico e estão sujeitos a ameaças e vulnerabilidades. Uma ameaça é qualquer ação que possa danificar um ativo. Uma vulnerabilidade é qualquer ponto fraco em um sistema que possibilite que uma ameaça cause danos ao sistema [Kim and Solomon 2018]. Segundo o CVE Details, foram relatadas 14.714 vulnerabilidades em 2017, mais do que o dobro do ano anterior. Em 2018 esse número foi ainda maior: 16.556. Em 2019 o número de vulnerabilidades diminuiu para 12.174, porém ainda o dobro de 2016 [MITRE 2019].

Softwares Web não estão somente sujeitos à ameças externas. A própria equipe de desenvolvimento, por desconhecer boas práticas de segurança da informação, acaba por inserir brechas de segurança durante o desenvolvimento do sistema, conforme [Uto 2013]. Ainda de acordo com Uto, um software seguro é aquele que satisfaz os requisitos implícitos e explícitos de segurança em condições normais de operação e em situações decorrentes de atividade maliciosa de usuários. Comumente, o desenvolvimento de aplicações em geral, incluindo àquelas para a Web, limita-se a testar falhas após o desenvolvimento do software. Contudo, abordar as vulnerabilidades nesse momento é mais

caro do que em fases de análise e implementação, podendo atrasar facilmente a entrega do software. É então essencial que para cada fase de um ciclo de desenvolvimento de software haja um conjunto de recomendações de segurança para serem implementadas [Kissel et al. 2008].

De fato, a construção de um software seguro exige atenção constante de toda a equipe de desenvolvimento [Zenah and Aziz 2011]. No entanto, segurança de informação e engenharia de software são duas grandes áreas em si e geralmente são apresentadas tanto a estudantes de graduação quanto a profissionais de forma isolada, sem interseção. Apesar disso, há na literatura diversos trabalhos que abordam ferramentas, metodologias e processos disponíveis para evitar a inclusão de vulnerabilidades bem como incorporar e avaliar a segurança durante o desenvolvimento dos sistemas e após sua implantação.

O objetivo deste trabalho é apresentar uma revisão sistemática sobre o desenvolvimento de aplicações Web a partir das principais vulnerabilidades de segurança encontradas na lista OWASP TOP10. Esta revisão sistemática [Kitchenham 2004] considera tanto aspectos da segurança da informação presentes no processo de desenvolvimento quanto na avaliação do software já concluído e visa identificar pesquisas, ferramentas e metodologias que proporcionam formas seguras de desenvolver sistemas Web. Baseia-se nas principais vulnerabilidades expostas na mais recente lista da organização sem fins lucrativos *Open Web Application Security Project Foundation*, lançada em 2017, chamada de OWASP TOP 10. A OWASP oferece suporte aos esforços de profissionais de segurança empenhados em mitigar vulnerabilidades nos sistemas. Observar os riscos e prevenir a inserção de vulnerabilidades com base nessa lista auxilia a construir sistemas imunes às falhas mais comuns [OWASP 2017a].

Inicialmente, foram encontrados 165 trabalhos em 3 importantes bases de conhecimento. Dentre esses, 96 foram selecionados como relevantes, analisados e classificados em 3 subáreas: a) trabalhos que abordam diretamente as vulnerabilidades descritas na lista da OWASP; b) trabalhos que descrevem a implementação ou análise de um método e; c) ferramentas ou arcabouços para auxiliar o desenvolvimento de um software seguro. A primeira subárea concentrou 32% dos trabalhos, a segunda 46% e a terceira subárea trouxe 22% dos trabalhos relevantes. Das 10 vulnerabilidades listas na OWASP, a revisão encontrou duas principais: ataques de injeção e cross-site scripting (XSS).

Este trabalho segue organizado da seguinte forma. A Seção 2 apresenta definições básicas sobre o desenvolvimento de software e a segurança da informação. A Seção 3 descreve a metodologia adotada para a seleção dos artigos. A Seção 4 analisa os resultados obtidos e, por fim, a Seção 5 apresenta a conclusão.

2. Segurança da Informação e Desenvolvimento de Software

A segurança da informação é o conjunto de atividades que protegem o sistema de informação e os seus dados [Kim and Solomon 2018]. Para alcançar seus objetivos, a segurança da informação estabelece três princípios que, quando satisfeitos, garantem os requisitos para manter as informações seguras: confidencialidade, integridade e disponibilidade. O primeiro garante que as informações são acessadas apenas por pessoas autorizadas. A integridade requer que apenas usuários autorizados possam alterar as informações. Já a disponibilidade assume que as informações são acessíveis apenas por usuários autorizados, sempre que solicitarem. Adicionalmente, outras propriedades, como autenticidade,

responsabilidade, não repúdio e confiabilidade, podem também estar envolvidas.

O desenvolvimento seguro de software é um processo que padroniza as práticas recomendadas de segurança em produtos e aplicações [Microsoft 2012]. Um ciclo de desenvolvimento seguro de software visa diminuir ou evitar que vulnerabilidades sejam inseridas durante a construção do sistema ao adicionar atividades relacionadas a segurança da informação a cada fase de um ciclo de desenvolvimento de software [Uto 2013]. O processo de aplicar as recomendações de segurança em cada fase do ciclo é descrito como Ciclo de Vida de Desenvolvimento Seguro (CVDS) [Kim and Solomon 2018]. Diversas organizações e empresas desenvolvem seus próprios CVDS, como o Microsoft SDL [Microsoft 2012], BSIMM SSDL [Sammy Migues, John Steven and Mike Ware 2019] e OWASP SDLC [OWASP 2017b]. O objetivo do ciclo de vida seguro da OWASP é ajudar os usuários a reduzir os problemas de segurança e elevar o nível de segurança de todas as etapas usando a sua metodologia. Além disso, a OWASP descreve a OWASP TOP 10.

A "OWASP TOP 10" apresenta uma lista com as 10 principais ameaças a que aplicações Web estão expostas. Em comparação com a última lista divulgada em 2013, verifica-se que continua a dificuldade dos desenvolvedores e organizações em lidar com as vulnerabilidades mais comuns. A lista está ordenada respectivamente pela vulnerabilidade de Ataques de Injeção, seguida por Quebra de Autenticação, Exposição de Dados Sensíveis, Ataques de XXE (*XML External Entities*), Quebra do Controle de Acesso, Configurações Inseguras, *Cross-Site Scripting* (XSS), Desserialização Insegura, Uso de Componentes com Vulnerabilidades Conhecidas e Monitoramento e Controle de Registro Insuficiente.

3. Materiais e Métodos

As seguintes questões de pesquisa norteiam a execução desta revisão sistemática: a) quais são os principais métodos de desenvolvimento seguro de software? b) quais são as principais ferramentas que auxiliam no desenvolvimento de um software seguro? c) quais as principais dificuldades em implementar métodos de segurança no projeto de software? d) qual a vulnerabilidade da lista OWASP TOP 10 mais abordada na literatura especializada?

As principais bases selecionadas para executar a busca são as seguintes: ACM Digital Library, IEEE Xplore e Springer Link. O primeiro passo tomado foi estabelecer a seguinte *string* de busca, estruturada de forma que pudesse abranger a maior parte significativa de trabalhos conforme a base de pesquisa:

- (("security" OR "secure software") AND ("framework" OR "tool") AND ("OWASP"))
- (("security" OR "secure software") AND ("development" OR "pattern") AND ("project" OR "application") AND ("OWASP"))
- (("security" OR "secure software") AND ("OWASP"))

Para seleção dos estudos foram propostos dois tipos de critérios, os de inclusão e os de exclusão. O primeiro foi usado para incluir os trabalhos em que o estudo principal discute sobre uma ou mais ferramentas para desenvolvimento seguro e que propõe um modelo/processo/framework/metodologia de desenvolvimento seguro. Já o segundo exclui trabalhos nos quais o estudo principal tem pouca ou nenhuma relação com segurança

da informação, não fornece informações suficientes para o desenvolvimento do software ou para o uso da ferramenta. Foram selecionados artigos de 2008 a 2019.

Tabela 1. Obras restantes em cada etapa.

Biblioteca	Quantidade	Obras Relevantes
ACM Digital Library	26	10
IEEE Xplore	66	36
Springer Link	186	50
Total	278	96

4. Análise dos Resultados

A pesquisa encontrou 278 obras, sendo 165 delas não duplicadas. O passo seguinte foi avaliar as obras através da leitura dos seus resumos (*abstract*), dos tópicos abordados no trabalho e da significância das informações. Como resultado, foram selecionadas as obras relevantes à revisão sistemática. Foram identificas 96 obras como relevantes após aplicação dos critérios de inclusão e exclusão (ver Tabela 1). Das obras relevantes, observou-se um aumento crescente no número de artigos relacionados a *string* de busca. Por exemplo, houve 3 obras selecionadas em 2008, 10 em 2015 e 21 em 2018. Isso evidencia que a segurança da informação recebe maior atenção a cada ano. Outra observação a partir dos resultados é que a inclusão de termos relacionado a lista TOP 10 OWASP refinou a quantidade de obras encontradas.

As 96 obras foram classificadas em 3 grupos: a) vulnerabilidades descritas na lista da OWASP que encontrou 31 artigos; b) implementação ou análise de um método de desenvolvimento seguro com 44 artigos e; c) ferramenta ou arcabouço para auxiliar o desenvolvimento seguro com 21 obras. As subseções seguintes detalham cada uma dos grupos respectivamente.

4.1. Vulnerabilidade da lista OWASP TOP 10

A partir das vulnerabilidades listadas na OWASP TOP 10, a revisão sistemática encontrou três vulnerabilidades: falhas de injeção com 13 artigos, *cross-site scripting* (XSS) com 17 artigos e 1 artigo sobre quebra de controle de acesso. A vulnerabilidade de *cross-site request forgery* (CSRF) e o ataque de negação de serviço contaram com 3 artigos cada, mas pertencem a versão anterior da lista OWASP.

Conforme a Tabela 2, a maioria dos trabalhos encontrados destacam a falha de injeção e XSS. Prevenir o ataque de injeção requer manter os dados separados dos comandos e consultas, como afirma [OWASP 2017a]. Uma das alternativas preferenciais, conforme [Nagpal et al. 2017], é usar uma API segura para evitar o uso do interpretador ou fornecer uma interface parametrizada. É possível ainda utilizar de validações de entrada ou ”whitelist”do lado do servidor.

Já para a prevenção de XSS, é necessário a separação de dados não confiáveis do conteúdo do navegador ativo. Para tanto, é recomendado utilizar de *frameworks* atualizados que já possuem por padrão contramedidas ao XSS, como o Ruby on Rails e o React JS. Evitar também dados de solicitação HTTP não confiáveis com base no contexto da saída HTML (corpo, atributo, JavaScript, CSS ou URL) [Gupta and Gupta 2017]. Para

Tabela 2. Principais obras da Seção 4.1 e suas contribuições

Autor	Principais contribuições
[Johari and Sharma 2012]	Apresenta técnicas atuais de defesa contra injeção SQL e explorações de XSS. Além disso, propõe o desenvolvimento de contramedidas contra ataques de injeção SQL.
[Sharma et al. 2012]	Propõe um modelo para impedir a injeção SQL e estabelecer contramedidas aos ataques de *script* XSS em sites e aplicativos da Web construídos em PHP, além de prevenir falhas XSS.
[Nagpal et al. 2017]	Propõe uma modificação e extensão do método descrito por [Sharma et al. 2012]. Na obra proposta, o modelo existente é modificado incorporando um algoritmo para impedir diversas formas de ataques de injeção SQL e XSS. Além disso, novos módulos são integrados para evitar ataques de CSRF.
[Dong et al. 2018]	Apresenta um novo sistema adaptativo para detectar consultas maliciosas em solicitações da Web, chamado AMOD. O AMOD utiliza uma estratégia eficiente de aprendizado adaptativo, o SVM HYBRID, que é um híbrido de seleção de suspeitas e seleção exemplar.
[Kaur and Kaur 2014]	Argumenta sobre a incorporação da segurança durante todas as fases do ciclo de vida de desenvolvimento de software, afirmando que pode-se economizar tempo, despesas e fornecer uma defesa bem mais profunda em comparação à remoção de vulnerabilidades após o desenvolvimento do software.
[Gupta and Gupta 2017, Gupta and Gupta 2018]	Lista uma série de questões essenciais para uma interpretação aprimorada das vulnerabilidades XSS e sua influência nas infraestruturas modernas das plataformas web.
[Zlomislić et al. 2017]	Aborda a dificuldade de desenvolver uma defesa efetiva contra casos de ataques externos, como negação de serviço (*Denial of Service* ou DoS). Esses são citados como vulnerabilidades mais comuns nos sistemas e ameaçam constantemente aplicações Web devido aos casos diferenciados.
[Gauthier and Merlo 2012]	Apresenta a ferramenta ACMA (*Access Control Models Analyzer*) para detectar vulnerabilidades de controle de acesso em aplicativos da Web construídos em PHP.

proteções do lado do cliente, como afirma [Gupta and Gupta 2018], as soluções existentes são geralmente integradas como uma extensão em navegadores da web. Dessa forma é importante aplicar codificação contextual ao modificar o documento do navegador, evitando assim ataques XSS.

4.2. Implementação ou análise de métodos de desenvolvimento seguro

A revisão sistemática encontrou 44 artigos que propõem métodos, técnicas, desenvolvem e/ou implementam ferramentas para auxiliar na detecção de vulnerabilidades. Os artigos informam principalmente sobre métodos e processos usados, como a modelagem de ameaças, testes de segurança e revisão de código para garantir a inclusão da segurança no ciclo de desenvolvimento de software, conforme a Tabela 3. A detecção e mitigação de vulnerabilidades são tratadas constantemente, de forma que o principal objetivo seja propor soluções de segurança [Martínez et al. 2018, Skouby et al. 2017, Rafique et al. 2015, Johns 2008].

A modelagem de ameaças é o processo de enumerar e classificar os agentes maliciosos, seus ataques e os possíveis impactos desses ataques nos ativos de um sistema [Masood and Java 2015]. Dessa forma, o processo ajuda a detectar falhas desde o

início, oferecendo uma oportunidade para elaborar os requisitos de segurança do sistema e, consequentemente, revisar as opções de projeto ou refinar o modelo de arquitetura.

Tabela 3. Principais obras da Seção 4.2 e suas contribuições

Autor	Principais contribuições
[Masood and Java 2015]	Debate sobre a importância de adotar a modelagem de ameaças e como ela pode preparar melhor os desenvolvedores para enfrentar e mitigar ataques e tornar as arquiteturas orientadas a serviços mais seguras. Os autores ainda buscam identificar e explicar como as APIs públicas, especialmente as RESTFul, são expostas a negação de serviço e outras ameaças relacionadas a aplicativos da Web.
[Scandariato et al. 2013, Deng et al. 2011]	Discutem e analisam uma técnica popular para modelagem de ameaças, o STRIDE da Microsoft.
[Salva and Regainia 2019]	Descreve uma abordagem para orientar os desenvolvedores na geração de árvores detalhadas de defesa contra ataques que expressam as possibilidades do invasor de comprometer um aplicativo. Também fornecem as defesas que podem ser implementadas para impedir ataques com padrões de segurança.
[de Jiménez 2016, Bertoglio and Zorzo 2017]	Apresentam um novo sistema adaptativo para detectar consultas maliciosas em solicitações da Web, chamado AMOD. O AMOD utiliza uma estratégia eficiente de aprendizado adaptativo, o SVM HYBRID, que é um híbrido de seleção de suspeitas e seleção exemplar.
[Felderer and Katt 2015]	Apresenta um processo para evolução da segurança no ciclo de vida de desenvolvimento de software. De acordo com os autores é necessário que a segurança seja abordada desde o inicio do ciclo, apresentando atividades a serem implementadas em cada fase.
[Sahu and Tomar 2017, Erdogan et al. 2014, Aziz et al. 2016]	Apresentam princípios de codificação segura. Um sistema de análise de código estático interativo é desenvolvido, o que ajuda os desenvolvedores de sistemas Web a escrever códigos seguros, também facilitando programadores a mitigar vulnerabilidades mais comuns.
[Zenah and Aziz 2011, Hayrapetian and Raje 2018, Singh and Dua 2018]	Discorrem sobre formas de remediar dificuldades de implementar métodos de segurança, criando uma cultura que destaca a importância da segurança na mente de toda a equipe, durante algumas das principais fases de um desenvolvimento de software, como levantamento de requisitos, projeto, codificação, testes e implantação.

Um método bastante relevante para desenvolver aplicativos da web confiáveis, é a utilização de padrões de codificação seguros. O trabalho [Sahu and Tomar 2017] destaca-se pois traz uma abordagem abrangente baseada em gráficos para a aplicação de padrões de codificação segura textuais para o desenvolvimento de aplicações web mais resistentes. Além disso, manter para cada fase do ciclo de vida de software as recomendações de segurança adequadas [Felderer and Katt 2015].

4.3. Ferramentas e Frameworks

As principais ferramentas encontradas na revisão sistemática, listadas na Tabela 4, trazem funcionalidades voltadas ao teste de segurança, como o teste de invasão e análise estática [Cai et al. 2016, Nunes et al. 2018, Trunde and Weippl 2015]. Ao todo foram selecionados 21 artigos para esta subseção. O principal objetivo do teste de invasão é encontrar falhas que passam despercebidas, realizar testes que simulam os métodos maliciosos utilizados em busca de alguma vulnerabilidade exposta, sem comprometer a aplicação alvo. Em [Nagpure and Kurkure 2017] os autores trazem a comparação das vantagens e desvantagens de alguns métodos de testes. Ainda nessa linha, [Vibhandik and Bose 2015]

fornece uma nova abordagem de teste para detectar vulnerabilidades em aplicativos da Web ao selecionar um conjunto de ferramentas de maneira otimizada e organizada.

Tabela 4. Principais obras da Seção 4.3 e suas contribuições

Autor	Principais contribuições
[Shah and Mehtre 2015]	Descreve uma metodologia para analisar as vulnerabilidades em quatro fases de avaliação e listam ferramentas comerciais e de código aberto a serem usadas nos testes. Um estudo de caso de um teste de VAPT (*Vulnerability Assessment and Penetration Testing*) é conduzido.
[de Jiménez 2016]	Apresenta detalhes sobre todas as etapas de realização de um teste de invasão.
[Radwan and Prole 2015]	Destaca como a técnica de análise de código melhora a cobertura de testes de invasão e ainda oferece suporte a testes manuais e automatizados, resultando em um diagnóstico de análise.
[Dashevskyi et al. 2019]	Apresenta uma ferramenta que traz a proposta de combinar aplicativos, ambientes de execução e explorações automatizadas para que os pesquisadores possam realizar seus testes e experimentos e obter relatórios em vários níveis de detalhes.
[Visoottiviseth et al. 2017]	Apresenta a ferramenta de teste PENTOS, elaborado seguindo as recomendações da lista das 10 vulnerabilidades mais comuns da OWASP.
[Salva and Zafimiharisoa 2015]	Apresenta a ferramenta APSET (*Android aPplication SEcurity Testing*) para a geração e execução automáticas de casos de teste em smartphones ou emuladores.
[AlBreiki and Mahmoud 2014]	Expõe uma avaliação da eficácia das ferramentas de análise estática de segurança, pois argumentando sobre os pontos fortes e fracos de softwares não comerciais disponíveis para práticas de revisão da segurança de um código e avalia a eficácia destes para descobrir fraquezas.
[Deng et al. 2011]	Apresenta formas de modelar ameaças à privacidade em sistemas de software, de obter os requisitos de privacidade necessários e de estabelecer contramedidas que aumentem a privacidade, fornecendo um extenso catálogo de padrões de ameaças específicas que possibilitam tal proposta.

Em termos de testes de invasão, destacam-se as ferramentas Burp suite e OWASP Zed Attack Proxy (ou ZAP) [Nagpure and Kurkure 2017]. O primeiro é um agrupamento de várias ferramentas que se juntam para trabalhar em um modo passivo ou ativo, apoiando o testador em todo o processo de teste, desde a fase de planejamento até a identificação de vulnerabilidades e exploração dessas vulnerabilidades. O ZAP é uma ferramenta de *pentesting* e proxy. Seu objetivo é permitir que os desenvolvedores testem a estabilidade e a segurança de seu site ou aplicativo.

Para testes de segurança, o TestREx fornece aos testadores e desenvolvedores um ambiente para experimentos automatizados em grande escala [Dashevskyi et al. 2019]. Com a ferramenta é possível cobrir diferentes fases do SDL, atendendo às necessidades de diferentes partes. Fornece meios para a avaliação de exploits, considerando que os exploits são reproduzidos em vários contextos diferentes, facilita a compreensão dos efeitos dos exploits em cada contexto, bem como a descoberta de novas vulnerabilidades potenciais.

5. Conclusão

Este trabalho realizou uma revisão sistemática sobre a segurança da informação nos sistemas Web para apoiar projetistas e desenvolvedores a evitar as ameaças mais frequentes. Foram identificadas pesquisas, ferramentas e metodologias que proporcionam formas seguras de desenvolver sistemas Web a partir das principais vulnerabilidades expostas na mais recente lista da OWASP TOP 10.

Dos 165 trabalhos encontrados com as *strings* de busca, foi possível selecionar 96 obras relevantes para as questões de pesquisa que foram elaboradas. Dentre esse grupo, 31 (32%) obras abordaram uma vulnerabilidade da lista OWASP Top 10 como tema principal, 44 (46%) trouxeram uma análise ou implementação de um método de desenvolvimento seguro e 21 (22%) apresentaram ferramentas ou *frameworks* para auxiliar no desenvolvimento seguro. Em relação às vulnerabilidades mais comuns expostas na lista Top 10 de 2017, as duas com maior destaque foram injeção de código com 13 obras e *cross-site scripting* com 17 obras relacionadas. Para os métodos de desenvolvimento seguro, os temas principais foram modelagem de ameaças, codificação segura e testes de segurança. Já as ferramentas e *frameworks* listados na revisão sistemática, houve uma preocupação dos artigos em tratar sobre revisão estática de código, testes de invasão e segurança e detecção e mitigação de falhas.

A integração de práticas de segurança no ciclo de vida de desenvolvimento de software ainda é um desafio. No entanto, se aliada a verificação da segurança das funcionalidades do aplicativo, é possível reduzir os riscos de fontes internas e externas. Portanto a prevenção ajuda a diminuir custos das empresas porque elas não precisam responder a incidentes ou lidar com danos à sua reputação. Incorporar um modelo que adéque requisitos de segurança ao software, como a integração de ferramentas e serviços de testes automatizados ajudam os desenvolvedores a encontrar e corrigir problemas de segurança, prevenindo problemas futuros.

Para trabalhos futuros, um estudo de caso que possui algum *exploit*, programa malicioso que explora vulnerabilidades de sistemas, será significativo para aplicar e examinar funcionalidades das ferramentas *open source* encontradas com esta pesquisa. Por consequência, desenvolver uma aplicação que adote padrões de segurança descritos na busca e submetê-los aos testes, acrescentará grande relevância as obras citadas, garantindo a qualidade dos resultados apresentados.

Referências

AlBreiki, H. H. and Mahmoud, Q. H. (2014). Evaluation of static analysis tools for software security. In *2014 10th International Conference on Innovations in Information Technology (IIT)*, pages 93–98.

Aziz, N., Shamsuddin, S., and Hassan, N. (2016). Inculcating secure coding for beginners. pages 164–168.

Bertoglio, D. and Zorzo, A. (2017). Overview and open issues on penetration test. *Journal of the Brazilian Computer Society*, 23.

Cai, J., Zou, P., Ma, J., and He, J. (2016). Sworddta: A dynamic taint analysis tool for software vulnerability detection. *Wuhan University Journal of Natural Sciences*, 21:10–20.

Dashevskyi, S., dos Santos, D. R., Massacci, F., and Sabetta, A. (2019). Testrex: a framework for repeatable exploits. *International Journal on Software Tools for Technology Transfer*, 21(1):105–119.

de Jiménez, R. E. L. (2016). Pentesting on web applications using ethical - hacking. In *2016 IEEE 36th Central American and Panama Convention (CONCAPAN XXXVI)*, pages 1–6.

Deng, M., Wuyts, K., Scandariato, R., Preneel, B., and Joosen, W. (2011). A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements. *Requirements Engineering*, 16(1):3–32.

Dong, Y., Zhang, Y., Ma, H., Wu, Q., Liu, Q., Wang, K., and Wang, W. (2018). An adaptive system for detecting malicious queries in web attacks. *Science China Information Sciences*, 61.

Erdogan, G., Li, Y., Runde, R., Seehusen, F., and Stølen, K. (2014). Approaches for the combined use of risk analysis and testing: A systematic literature review. *International Journal on Software Tools for Technology Transfer*, 16.

Felderer, M. and Katt, B. (2015). A process for mastering security evolution in the development lifecycle. *International Journal on Software Tools for Technology Transfer*, 17.

Gauthier, F. and Merlo, E. (2012). Fast detection of access control vulnerabilities in php applications. In *2012 19th Working Conference on Reverse Engineering*, pages 247–256.

Gupta, S. and Gupta, B. B. (2017). Cross-site scripting (xss) attacks and defense mechanisms: classification and state-of-the-art. *International Journal of System Assurance Engineering and Management*, 8(1):512–530.

Gupta, S. and Gupta, B. B. (2018). Evaluation and monitoring of xss defensive solutions: a survey, open research issues and future directions. *Journal of Ambient Intelligence and Humanized Computing*.

Hayrapetian, A. and Raje, R. (2018). Empirically analyzing and evaluating security features in software requirements. pages 1–11.

Johari, R. and Sharma, P. (2012). A survey on web application vulnerabilities (sqlia, xss) exploitation and security engine for sql injection. *Proceedings - International Conference on Communication Systems and Network Technologies, CSNT 2012*.

Johns, M. (2008). On javascript malware and related threats. *Journal in Computer Virology*, 4:161–178.

Kaur, N. and Kaur, P. (2014). Mitigation of sql injection attacks using threat modeling. *SIGSOFT Softw. Eng. Notes*, 39(6):1–6.

Kim, D. and Solomon, M. G. (2018). *Fundamentals of Information Systems Security*. Jones & Bartlett Learning.

Kissel, R., Stine, K. M., Scholl, M. A., Rossman, H., Fahlsing, J., and Gulick, J. (2008). Sp 800-64 rev. 2. security considerations in the system development life cycle. Technical report, Gaithersburg, MD, USA.

Kitchenham, B. (2004). Procedures for performing systematic reviews. *Keele, UK, Keele Univ.*, 33.

Martínez, R., Betarte, G., and Pardo, A. (2018). Web application attacks detection using machine learning techniques.

Masood, A. and Java, J. (2015). Static analysis for web service security - tools & techniques for a secure development life cycle. pages 1–6.

Microsoft, C. (2012). *Security Development Lifecycle*, 5.2 edition.

MITRE (2019). Cve, common vulnerabilities and exposures. Available from MITRE.

Nagpal, B., Chauhan, N., and Singh, N. (2017). Secsix: security engine for csrf, sql injection and xss attacks. *International Journal of System Assurance Engineering and Management*, 8(2):631–644.

Nagpure, S. and Kurkure, S. (2017). Vulnerability assessment and penetration testing of web application. In *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA)*, pages 1–6.

Nunes, P., Medeiros, I., Fonseca, J., Neves, N., Correia, M., and Vieira, M. (2018). An empirical study on combining diverse static analysis tools for web security vulnerabilities based on development scenarios. *Computing*.

OWASP, T. O. W. A. S. P. (2017a). Owasp top 10 - 2017. Publication, OWASP Foundation.

OWASP, T. O. W. A. S. P. (2017b). Secure software development lifecycle project(s-sdlc). Publication, OWASP Foundation.

Radwan, H. and Prole, K. (2015). Code pulse: Real-time code coverage for penetration testing activities. In *2015 IEEE International Symposium on Technologies for Homeland Security (HST)*, pages 1–6.

Rafique, S., Humayun, M., Hamid, B., Abbas, A., Akhtar, M., and Iqbal, K. (2015). Web application security vulnerabilities detection approaches: A systematic mapping study. pages 1–6.

Sahu, D. R. and Tomar, D. S. (2017). Analysis of web application code vulnerabilities using secure coding standards. *Arabian Journal for Science and Engineering*, 42(2):885–895.

Salva, S. and Regainia, L. (2019). An approach for guiding developers in the choice of security solutions and in the generation of concrete test cases. *Software Quality Journal*, 27:675–.

Salva, S. and Zafimiharisoa, S. R. (2015). Apset, an android application security testing tool for detecting intent-based vulnerabilities. *International Journal on Software Tools for Technology Transfer*, 17(2):201–221.

Sammy Migues, John Steven and Mike Ware (2019). *BSIMM10 SSDL*, 10 edition.

Scandariato, R., Wuyts, K., and Joosen, W. (2013). A descriptive study of microsoft's threat modeling technique. *Requirements Engineering*, 20.

Shah, S. and Mehtre, B. M. (2015). An overview of vulnerability assessment and penetration testing techniques. *Journal of Computer Virology and Hacking Techniques*, 11(1):27–49.

Sharma, P., Johari, R., and Sarma, S. S. (2012). Integrated approach to prevent sql injection attack and reflected cross site scripting attack. *International Journal of System Assurance Engineering and Management*, 3(4):343–351.

Singh, H. and Dua, M. (2018). Website attacks: Challenges and preventive methodologies. In *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*, pages 381–387.

Skouby, K., Tadayoni, R., and Tweneboah-Koduah, S. (2017). Cyber security threats to iot applications and service domains. *Wireless Personal Communications*.

Trunde, H. and Weippl, E. (2015). Wordpress security: An analysis based on publicly available exploits. In *Proceedings of the 17th International Conference on Information Integration and Web-Based Applications & Services*, New York, NY, USA. ACM.

Uto, N. (2013). *Teste de Invasão de Aplicações Web*. Escola Superior de Redes.

Vibhandik, R. and Bose, A. K. (2015). Vulnerability assessment of web applications - a testing approach. In *2015 Forth International Conference on e-Technologies and Networks for Development (ICeND)*, pages 1–6.

Visoottiviseth, V., Akarasiriwong, P., Chaiyasart, S., and Chotivatunyu, S. (2017). Pentos: Penetration testing tool for internet of thing devices. In *TENCON 2017 - 2017 IEEE Region 10 Conference*, pages 2279–2284.

Zenah, N. H. Z. and Aziz, N. A. (2011). Secure coding in software development. In *2011 Malaysian Conference in Software Engineering*, pages 458–464.

Zlomislić, V., Fertalj, K., and Sruk, V. (2017). Denial of service attacks, defences and research challenges. *Cluster Computing*, 20(1):661–671.