ANDERSON ALMEIDA DOS SANTOS

ARIADYNE SILVA DE OLIVEIRA

LAURA REGINA DE SANTANA

LUCIANE HISSAE TANAKA MOREIRA

MARCIA FLORENCIO LEITE

CÓDIGO ABERTO: Guia com as melhores práticas para o desenvolvimento de aplicações seguras

São Caetano do Sul

2022

ANDERSON ALMEIDA DOS SANTOS
ARIADYNE SILVA DE OLIVEIRA
LAURA REGINA DE SANTANA
LUCIANE HISSAE TANAKA MOREIRA
MARCIA FLORENCIO LEITE

CÓDIGO ABERTO: Guia com as melhores práticas para o desenvolvimento de aplicações seguras

Trabalho de Graduação apresentado à Faculdade de Tecnologia de São Caetano do Sul, como requisito parcial para obtenção do grau de Tecnólogo em Segurança da Informação.
Orientador(a): Profª Drª. Edna Mataruco Duarte

São Caetano do Sul
2022

ANDERSON ALMEIDA DOS SANTOS
ARIADYNE SILVA DE OLIVEIRA
LAURA REGINA DE SANTANA
LUCIANE HISSAE TANAKA MOREIRA
MARCIA FLORENCIO LEITE

CÓDIGO ABERTO: Guia com as melhores práticas para o desenvolvimento de aplicações seguras

Trabalho de Graduação apresentado à Faculdade de Tecnologia de São Caetano do Sul, como requisito parcial para a obtenção do grau de Tecnólogo em Segurança da Informação

Banca Examinadora:

_____

Professora Dra. Edna Mataruco Duarte - Orientadora

_____

Examinador 1 – Banca

_____

Examinador 2 - Banca

SÃO CAETANO DO SUL

2022

Dedicamos a todos que participaram desta nossa jornada chamada vida acadêmica, até este momento.

Dedicamos também a todos que puderem contribuir com este projeto no futuro.

AGRADECIMENTOS

Agradecemos a cada integrante deste grupo e a nossa Orientadora Profª. Drª. Edna Mataruco Duarte, por transformar este projeto em realidade.

Agradecemos aos nossos professores e professoras, que compartilharam os seus conhecimentos com a gente.

Agradecemos às nossas famílias por nos apoiarem e por compreenderem a nossa ausência em alguns momentos.

“O mundo não está em seus livros e mapas. Ele está lá fora!”

(Gandalf, Senhor dos Anéis - filme)

RESUMO

SANTOS, Anderson Almeida dos; OLIVEIRA, Ariadyne Silva de; SANTANA, Laura Regina de; MOREIRA, Luciane Hissae Tanaka Moreira; LEITE, Marcia Florencio. **Código Aberto:** Guia com as melhores práticas para o desenvolvimento de aplicações seguras. **Trabalho de Conclusão de Curso**. FATEC São Caetano do Sul – Antônio Russo. 2022. 87 folhas.

É cada vez mais comum identificar falhas de segurança em códigos de *software*, e quando essas falhas não são tratadas de forma inicial no projeto e/ou no ciclo inteiro usuários tendem a encontrar vulnerabilidades no futuro. Levando em conta o potencial problema que essas vulnerabilidades podem trazer, este estudo tem como objetivo apresentar o roteiro para o desenvolvimento de um *software* seguro, utilizando aplicações com código aberto, tendo como base o Ciclo de Desenvolvimento Seguro da Microsoft, o *Open Source Security Foundation* (OSSF), o *Open Web Application Security Project* (OWASP), os processos AST e o *benchmark* de duas ferramentas SAST: *Fortify* e *Sonarqube*. Este trabalho é uma pesquisa bibliográfica onde foram levantados materiais de apoio para construção do referencial teórico, buscando informações em bibliotecas *Open Source* e guias feitos por empresas, igualmente buscamos em livros, artigos, dissertações e teses a respeito do uso de bibliotecas *Open Source* e medidas de segurança durante o desenvolvimento de *software*. O resultado deste trabalho é voltado para desenvolvedores ou pessoas que atuam no ciclo de desenvolvimento de um *software* de forma que possa agregar na qualidade do código e nos cuidados para evitar falhas de segurança, posterior ao processo de desenvolvimento do *software*.

Palavras-chave: *Software*, desenvolvimento seguro, código-aberto, *design* de segurança.

ABSTRACT

SANTOS, Anderson Almeida dos; OLIVEIRA, Ariadyne Silva de; SANTANA, Laura Regina de; MOREIRA, Luciane Hissae Tanaka Moreira; LEITE, Marcia Florencio. **Código Aberto:** Guia com as melhores práticas para o desenvolvimento de aplicações seguras. **Trabalho de Conclusão de Curso**. FATEC São Caetano do Sul – Antônio Russo. 2022. 87 pages.

It has become more and more common identifying flaws of security in software codes, and when these flaws are not addressed early in the project and/or throughout the entire cycle users will tend to complain about vulnerabilities identified later. Considering the potential problem that these vulnerabilities can bring, this project aims at presenting a roadmap to the development a software that can provide security in applications using open source, based on the Microsoft Secure Development Cycle, Open-Source Security Foundation (OSSF), Open Web Application Security Project (OWASP), AST processes, and the benchmark of two SAST tools: Fortify and Sonarqube. This study is bibliographic research in which support materials were raised for the construction of the theoretical framework, searching information about safe development in open-source libraries, and guides produced by enterprises, likewise we researched in books, articles, dissertations, regarding the use of open-source libraries and security measures during software development. The result of this project is focused on developers or people that are acting on the development software cycle to aggregate in the code quality, and in its care to avoid security flaws, after the software development process.

Keywords: Software, safe development, open-source, security by design.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

Banco Central	BACEN
Integração e Entregas Contínuas	CI/CD
Curso de Especialização em Gestão da Segurança da Informação e Comunicações	CEGSIC
Common Vulnerabilities and Exposures	CVE
Common Weakness Enumeration	CWE
Development Operational	DevOps
Development Security Operational	DevSecOps
General Data Protection Regularment	GDPR
Hyper Text Transfer Protocol (Secure)	HTTP(S)
Inteligência Artificial	IA
Internet of Things	IoT
International Organization of Standardization / International	ISO/IEC
Electrotechnical Commission	
Lei Geral de Proteção de Dados	LGPD
Multi Fator de Autenticação	MFA
National Institute of Standards and Technology	NIST
Open Source Security	OSS
Open Source Security Foundation	OSSF
Open Web Application Security Project	OWASP
Payment Card Industry Data Security Standard	PCI DSS
Quality Assurance	QA
Security Development Lifecycle	SDL
Segurança da Informação / Information Security	SI / IS
Transport Layer Security	TLS

Sumário

INTRODUÇÃO

A segurança não se faz somente com antivírus, uso de *firewall*, regras e controle de portas de acesso. Para que as vulnerabilidades possam ser minimizadas, é de extrema importância também garantir e visar a segurança em todo o ciclo de desenvolvimento do *software.*

Em aplicações Open Source, é possível realizar alterações no código fonte e trabalhar com bibliotecas, com objetivo de encontrar soluções para aprimoramento do programa.

A biblioteca é uma coleção de *scripts* feitos por diversos desenvolvedores, que publicam suas soluções para outras pessoas usufruírem do conteúdo.

Porém, há um risco muito grande ao desenvolver um *software* de código aberto, pois existe a possibilidade de um agente externo realizar a injeção de um código malicioso no *script* e acabar explorando vulnerabilidades existentes no ambiente, ocasionando um incidente de Segurança da Informação. Tudo isto, passando despercebido pelos desenvolvedores de uma organização.

Diante deste cenário, o objetivo deste trabalho de conclusão é apresentar um roteiro para o desenvolvimento de *softwares* seguros em aplicações que utilizem código aberto, tendo como base o Ciclo de Desenvolvimento Seguro da Microsoft, o *Open Source Security Foundation* (OSSF), o *Open Web Application Security Project* (OWASP), os processos AST e o *benchmark* de duas ferramentas SAST: Fortify e Sonarqube.

A necessidade de pesquisas que tratam desse tema se justifica à medida que notícias como a apresentada recentemente pelo CanalTech, que relata a existência de um *malware* que rouba dados de cartão e armazena na biblioteca oficial da linguagem do *python*, se fazem presentes nos ambientes. Segundo notificação da PyPI (2021, *online*), a praga estava presente em oito pacotes disponibilizados publicamente, que de acordo com os números oficiais, teriam acumulado mais de 30 mil *downloads* — esse total, entretanto, pode ser questionável.

Vale ressaltar que esse não é o único caso em que se tem relato da utilização de bibliotecas de desenvolvimento para implantar malware na empresa, conforme Espitia no ano passado [2019], vários exemplos dessa prática foram encontrados usando principalmente os pacotes da biblioteca NPM e pacotes da biblioteca *python*

> [...] Nos pacotes da biblioteca NPMjs, essa técnica foi detectada em diversas ocasiões, usando pacotes como twilio, que tem cerca de 500 mil downloads, para criar um pacote malicioso que usa seu reconhecimento para se fazer passar por ele, com o pacote twilio-npm, que com apenas 3 dias online consegui 371 downloads (Espitia, 2020, *online*)

Este processo tem um valor inestimável para os criminosos, pois nesta prática é possível disseminar o *malware* sem que os desenvolvedores saibam, e dificultam a rastreabilidade de quem está distribuindo estes segmentos de código.

Este trabalho se caracteriza como uma pesquisa bibliográfica, no qual serão levantados os materiais de apoio para construção do referencial teórico, buscando informações sobre o desenvolvimento seguro em bibliotecas *open source.* Empresas que montaram guias também serão utilizados, e ainda, livros, artigos, dissertações, teses a respeito do uso de bibliotecas *open source* e medidas de segurança durante o desenvolvimento de *software*. Uma vez finalizada esta parte da pesquisa, será desenvolvido o produto tendo como princípio as informações levantadas no referencial teórico.

As bibliotecas *open source* são repositórios de *scripts* públicos para auxiliar os desenvolvedores da comunidade em seus projetos. Porém, pessoas mal-intencionadas se aproveitam e adicionam seus *scripts* maliciosos nas bibliotecas. Por isso, para que se mitigue os riscos de o desenvolvedor criar uma vulnerabilidade na aplicação ao consultar as bibliotecas open source, é necessário que desenvolvedores pensem em segurança desde o início da concepção de uma nova aplicação ou atualização do *software* já existente.

Com isso, o presente estudo busca contribuir com o desenvolvimento de um *software* seguro, pontuando alternativas e boas práticas, sendo esse o produto. Assim, ao final será apresentado um guia com as melhores práticas de desenvolvimento seguro, tendo como foco código aberto, pois entende-se que *security by design é* a melhor alternativa diante deste cenário.

Para um melhor entendimento, o trabalho foi dividido em quatro capítulos, começando com a introdução, que apresentou a contextualização, objetivo, justificativa e metodologia. Em seguida, no Capítulo 1 foi apresentado o ciclo de desenvolvimento seguro e a proposta da *Microsoft* SDL. No capítulo 2, foram apresentadas as fundações de apoio para segurança de *software*, que são

organizações já existentes com o objetivo em comum de trazer maior segurança nas aplicações OWASP e OpenSSF. Em seguida, nos capítulos 3 e 4 foram abordadas as ferramentas AST e apresentado um *benchmark sobre* as ferramentas *Sonarqube* e *Fortify*. No Capítulo 5, foi apresentado o produto, um compilado com as melhores práticas do mercado relacionadas ao desenvolvimento seguro, para orientar e questionar o desenvolvedor sobre a aplicação durante o desenvolvimento do *software*. Por fim, são apresentadas as considerações finais sobre o tema e as referências bibliográficas utilizadas para este projeto.

1. CICLO DE DESENVOLVIMENTO SEGURO

Uma vez que o ambiente da *World Wide Web* concentra pessoas de todos os cenários do globo, qualquer que seja sua localização geográfica ou seu *background* cultural, são frequentes as preocupações com questões de segurança no desenvolvimento das aplicações *web*. Paralelamente à necessidade de desenvolver *softwares* seguros, há a questão comercial, que leva os profissionais de segurança da informação a considerar as vantagens de cada tipo de aplicação conforme as necessidades dos usuários.

Para cada cliente há uma demanda a ser satisfeita. Com base nisso, diversas metodologias de desenvolvimento de sistemas podem ser úteis, de acordo com o levantamento de requisitos de cada negócio. Cada negócio exige um tipo de operação para que a máxima performance seja atingida e os riscos sejam minimizados. Com o pleno conhecimento de cada uma das fases do ciclo de desenvolvimento do *software*, a aplicação trará o rendimento esperado, caracterizando alto retorno sobre o investimento.

> A integração entre os objetivos do negócio, os processos de negócio e sistemas de informação é um fator determinante da dinâmica necessária à organização e também um desafio aos gerentes. Nesse cenário, os sistemas de informações são os habilitadores do negócio e, portanto, precisam estar alinhados com os reais objetivos deste negócio (AZEVEDO JUNIOR; CAMPOS, 2008, p.27).

Considerar o ciclo de desenvolvimento de um *software* está no centro de um planejamento coerente e capaz de evitar riscos. Fazer a concepção do modelo correto para cada aplicação permite que os gestores mantenham seus times envolvidos com a operação, consolidando de forma sustentável a funcionalidade do sistema. Não basta apenas escolher qual a linguagem que será utilizada na construção do *software*, mas também deve ser considerada a importância da metodologia de desenvolvimento, que fará com que ele exista e seja produtivo.

Conhecer as fases do ciclo de desenvolvimento de *software* é algo que poderá contribuir com o entendimento das vulnerabilidades existentes. Por mais que existem diferenças entre elas, quatro fases são imprescindíveis para que qualquer metodologia seja eficiente: Definição, Desenvolvimento e Manutenção.

Seria impossível desenvolver uma aplicação sem avaliar de que forma isso ocorre, quais os benefícios e riscos envolvidos no processo, quais são as necessidades do cliente e os resultados que o *software* proporcionará. Na fase de definição, acontece a projeção da arquitetura, e ainda pode incluir a construção de protótipos.

A fase de desenvolvimento é quando se começa a construir o *software*, é a codificação em si. Além disso, ela ainda engloba a elaboração de planos de testes integrados e a delimitação do cronograma de quando acontecerão, além da criação ou atualização do manual do *software*. O processo de desenvolvimento pode ser iterativo (gradualmente se tornando mais complexo e refinado) ou incremental (entrega por subconjuntos até que o *software* possa ser implementado em sua totalidade).

Na etapa Manutenção, a prioridade é a estratégia de consolidação do *software* em construção, o que inclui a análise de *hardwares* e *softwares* já mantidos pelo cliente, para que a transição ocorra bem. Assim como o treino dos usuários, a realização de testes *in loco* e a previsão de quando ocorrerão as manutenções. Nessas manutenções, o ciclo de desenvolvimento pode retroceder ou até mesmo reiniciar, com objetivo de serem realizadas atualizações e melhorias.

1.1. MODELOS DE ENGENHARIA DE *SOFTWARE*

As etapas de desenvolvimento de *software* podem ter mais ou menos mudanças de acordo com o modelo de engenharia adotado. Modelos clássicos ainda são eficientes nos dias de hoje, porém, com a dinamização dos processos gerenciais, as empresas buscam cada vez mais otimizar suas tarefas, o que nos leva a modelos mais modernos de desenvolvimento, como as metodologias ágeis.

> Quatro modelos de engenharia de software têm sido amplamente discutidos: o ciclo de vida clássico (ou cascata), a prototipação, o modelo espiral e as técnicas de Quarta Geração (PRESSMAN, 2002). Atualmente um novo modelo tem sido bastante usado: o modelo iterativo e incremental (JACOBSON; BOOCH; RUMBAUGH, 1999; PAULA FILHO, 200) (AZEVEDO JUNIOR; CAMPOS, 2008, p.28).

As metodologias ágeis surgiram com o Manifesto Ágil (2001), resultado da reunião de 17 estudiosos com o objetivo de otimizar a relação entre indivíduos, os processos de desenvolvimento de *software* e as mudanças de mercado. Com isso,

em vez do modelo clássico em cascata, alternativas foram criadas e apresentadas, como as metodologias *Scrum* e XP.

> Para os estudiosos da abordagem ágil, o cliente adquire conhecimento sobre seu produto durante o seu desenvolvimento, gerando mudanças ao longo deste processo. Este processo busca dividir o projeto em pequenas entregas ao longo do cronograma, onde o cliente pode avaliar o desenvolvimento, agregar valor ao negócio, priorizar e solicitar mudanças de funcionalidades (OLIVEIRA; SEABRA, 2015, p. 28)

Dentro deste panorama e da importância de se aplicar o desenvolvimento seguro de *softwares* e aplicação, a Microsoft desenvolveu um guia para auxiliar os desenvolvedores aplicarem medidas de segurança e privacidade em todo o ciclo de vida do desenvolvimento, conhecido como *Microsoft Security Development Lifecycle* (SDL).

1.2. *MICROSOFT SECURITY DEVELOPMENT LIFECYCLE* (*SDL*)

A Microsoft publicou um guia oficial e online para os desenvolvedores com as melhores práticas durante o desenvolvimento de *software*/ aplicação. O Microsoft *SDL* nasceu em 2004, no mês de janeiro, com o propósito de reduzir custos e aumentar a confiabilidade do *software* em relação a segurança. A Microsoft foi pioneira e vem otimizando esse processo até os dias de hoje. Pode-se afirmar que o *Security Development Lifecycle (SDL*):

> É um processo de garantia de segurança que tem seu foco no desenvolvimento de software. Com uma iniciativa que abrange toda a empresa e uma política obrigatória desde 2004, o SDL desempenhou uma função essencial na segurança e na privacidade incorporadas ao software e à cultura da Microsoft. Combinando uma abordagem holística e prática, o SDL tem o objetivo de reduzir o número e a gravidade das vulnerabilidades dos softwares. O SDL introduz segurança e privacidade em todas as fases do processo de desenvolvimento. (MICROSOFT SDL, 2010, p.3)

Para que o Microsoft SDL tenha efeito, é preciso foco em três componentes indispensáveis: Educação, Melhoria Contínua dos Processos e Responsabilidades. As empresas, de um modo geral, devem voltar parte de seus investimentos para a transferência de conhecimento, como pontua o Guia Microsoft SDL, sendo uma forma de adequar as mudanças tecnológicas com o panorama das ameaças. Trazendo para si a proposta de melhoria contínua, pois os riscos de segurança não são estáticos,

muito pelo contrário, eles são voláteis e exigem um grau de entendimento das causas e efeitos dos riscos analisados em cada processo, e a resposta em cada incidente.

> [...] o SDL requer o arquivamento de todos os dados necessários para atender um aplicativo em uma crise. Quando associada a planos detalhados de comunicação e resposta de segurança, uma organização pode fornecer uma orientação concisa e convincente a todas as partes afetadas (MICROSOFT SDL, 2010, p.4).

Isto é possível com o armazenamento de todos os dados ao longo dos processos, para gerar métricas que auxiliem em futuras mudanças. O Microsoft SD*L* pode ser simplificado em 7 tópicos: Treinamento, Requisitos, *Design*, Implementação, Verificação, Liberação e por fim Resposta.

1.2.1. Treinamento

Antes de utilizar qualquer plataforma de desenvolvimento, é necessário treino nas práticas de segurança e privacidade do ambiente da Azure[1], pois ao realizar essa etapa, poderá reduzir o número e a gravidade das vulnerabilidades exploráveis em seu aplicativo. Assim, estará mais preparado para reagir adequadamente ao panorama de ameaças em constante mudança (LANFEAR, 2021, *online*).

Diante disso, a *Microsoft Azure* disponibiliza 7 tipos de treinamento para desenvolvedores antes de programar, que são:

- Guia do desenvolvedor para o Azure2 para mostrar como utilizar a Azure e seus serviços disponíveis para armazenar dados, executar aplicativos, utilizar a IA e aplicações de IoT3 de forma segura e eficiente;
- Guia de introdução para desenvolvedores do Azure4: voltado para desenvolvedores que buscam começar a utilizar a plataforma Azure para as suas próprias necessidades de desenvolvimento;
- SDKs e ferramentas5: apresenta as ferramentas disponíveis para os desenvolvedores dentro da Azure;
- Azure DevOps Services6: fornece aos desenvolvedores ferramentas de colaboração de desenvolvimento, como o GitHub, pipelines, quadros kanban;

[1] Ressaltando que este capítulo fala sobre o *Microsoft SDL*, por isto abordará o desenvolvimento na Azure. Caso o desenvolvedor utilize qualquer outra plataforma, ele deverá realizar treinamento de segurança e privacidade da plataforma em nuvem que ele for desenvolver.
[2] Disponível em: https://azure.microsoft.com/campaigns/developer-guide/
[3] IoT: Sigla de *Internet of Things* ou internet das coisas.
[4] Disponível em: https://docs.microsoft.com/pt-br/azure/guides/developer/azure-developer-guide
[5] Disponível em: https://docs.microsoft.com/pt-br/azure/?pivot=sdkstools
[6] Disponível em: https://docs.microsoft.com/pt-br/azure/devops/

- Central de Recursos de DevOps7: a combinação de recursos de aprendizagens de práticas DevOps com métodos Agile8;
- Os cinco principais itens de segurança a serem considerados antes de enviar para produção9, ensina métodos de proteção aos aplicativos Web na Azure;
- Kit de Segurança de DevOps para Azure10: este kit contém diversas ferramentas, scripts, extensões e automações para que a aplicação desenvolvida esteja de acordo com as melhores práticas de segurança.
- As melhores práticas de segurança para soluções do Azure11: uma coletânea das melhores práticas de segurança para proteger, implantar e gerenciar as aplicações usando a nuvem Azure.

Portanto, todos os responsáveis pela esteira de desenvolvimento devem treinar os requisitos básicos de segurança e suas tendências.

1.2.2. Requisitos

Essa é uma fase crucial dentro do desenvolvimento seguro, pois é o momento que o desenvolvedor define o que será a aplicação e qual a sua utilidade. Mas também é uma fase para pensar sobre os controles de segurança que devem conter na aplicação a ser desenvolvida. Conforme pontua Terry Lanfear e Olprod no Microsofts Docs (2021), deve-se considerar além das questões de segurança, as problemáticas relacionadas à privacidade, definindo níveis aceitáveis juntamente com a equipe, tais níveis:

> Entenda os riscos associados a problemas de segurança. Identificar e corrigir bugs de segurança durante o desenvolvimento. Aplique níveis estabelecidos de segurança e privacidade em todo o projeto (Lanfear, 2021, online.)

Vale ressaltar que esta equipe é formada por desenvolvedores *Quality Assurance (QA)* e analista de segurança, para conseguir abranger todas estas demandas de segurança e que cumpram os princípios básicos da ISO 27000: confidencialidade, integridade e disponibilidade dos dados. Com o objetivo de mapear

---

[7] Disponível em: https://docs.microsoft.com/pt-br/azure/devops/learn/

[8] Conforme Gomes, Agil é Disciplina composta por comportamentos, processos, práticas e ferramentas utilizados para a criação de produtos e sua subsequente disponibilização para os usuários finais. Disponível em: https://www.linkedin.com/pulse/o-que-%C3%A9-%C3%A1gil-uma-nova-defini%C3%A7%C3%A3o-formal-andr%C3%A9-gomes

[9] Disponível em: https://docs.microsoft.com/pt-br/learn/modules/top-5-security-items-to-consider/index?WT.mc_id=Learn-Blog-tajanca

[10] Disponível em: https://azsk.azurewebsites.net/index.html

[11] Disponível em: https://azure.microsoft.com/resources/security-best-practices-for-azure-solutions

os requisitos mínimos de privacidade e segurança, especificando cada item para um futuro monitoramento de vulnerabilidades.

Lanfear e Olprod (2021) pontuam, ainda no artigo, sobre criar aplicativos seguros no Azure, com uma lista de perguntas a serem realizadas nesta etapa:

> Meu aplicativo contém dados confidenciais? Meu aplicativo coleta ou armazena dados que exigem a adesão a padrões do setor e programas de conformidade como o FFIEC (Federal Financial Institution Examination Council) ou o PCI DSS (Padrão de Segurança de Dados do Setor de Cartões de Pagamento)? O meu aplicativo coleta ou contém dados confidenciais pessoais ou dos clientes que podem ser usados, por conta própria ou com outras informações, para identificar, contatar ou localizar uma pessoa? Meu aplicativo coleta ou contém dados que podem ser usados para acessar informações médicas, acadêmicas, financeiras ou trabalhistas de um indivíduo? [...] Onde e como meus dados são armazenados? [...] Será possível influenciar o registro em log para coletar dados mais detalhados e analisar um problema detalhadamente? Meu aplicativo estará disponível para o público (na internet) ou apenas internamente? Você entende o seu modelo de identidade antes de começar a projetar o aplicativo? Como você determinará que os usuários são quem dizem ser e o que um usuário está autorizado a fazer? O meu aplicativo executa tarefas confidenciais ou importantes (como transferir dinheiro, destrancar portas ou entregar remédios)? [...] Meu aplicativo executa atividades de software arriscadas, como permitir que usuários carreguem ou baixem arquivos ou outros dados? [...](Lanfear, 2021, online).

Dessa forma é possível identificar o nível de confidencialidade e as ferramentas necessárias para sua proteção. Ter um requisito de monitoramento de armazenamento dos dados da aplicação, permitindo coleta e armazenamento de *logs*. Ferramentas de autenticação conforme o nível de permissão de acesso, com autenticação e autorização. E ter como garantir a proteção do usuário contra dados e arquivos mal-intencionados.

1.2.3. *Design*

Implantar o *design* correto no início do projeto é de suma importância, pois permite que sejam consideradas as questões de segurança e privacidade, tornando-se uma opção mais barata quando feita no estágio inicial do projeto.

> O momento ideal para influenciar a confiabilidade do design de um projeto é o início de seu ciclo de vida. É muito importante considerar as questões de segurança e de privacidade cuidadosamente durante a fase de design. A mitigação dos problemas de segurança e de privacidade é muito mais barata quando realizada durante os estágios iniciais do ciclo de vida de um projeto. As equipes de projeto devem parar de usar a prática de separar os recursos e as mitigações de

> segurança e de privacidade perto do fim do desenvolvimento de um projeto (MICROSOFT SDL, 2010, p.10).

O Microsoft SDL reforça que é muito importante entender a diferença entre “recursos seguros” e “recursos de segurança”:

> Recursos seguros são definidos como recursos cuja funcionalidade é bem-estruturada do ponto de vista da engenharia com respeito à segurança, incluindo a validação rigorosa de todos os dados antes do processamento ou da implementação criptograficamente robusta de bibliotecas para o serviço de criptografia. O termo recursos de segurança descreve a funcionalidade do programa com as implicações de segurança, como a autenticação Kerberos ou um firewall. (Microsoft SDL, 2010, p.10).

Reduzir a superfície de ataque e a modelagem de ameaças são estratégias parecidas com abordagens específicas. Reduzir a superfície de ataque significa diminuir/fechar/restringir o acesso aos serviços e sistemas, considerando a aplicação de privilégio mínimo e aumentando as camadas de segurança. A modelagem de ameaças entra em vigor quando existe um risco significativo de segurança. E impõe que as equipes considerem, documentem e discutam as dificuldades de segurança.

Portanto, esta é a fase ideal para estabelecer as melhores práticas para *design* e especificações funcionais da aplicação, conforme pontua Lanfear (2021), pois esta fase é essencial para realização da análise de risco da aplicação, auxiliando a redução de problemas de segurança e privacidade em um projeto.

O conceito de *Design* Seguro de Lanfear (2021), é quando se pode minimizar ou evitar eventos de falha de segurança, evitando de colocar em risco toda a aplicação desenvolvida e dados dos envolvidos, porque uma falha de segurança é a “supervisão no *design* do aplicativo que pode permitir que um usuário execute ações mal-intencionadas ou inesperadas após o lançamento do aplicativo (LANFEAR, 2021, *online*)”.

Com a ideia de um *design* seguro, Lanfear (2021) orienta aplicar os seguintes controles de segurança ao projetar aplicativos seguros:

- Usar uma biblioteca de codificação segura e uma estrutura de *software*.
- Verificar se há componentes vulneráveis.
- Usar a modelagem de ameaças durante o *design* do aplicativo.
- Reduzir a superfície de ataque.
- Adotar uma política de identidade, como o perímetro de segurança primário.

- Exigir uma nova autenticação para transações importantes.
- Usar uma solução de gerenciamento de chaves para proteger chaves, credenciais e outros segredos.
- Proteger dados confidenciais.
- Implementar medidas à prova de falhas.
- Aproveitar o tratamento de erros e exceções.
- Usar o registro em *log* e os alertas.

1.2.4. Implementação

Na implementação, o Microsoft SDL propõe que se utilize ferramentas aprovadas e que foram previamente homologadas pelo consultor de segurança do projeto, possibilitando que os desenvolvedores utilizem ferramentas recentes e aproveitem novas funcionalidades de análise de segurança. É importante que o time também desaprove funções não seguras, isso consiste em analisar e levar em consideração o ambiente atual de ameaças e proíba o uso de funções e *APIs*[12] (*Application Programming Interface*) comprometidas e/ou não seguras.

O Microsoft SDL define o passo de análise estática como uma fase importante de verificação do código fonte e entende que:

> A análise de código estática sozinha é geralmente insuficiente para substituir uma revisão de código manual. A equipe de segurança e os consultores de segurança devem estar cientes das forças e fraquezas das ferramentas de análise estática e devem estar preparados para acrescentar às ferramentas de análise estática outras ferramentas ou revisão humana, como for apropriado (Microsoft SDL, 2010, p.12)

Para Lanfear (2021), na fase de implementação o foco é estabelecer as práticas de prevenção antecipada, detecção e remoção dos problemas de segurança relacionados ao código. Quem normalmente realiza esta tarefa são os *Quality Assurance (QA), responsáveis* por testar o *script* e a aplicação de maneiras não usuais, para evitar seu uso indevido. Quem normalmente realiza esta tarefa são os *Quality Assurance (QA),* responsáveis por testar o *script* e a aplicação de maneiras não usuais, para evitar seu uso indevido. E uma das maneiras é realizar o seguinte passo a passo:

[12] API significa interface de programação de aplicações, um conjunto de definições e protocolos para criar e integrar softwares de aplicações. (RedHat, 2017, *online*)

- Executar revisões de código, para reduzir o risco de criação de *bugs* e aumentar a qualidade geral da aplicação. Para realizar o processo de revisão do código antes da implementação, a Microsoft disponibiliza o *Visual Studio*.
- Executar análise de código estático ou a análise do código fonte é uma etapa da revisão do código, para detectar vulnerabilidades do código que não são detectadas por meio das técnicas de verificação de contaminação e análise de fluxo de dados.
- Validar e limpar todas as entradas do aplicativo e tratá-las como não confiáveis, para proteger todas as portas de entradas que possam ser exploradas por alguma ameaça, como um ataque de injeção ao código por não filtrar os dados necessários. Sendo muito usado uma listagem de bloqueados e permitidos.
- Verificar as saídas do aplicativo, assim como a entrada, é importante que as saídas de dados estejam seguras para que não haja nenhum vazamento de segurança. Uma forma é a aplicação de codificação da saída para que dados não confiáveis sejam explorados em um ataque de injeção.
- Usar consultas parametrizadas quando entrar em contato com o banco de dados, para que códigos mal-intencionados não sejam inseridos no banco de dados por ter realizado uma consulta ao banco de dados em tempo real;
- Remover cabeçalhos de servidor padrão para evitar uma impressão digital do aplicativo, revelando informações sobre servidores e tecnologias adjacentes;
- Separar os dados de produção, que é a segregação do ambiente em: ambiente de desenvolvimento, teste/homologação e produção. Esta é uma medida para garantir que não haja vazamento[13] ou a quebra dos dados.

[13] Vale ressaltar que a lei geral de proteção de dados (LGPD) traz o conceito de minimização do uso de dados, que somente aqueles que realizam algum tratamento do dado pessoal pode ter acesso e evitar acessos desnecessários.

- Implementar uma política de senha forte, para garantir uma defesa dos ataques de força bruta e adivinhação por listas prontas de senhas, garantindo um controle de acesso à aplicação e aos dados.
- Validar *uploads* de arquivos, pois a primeira etapa de muitos ataques à aplicação é a injeção de um código mal-intencionado.
- Não armazenar conteúdo confidencial em *cache*, pois estes dados são salvos em uma pasta "Arquivos da *internet*", facilitando o acesso a informações confidenciais da empresa por acesso à pasta.

1.2.5. Verificação

A fase de verificação é responsável por monitorar o comportamento da aplicação quanto a problemas com memória, problemas de privilégio e demais problemas de segurança de criticidade alta. Neste processo são utilizadas ferramentas de tempo de execução como o *AppVerifier*[14] em conjunto com o teste de *fuzzing*.

O teste de *fuzzing* geralmente é automatizado ou semiautomatizado e trata-se de tentar induzir falha no programa através da inserção de dados aleatórios e/ou defeituosos. Essa prática consegue detectar partes do programa e precisa de atenção especial, sendo necessário auditar o código.

É importante revisar a superfície de ataque, isso garante que a implementação do sistema seja reconsiderada e novas possibilidades de ataques sejam mitigados.

> É comum para um aplicativo desviar significativamente das especificações funcionais e de design criadas durante as fases de requisitos e de design de um projeto de desenvolvimento de software. Desse modo, é essencial revisar os modelos de ameaça e a medição da superfície de ataque de um determinado aplicativo quando seu código estiver concluído (Microsoft SDL, 2010, p.12).

Dessa forma é validado se o código atende a todos os requisitos de segurança e privacidade que foram estabelecidos no início do ciclo. E para Lanfear (2021), além do teste de *fuzzing,* é necessário localizar e corrigir vulnerabilidades em suas

[14] O *Application Verifier* (AppVerif.exe) é uma ferramenta de verificação dinâmica para aplicativos de modo de usuário. Essa ferramenta monitora as ações do aplicativo enquanto o aplicativo é executado, sujeita o aplicativo a uma variedade de tensões e testes e gera um relatório sobre possíveis erros na execução ou design do aplicativo. Disponível em: https://docs.microsoft.com/pt-br/windows-hardware/drivers/devtest/application-verifier

dependências de aplicativo, testar o aplicativo em um estado operacional, fazer o exame da superfície de ataque, executar teste de penetração de segurança e executar testes de verificação de segurança.

1.2.6. Liberação e Reposta

A primeira prática da fase de liberação é o Plano de Resposta de Incidentes, isso inclui os programas que não possuem vulnerabilidades conhecidas e com o tempo podem sofrer ameaças. O Microsoft SDL define que o plano deve incluir:

> Uma equipe de SE (Engenharia sustentada) identificada ou, se a equipe for muito pequena para ter recursos de SE, um ERP (Plano de resposta de emergência) que identifica a equipe de engenharia, de marketing, de comunicações e de gerenciamento adequadas para agir como pontos de contato primário em uma emergência de segurança. Estão disponíveis contatos na chamada com autoridade de tomar decisões 24 horas por dia, sete dias por semana. Planos de serviço de segurança para código herdados de outros grupos dentro da organização. Planos de serviço de segurança para código de terceiros licenciados, incluindo nomes de arquivo, versões, código-fonte, informações de contato de terceiros e permissão contratual para fazer alterações (se apropriado) (Microsoft SDL, 2010, p.13).

Leanfer (2021) pontua, que o foco da versão[15] é preparar o lançamento da aplicação ao público através da verificação do desempenho do aplicativo antes da inicialização; instalação de um *firewall* de aplicativo web; a criação de um plano de resposta incidentes para resolução de novas ameaças; a condução de uma revisão de segurança final e por fim a certificação do lançamento e os arquivos.

Após a publicação, a fase de Resposta aponta que a equipe tem que ser capaz de responder aos relatórios de vulnerabilidade adequadamente e quando necessário, executar o plano de resposta a incidentes e monitorar o desempenho do aplicativo.

É importante levar em consideração um ciclo de desenvolvimento seguro como o Microsoft SDL, para garantir a equidade no processo de desenvolvimento. No próximo capítulo será abordado sobre as fundações de apoio para segurança de *software*, sua relevância e usabilidade.

[15] A autora nomeia esta fase de Versão enquanto o guia SDL da Microsoft aborda como liberação.

2. FUNDAÇÕES DE APOIO PARA SEGURANÇA DE *SOFTWARE*

As comunidades que utilizam bibliotecas de *softwares Open Source* formaram duas organizações que trazem grandes contribuições nos aspectos de segurança para os desenvolvedores de *softwares* de código aberto, a *Open Web Application Security Project®*[16] (OWASP) e a *Open Source Security Foundation (*OSSF).

A colaboração dessas organizações é de suma importância para os profissionais de TI, principalmente nos requisitos de Segurança da Informação e Privacidade exigidos pelo time de Segurança da Informação (SI), com o objetivo de evitar as falhas de segurança, proteger os sistemas das tentativas de acessos não autorizados, além mitigar as possibilidades de divulgação não autorizada de informações confidenciais da empresa e de seus usuários.

2.1. OWASP

OWASP é uma fundação sem fins lucrativos com o objetivo de melhorar a segurança do *software* com o foco em aplicações web. Auxiliando desenvolvedores e analistas a garantir a Segurança da Informação em projetos de *softwares Open Source*, permitindo que as organizações possam desenvolver, comprar e manter aplicações *API's* seguras e confiáveis.

Esta fundação é uma das mais conhecidas globalmente com dezenas de milhares de membros ao redor do globo, realizando conferências educacionais e treinamentos de lideranças. Conforme a orientação da própria OWASP deve-se aplicar a análise do OWASP Top10, conforme a figura 1, em cada etapa do desenvolvimento.

A Tabela 01 aponta tópicos similares ao SDL da Microsoft, como a necessidade de haver um plano de conscientização e treinamento em segurança do desenvolvimento; pensar em segurança desde o início do projeto e da arquitetura, tendo um padrão de desenvolvimento; também a realização de testes e verificações da aplicação garantindo a segurança.Porém, a OWASP detalha os tipos de testes realizados e não somente a realização de teste de *fuzzing* ou revisão dos códigos,

[16] Tradução: Projeto Aberto de Segurança em Aplicações Web

mas também realizar testes de penetração antes de implementar a aplicação na produção.

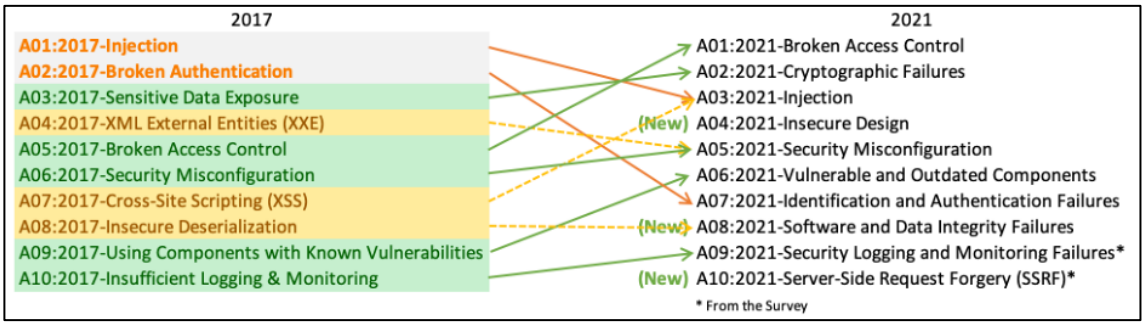
Tabela 01 – Utilizando a OWASP Top 10

Caso de uso	Top 10 OWASP 2021	Padrão de verificação de segurança de aplicativos OWASP
Consciência	Sim	
Treinamento	Nível de entrada	Compreensivo
Projeto e arquitetura	Ocasionalmente	Sim
Padrão de codificação	Mínimo	Sim
Revisão do código seguro	Mínimo	Sim
Lista de verificação de revisão por pares	Mínimo	Sim
Teste de unidade	Ocasionalmente	Sim
Teste de integração	Ocasionalmente	Sim
Teste de penetração	Mínimo	Sim
Suporte de ferramentas	Mínimo	Sim
Cadeia de Suprimentos Segura	Ocasionalmente	Sim

Fonte (OWASP,2021, *online*)

Um de seus projetos de maior destaque é o OWASP *Top Ten*, que é uma lista com os 10 principais riscos de segurança de aplicativos web. Ela foi formulada baseada na probabilidade de exploração da vulnerabilidade e o impacto em caso de sua exploração, isto baseado também em quantidade de CVEs e CWEs existentes, dados estatísticos e taxa de incidência das vulnerabilidades.

Figura 01 – OWASP *Top*10 2017 e 2021



Fonte: (OWASP, 2021, *online*)

A figura 01, traz a comparação dos *Top* 10 em 2017 e 2021, sendo possível inferir que há 3 novos tópicos, que são: A04 *Design* Inseguro, A08 Falhas de integridade do *software* e dados, e por fim o A10 Falsa solicitação de acesso ao servidor. Todas elas têm como base a arquitetura não realizada de forma segura. Nos próximos tópicos serão abordados item a item da lista T*op* 10, pois é fundamental para o desenvolvimento do guia de melhores práticas.

2.1.1. Broken Access Control

Em tradução livre, falha no controle de acesso da aplicação. Este controle tem como objetivo delimitar o acesso e as interações com o sistema para cada usuário. Caso haja falha, o sistema pode ser alvo de vazamento de dados, modificações e até mesmo a destruição dos dados. As principais vulnerabilidades que se deve atentar, conforme a OWASP (2021), são:

- Violação do princípio de privilégio mínimo;
- Modificações de solicitações de *APIs* para *APIs* inseguras;
- Permissionamento de visualização e edição de contas dos demais usuários;
- Manipulação de metadados, adulterando *tokens* e *cookies* da aplicação.

Para prevenir a falha no controle de acesso, conforme a OWASP (2021), é necessário implementar os códigos em um servidor confiável e as *APIs* sem servidor, para que a invasão não consiga acessar, nem modificar qualquer verificação ou metadados da aplicação.

- Negar por padrão o acesso a todos os recursos, exceto aqueles classificados como públicos;
- Implementar mecanismos de controle de acesso que façam a verificação em toda a aplicação;
- O controle de acesso da aplicação deve impor a propriedade de registro em vez de permitir que o usuário tenha permissão de leitura, escrita, exclusão e execução;
- Desativar as listagens de diretórios *web*;
- Certificar que os arquivos de dados e *backup* não tenham acesso à *web*;
- Monitorar, alertar e registrar as falhas para os administradores;
- Para minimizar a possibilidade de ataques automatizados, controlar a quantidade de acesso a *API* e a controladora;
- Utilização de *tokens* de curta duração;
- Seguir os procedimentos de *AuthO* para revogação de acesso.

Este tópico deve ser realizado pela equipe de desenvolvedores e *quality assurance (QA)* nos testes de integração.

2.1.2. *Cryptographic Failures*

Este controle deve ser aplicado na etapa de *Design* do *SDL* da *Microsoft* ou no Projeto de Arquitetura da aplicação na OWASP, pois o primeiro passo é definir quais dados precisarão de criptografia em trânsito e qual o nível da criptografia em repouso. Conforme a OWASP (*Open Web Application Security Project)* orienta, estas definições deverão estar de acordo com as regulamentações locais, como GDPR (*General Data Protection Regularment)* e *PCI (Payment Card Industry Data Security Standard)*, aqui no Brasil também se deve atentar ao Bacen e a LGPD (Lei Geral de Proteção de Dados) quanto aos tipos de dados que necessitam de criptografia. E segue com perguntas similares propostas pelo *SDL* da Microsoft:

- Algum dado é transmitido em texto simples?
- Ainda é usado algum protocolo/algoritmo criptografado desatualizado ou fraco?
- As chaves são reutilizadas? Há um gerenciamento ou rotação de chaves implementado?
- Há algum parâmetro de segurança/criptografia de cabeçalho? Uso de protocolos HTTP/HTTPS?
- Estão devidamente validados os certificados do servidor e a cadeia de confiança?
- As chaves criptográficas foram projetadas e planejadas para atender os requisitos de SI e Privacidade da aplicação?
- A aplicação possui o uso de *hash* obsoletos? E utilizam *hash* com ou sem criptografia quando necessário?
- Possui monitoramento, alerta e registro de falha de criptografia?

Para prevenir a falha de criptografia, conforme a OWASP (2021), é minimamente necessário que responda as perguntas anteriormente apresentadas e que:

- Utilize chaves criptográficas atualizadas;
- Utilize protocolos criptografados para as aplicações *web*;
- Não armazene dados confidenciais ou sensíveis sem necessidade. O ideal é descartar estas informações após o uso;

- Usar um gerenciador de chaves adequado, garantindo que os algoritmos, protocolos e chaves criptográficas estejam fortes e atualizados;
- Dados em trânsito sejam criptografados com protocolo seguro, como TLS e HSTS;
- Verificar se a criptografia utilizada nas APIs é segura e adequadas para cada dado trafegado;
- Sempre verificar a eficácia das configurações da criptografia.

Alinhando as melhores práticas de mercado[17], é necessário que haja uma Política e Gestão de chaves criptográficas dos dados da aplicação e um mecanismo de classificação dos dados utilizados na aplicação, por exemplo, um dado sensível de uma pessoa brasileira tem que ter medidas de proteção especial de acordo com a LGPD.

2.1.3. *Injection*

As injeções mais conhecidas são de SQL e NoSQL, conforme Rossini e Camolesi (2016): “A injeção de código é uma técnica que pretende inserir dependência em uma classe por meio de código externo” (ROSSINI, CAMOLESI; 2016, p.01). Esta técnica é muito popular em sistemas maliciosos, produzidos por *hacker* ou pessoas mal-intencionadas que queiram obter informações, acesso e até gerar uma indisponibilidade na aplicação. Porém, Rossini e Camolesi (2016) ressaltam que esta técnica também pode ser utilizada para melhorar o desempenho da aplicação.

A OWASP (2021) define como uma aplicação vulnerável a ataques de injeção de código, quando a aplicação não realiza a validação, filtra ou higieniza os dados inseridos pelo usuário; realização de uma consulta ou chamada de informações sem análise da aplicação, em relação ao contexto para que serão utilizados no interpretador; possibilidade de obter os registros confidenciais através da utilização dos dados hostis, em parâmetros de pesquisa no *object relational mapping* (ORM);

[17] ISO 27000, NIST, LGPD, GDPR

utilização dos dados hostis usado diretamente ou concatenados com o código ou o SQL malicioso.

A forma de prevenir um ataque de injeção de código é a manutenção dos dados separados dos locais de comandos e consultas. Utilizar uma API segura, para que não seja invadido durante a parametrização da aplicação; utilizar a validação positiva para o lado do servidor que haja escapes específicos nas consultas dinâmicas residuais; usar comandos para evitar consultas e vazamentos em massa do SQL; não utilizar nomes das estruturas fornecidas pelo usuário.

2.1.4. *Insecure Design*

Esta categoria concentra as falhas de arquitetura e design da aplicação. Este tópico aborda que deve-se pensar na segurança desde o início da modelagem, também conhecido como *security by design*, que implementa a modelagem das ameaças, padrões de design seguros e arquiteturas de referência. Conforme pontua Del Esporte e Bezerra (2014):

> O design ganha um grau de importância maior à medida que a complexidade dos softwares aumentam durante o desenvolvimento, pois suas consequências são diretas sobre os atributos de qualidade do software tais como flexibilidade, testabilidade, manutenibilidade, desempenho e segurança (DEL ESPORTE; BEZERRA, 2014, p.27).

E para isso, deve-se pensar na segurança desde o início do *design* do projeto. Também é necessário ressaltar a diferença entre *design* inseguro e a implementação insegura. A implementação, conforme o CEGSI 2009-2011, consiste na produção do código executável, e diferente de um *design* inseguro, tem maior probabilidade de ser corrigido. Isso porque não foram criados os mecanismos de defesa contra-ataques específicos, com base nos controles de segurança.

Portanto, pontuado pela OWASP (2021), o *Design* Seguro é uma cultura e metodologia e não um complemento que será inserido na aplicação, consiste em todo um ciclo de vida de desenvolvimento seguro. Para prevenir a vulnerabilidade de *Design* inseguro, a OWASP sugere que seja estabelecido um ciclo de desenvolvimento seguro realizado em conjunto com profissionais do *AppSec*, para avaliar e projetar controles relacionados à Segurança da Informação e privacidade; modelagem de ameaças para autenticação crítica de controle de acesso, lógica de negócio e fluxo-chave; integre linguagens de segurança entre os usuários e

desenvolvedores; integre as verificações de vulnerabilidade em cada nível do seu aplicativo; teste de integração que valida todos os fluxos críticos dos modelos de ameaças; segregação das camadas em nível de sistema e de redes, conforme as exposições e proteção e limitação de recursos por usuários e serviços.

2.1.5. Security Misconfiguration

Esta vulnerabilidade aumentou sua posição em comparação com a última edição, um dos motivos é a mudança dos *softwares* permitindo configurações em alto nível. Esta configuração se dá em relação aos softwares com outras aplicações, com servidores, com banco de dados. Ortega (2021) também pontua, que as configurações mal documentadas ou o uso de uma configuração padrão, ou seja, de fábrica, também são portas abertas para esta vulnerabilidade.

Conforme a OWASP (2021), as formas desta vulnerabilidade ser explorada é em casos como: a falta de uma segurança robusta apropriada em qualquer parte da aplicação; permissões configuradas incorretamente em serviços em nuvem; instalação ou ativação de recursos desnecessário para a finalidade da aplicação; uso de contas e senhas padrões; a não atualização dos recursos de segurança da aplicação; não definido os valores seguros para as configurações de servidores, estrutura, bibliotecas, banco de dados da aplicação; mensagem de erros detalhadas para o usuário final.

A OWASP (2021) traz como proposta de prevenção, as seguintes medidas:

- Segregação do ambiente entre desenvolvimento, teste e produção, pois estes ambientes são idênticos, mas necessitam de credenciais diferentes de acesso. Além da necessidade de a construção deste ambiente ser automatizada, para evitar qualquer erro na configuração do ambiente
- Ter uma plataforma de desenvolvimento instalada somente com os recursos necessários para a realização da função da aplicação;
- Implementar a revisão periódica de atualização de patches de segurança da aplicação/plataforma, para evitar ataques e exposições zero-day;
- Criação de uma arquitetura de rede segmentada em contêiner ou grupos de segurança em nuvem;

- Envio de diretrizes de segurança ao usuário/cliente;
- Um processo de teste da eficácia de proteção do ambiente, validando as configurações e ajustes em todo o ambiente.

2.1.6. *Vulnerable and Outdated Components*

Este tópico apareceu em 2017 como uso de componentes com vulnerabilidade conhecida, e fala da dificuldade de manter o *software* sempre atualizado de acordo com as vulnerabilidades que são mapeadas.

> Quando uma vulnerabilidade é reportada e torna-se pública, paths de segurança são disponibilizados pelo fabricante, porém nem todas as corporações irão efetuar a atualização imediatamente, e os atacantes sabem disso, dessa forma vasculham a internet em busca de softwares com vulnerabilidades recém-descobertas, para poder explorar e comprometer os sistemas internos de grandes organizações. (SAMPAIO, 2021, p. 51)

Este é um dos motivos desta categoria ter saído da 9ª posição em 2017 para 6ª posição em 2021, considerando o cenário atual que a pandemia do Covid-19 fez emergir

> [...] a necessidade da adoção de tecnologias digitais em vários segmentos da sociedade, e a quantidade de dados que passaram a trafegar na Internet aumentou consideravelmente no mundo todo, isto por conta das restrições impostas pela necessidade de isolamento social (DUARTE, SANTANA, 2021, p.2)

Pensando em uma empresa, a OWASP (2021) observou que se faz necessário conhecer todos os componentes de *software* que são utilizados, sempre analisando o lado do cliente e do servidor, para detectar se esses estão vulneráveis, desatualizados ou se já não possuem mais suporte pelo fabricante. Monitoramento de publicação de novas vulnerabilidades são fundamentais, assim como a aplicação de correção dos patches de segurança. É preciso realizar testes de compatibilidade com as bibliotecas, sempre que houver atualização ou correção, e verificar a compatibilidade, além de realizar as configurações corretas dos componentes, remover qualquer dependência que tenha da aplicação e que não são utilizadas e realizar o monitoramento de bibliotecas e componentes que não são mantidos ou não criam *patches* de segurança para versões mais antigas e por fim tenha-se um inventário contínuo.

2.1.7. *Identification and Authentication Failures*

Falhas de identificação e autenticação, estas vulnerabilidades não atingem somente aplicações *web*, mas também acesso a redes, sistemas e até locais físicos. A confirmação da identidade do usuário, autenticação e gerenciamento de sessão são fundamentais para proteger os contra-ataques relacionados à autenticação.

Porém, elas podem conter falhas de segurança, tais como: o uso de credenciais fracas, padrão ou conhecidas, ataque de força bruta (ou outros ataques automatizados) *cookies* de sessão fracos, Uuo de armazenamentos de senhas em texto simples, criptografados ou com *hash* fraco, autenticação multifator ausente ou ineficaz,exposição do identificador de sessão na URL ou a reutilização do identificador de sessão após o login bem-sucedido e a não validação correta dos IDs de sessão.

Conforme Sampaio (2021), se um invasor conseguir encontrar falhas em um mecanismo de autenticação, ele obterá acesso às contas de outros usuários. Isso permitiria ao invasor acessar dados confidenciais (dependendo da finalidade do aplicativo). As principais formas de mitigar esta vulnerabilidade é a implementação de multifator de autenticação (MFA), de preferência que utilizam *tokens* ou aplicativos, não use credenciais padrões, principalmente em contas administrativas, tenha uma Política de Senha implementada com as melhores práticas do NIST 800-63b na seção 5.1.1, limitação de tentativa de *login,* uso de um gerenciador de sessão integrado, seguro e do lado do servidor que gera um novo ID de sessão aleatório com alta entropia após o login.

2.1.8. *Software and Data Integrity Failures*

Esta categoria é nova dentro do Top10 da OWASP (2021) e está relacionada com a ausência da camada de proteção contra violação de integridade no código e na infraestrutura, tornando-se uma vulnerabilidade de alto impacto no negócio. E a forma mais comum de exploração desta vulnerabilidade são as atualizações automáticas dos *softwares,* omento em que a empresa desenvolvedora pode colocar um pedaço de *script* de um *malware* sem perceber por usar a biblioteca pública de códigos abertos.

A principal forma de mitigar a exploração desta vulnerabilidade é o cliente ter um controle sobre esta atualização, e antes de aplicá-la no ambiente de produção é

necessário realizar testes em um ambiente apartado, mas com as mesmas configurações no ambiente principal. Também se deve, conforme a OWASP (2021), usar assinaturas digitais ou mecanismos semelhantes para verificar se o *software* ou os dados são de fonte confiável e não foram alterados, para garantir que bibliotecas e dependências estejam integradas a repositórios confiáveis.

2.1.9. *Security Logging and Monitoring Failures*

Este tópico já apareceu no *Top*10 OWASP 2017 e subiu uma posição para o Top10 2021. Esta vulnerabilidade consiste na falha de registro e monitoramento de segurança e vem para auxiliar ao analista detectar, escalar, responder a violação/ incidente que ocorrerem na aplicação, conforme pontua Sampaio (2021):

> O registro é importante pois, no caso de um incidente, as ações dos invasores podem ser rastreadas. Uma vez que suas ações são rastreadas, seu risco e impacto podem ser determinados. Sem o registro, não haveria como saber quais ações um invasor executou e se obteve acesso à determinados sistemas (SAMPAIO, 2021, p. 55)

Portanto, todas as ações realizadas pelo usuário devem ser registradas por meio de eventos pela aplicação. Mitigando o impacto regulatório, pensando além da Lei Geral de Proteção de Dados, como também a dificuldade de prevenir para novos ataques, em concordância com Ortega (2021), *a categoria foi expandida para incluir mais tipos de falhas, incluindo falhas que podem afetar diretamente a visibilidade, o alerta de incidentes e a perícia*. Para mitigar o risco desta vulnerabilidade ser explorada, o desenvolvedor deve ponderar os seguintes controles (OWASP 2021):

- Garanta que todas as falhas de login, controle de acesso e validação de entrada, possam ser registradas com contexto de usuário suficiente para identificar contas suspeitas ou maliciosas;
- Manter os registros por tempo suficiente para permitir análises forenses futuras;
- Certifique-se de que os logs sejam gerados em um formato que as soluções de gerenciamento de log possam consumir facilmente;
- Certifique-se de que os dados de log sejam codificados corretamente para evitar injeções ou ataques nos sistemas ou monitoramento e tenha problemas com a integridade da informação do log;

- Garanta que as transações de alto valor tenham uma trilha de auditoria com controles de integridade, para evitar adulteração ou exclusão;
- As equipes de DevSecOps devem estabelecer monitoramento e alertas eficazes para que atividades suspeitas sejam detectadas e respondidas rapidamente;
- Estabeleça ou adote um plano de resposta e recuperação de incidentes, como o Instituto Nacional de Padrões e Tecnologia (NIST) 800-61r2 ou posterior.

Vale ressaltar, que a trilha de auditoria ou o registro de *log* devem conter as seguintes informações: código de status *HTTP*(s); *timestamps*, nome de usuário, e*ndpoint* utilizado, *API*/diretório, Endereços *IP*.

2.1.10. *Server-Side Request Forgery* (SSRF)

De acordo com a OWASP (2021), as falhas do SSRF ou falsificação de solicitação do lado do servidor, ocorrem quando um aplicativo da *Web* está buscando um recurso remoto sem validar a URL (*Uniform Resource Locator*) fornecida pelo usuário. Ele permite que um invasor force o aplicativo a enviar uma solicitação criada para um destino inesperado, mesmo quando protegido por um *firewall*, VPN (*Virtual Private Network*) ou outro tipo de lista de controle de acesso à rede (ACL), portanto

> [...] um ataque SSRF bem-sucedido pode muitas vezes resultar em ações não autorizadas, acesso a dados dentro da organização ou quaisquer ataques mal-intencionados que levariam os analistas a entender que o problema se origina da organização que hospeda o aplicativo vulnerável (ORTEGA, 2021, p. 115).

A forma de prevenir contra a exploração desta vulnerabilidade, é implementar um controle de segurança em profundidade na camada de rede e aplicação. Camada de rede: segmentação da rede, controle do acesso remoto, aplicação de política de *firewall* ou de acesso a rede para bloquear todo o tráfego e liberar somente o essencial; implementar um ciclo de vida das regras de *firewall*; documentar e registrar todo o fluxo de dados da rede se foram aceitos ou bloqueados pelo *firewall*. Camada de aplicação: valida e sanitiza todos os dados fornecidos pelo usuário; aplica na *URL* uma porta e o destino da aplicação para uma lista de permissões aceitas; desativa redirecionamento *HTTP* e tem uma consistência de uso da *URL*. Mas vale lembrar

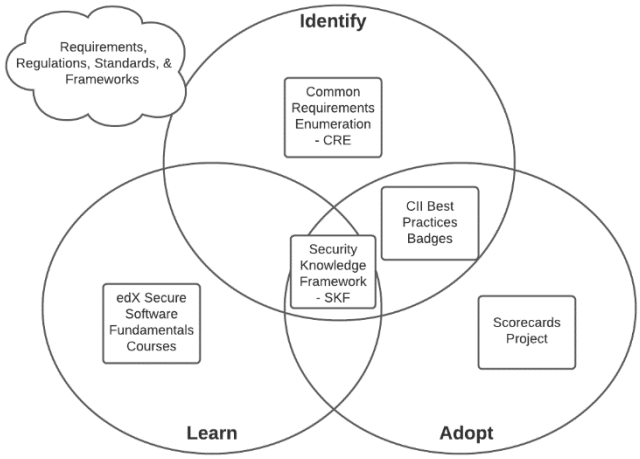
sempre que os invasores, possuem listas, ferramentas e habilidades para contornar a *blacklist*[18].

2.2. *OPENSSF*

A *Open Source Security Foundation* ou *OpenSSF* é uma comunidade que reúne lideranças para melhorar a segurança do *software* de código aberto (OSS[19]). Esta organização tem como objetivo primário auxiliar com métricas, ferramentas, validação de identidade do desenvolvedor e melhores práticas de divulgação de vulnerabilidades[20].

Todas as informações relacionadas a *OpenSSF* estão no *blog* oficial[21] e no *GitHub*[22]. Consultando os tópicos do *GitHub*, a visão da comunidade é baseada em identificar, aprender e adotar, conforme a figura 02.

Figura 02 – Diagrama de desenvolvimento seguro pela *OpenSSF*



Fonte: (OpenSSF, 2022, *online)*

Baseado também na OpenSSF foi desenvolvido o "*[RASCUNHO] Guia de uma página para o* desenvolvimento de *software* seguro <*Data*>[23]", que aborda tópicos

[18] Termo em inglês para uma lista com as URL bloqueadas.
[19] Sigla em inglês para *Open Source Security*
[20] Descrição conforme no faq ou "perguntas frequentes". Disponível em: https://openssf.org/about/faq/
[21] Disponível em: https://openssf.org/blog/
[22] Disponível em: https://github.com/ossf
[23] Original em inglês, *[DRAFT] One-page Guide for Developing Secure Software <Date>.* Disponível em: https://docs.google.com/document/d/16jUqTEFG-wscZUGR-NGa_3a81GF3YILtH9XgOSkLCTM/edit#heading=h.ffloaujgy1u0

básicos para o desenvolvimento seguro em bibliotecas de código aberto. Dentre os 17 tópicos selecionamos alguns sob a análise da OWASP, *Microsoft SDL* e os princípios de Segurança da Informação:

> Identificar boas práticas, requisitos e ferramentas que ajudam os desenvolvedores de código aberto a criar e manter um software mais seguro. Ajudando os desenvolvedores aprenda a escrever software seguro. Fornece ferramentas para ajudar os desenvolvedores a adotar essas boas práticas em seu trabalho diário24 (OpenSSF, 2022, online)

2.2.1. Aprenda a desenvolver *softwares* seguro

A OpenSSF considera a primeira como parte mais importante do desenvolvimento seguro, aprender sobre as ferramentas utilizadas, aprender sobre os princípios da segurança da informação. E para isto eles montaram um curso especializado para o desenvolvimento seguro em aplicações de código aberto. O que se assemelha com o Microsoft SDL em que o primeiro ciclo do desenvolvimento é o treinamento e capacitação do desenvolvedor. Este curso é divido em 3 partes e pontua como conteúdo importante para o desenvolvimento seguro os seguintes módulos: parte 01 - requisitos, d*esign* e reutilização, noções básicas de segurança, princípios de *design* seguro e reutilização de s*oftware* externo; parte 02 - implementação, validação de entrada, processamento de dados com segurança chamando outros programas, envio de saída e parte 03 - verificação de tópicos mais especializados, modelagem de ameaças e criptografia.

Um detalhe interessante é que ele aponta neste treinamento e em outros tópicos dentro do *GitHub,* a consulta de outras fontes sobre as principais vulnerabilidades, como o OWASP Top10 e o CWE Top 25.

2.2.2. Privilégio mínimo

Conforme o princípio de *design* seguro, a *OpenSSF* pontua em seu curso[25] a importância de ter uma gestão de acesso e principalmente de privilégio de acesso. Portanto, cada usuário e programador deve acessar o sistema usando o menor

[24] Original em inglês:
[25] Desenvolvendo *Software* Seguro (LFD121) disponível em: https://training.linuxfoundation.org/training/developing-secure-*software*-lfd121/

número de privilégios possível. Este princípio mitiga os danos de um incidente, seja ele um erro ou ataque de um *hacker*; reduz o número de interações potenciais entre programas privilegiados, com isto diminuindo a ocorrência de usos não intencionais, indesejados ou impróprios de acessos com privilégios de administrador.

É necessário se certificar de que todos os desenvolvedores, principalmente aqueles com acesso privilegiados utilizam *tokens* e autenticação multifatorial "*Multi-factoration* (MFA)". Assim, o acesso à aplicação deve depender de mais de uma condição, exemplo, o uso de uma senha e aprovação de um aplicativo de autenticação. Dessa forma, se um invasor conseguir quebrar uma condição, que é mais comum com senha, o sistema ainda permanecerá seguro. Vale ressaltar que os programas podem ser divididos em partes e cada parte possui um tipo de privilégio diferente. Essa abordagem às vezes é chamada de "segregação de privilégios", baseando em função do desenvolver e usuário. E este fator está de acordo com o *Top*1 da lista do *Top*10 OWASP 2021, ressaltando a importância das medidas de autenticação de acesso do usuário à aplicação.

2.2.3. Dependência com outros *softwares*

Tudo o que o objeto precisa para funcionar é chamado de dependência[26]. Cada dependência cria um caminho para vulnerabilidades (não intencionais e intencionais), especialmente porque muitas dependências trazem outras dependências transitivas.

Deve ser avaliado o *software* antes de seleciondo como uma dependência direta da aplicação que está sendo desenvolvida. Pois em vez de ter uma solução, poderá abrir uma porta para uma vulnerabilidade, se a arquitetura da aplicação conectada não for segura. Ao instalar uma nova dependência a sua aplicação deve verificar o nome do pacote para evitar erros de digitação e realizar a comunicação correta. Se você usar mais de um repositório, é necessário certificar de que é usado o repositório correto e não um malicioso.

[26] Disponível em: https://www.devmedia.com.br/abstracoes-e-dependencias-principios-de-orientacao-a-objetos/19142

Para isso a *OpenSSF* construiu o "*Quick Guide para avaliar o software de código aberto*[27]", um questionário para avaliar manualmente a dependência de *software*.

Tabela 02 - avaliação manual de *software* de código aberto

Questão	Tradução	Descrição
Should it be added at all?	Deve ser adicionado?	Esta questão deve ser feita, pois dependendo do escopo e a forma que desenvolve o projeto, não precisa de uma dependência, e colocar sem usabilidade poderá abrir uma porta a vulnerabilidade.
Review the last commit	Revise a última alteração do código	A revisão do código da última alteração ajuda a garantir uma maior cobertura contra falhas ou má qualidade do *software*
Is it easy to use securely?	É fácil de usar com segurança?	Quando algo é difícil de usar com segurança, o resultado provável que seja inseguro.
Is there evidence that its developers work to make it secure?	Há evidências de que os desenvolvedores atuam para tornar a aplicação segura?	O guia aponta que uma evidência é de a empresa ter o selo[28] *OpenSSFF*; como também a documentação do projeto, onde esteja mapeado cada etapa de segurança.
Is there evidence that the developers use tools to detect defects and vulnerabilities as early as possible?	Há evidências de que os desenvolvedores usam ferramentas para detectar defeitos e vulnerabilidades o mais cedo possível?	Os projetos devem ter uma linha de produção de integração contínua, que possa verificar as alterações no código, incluindo *scan* de vulnerabilidades.
Is there documentation explaining why its developers believe it is secure (aka an "assurance case")?	Há uma documentação explicando por que os desenvolvedores acreditam que é seguro ("caso de garantia")?	Esta aplicação que criará a dependência possuí uma documentação informando o porquê ela é segura? Seria fazer uma gestão de fornecedor na dependência (ainda mais se for com APIs de terceiro);
Is there evidence of a security audit, and that any problems found were fixed?	Há evidências de uma auditoria de segurança, e se detectado algum problema, foi corrigido?	A maior importância ao analisar uma auditoria de segurança da informação da "dependência" é se as vulnerabilidades detectadas foram corrigidas.
Are there instructions on how to report vulnerabilities?	Existem instruções sobre como relatar vulnerabilidades?	Ter uma documentação clara e objetiva de como relatar uma vulnerabilidade permite que haja uma gestão de fornecedor com a aplicação.

[27] Disponível em inglês: https://docs.google.com/document/d/19015o63pA-sZPBN5-R7I7Cg8ICPxo6fXH8vOI1RgBYM/edit

[28] Best Practices Badge Program, conforme a OpenSSF, é uma forma de projetos de *software* livre/livre e de código aberto (FLOSS) mostrarem que seguem as práticas recomendadas.

Is it maintained?	Este *software* tem atualizações?	Em uma sociedade da informação com novas tecnologias sendo publicadas diariamente, atacantes possuem novas formas de explorar uma vulnerabilidade. Caso a aplicação não esteja atualizada, poderá trazer esta vulnerabilidade em todo o *software*.
Does it have significant use?	Tem uso significativo [no mercado]?	Um *software* com clientes de multinacional pode indicar maior credibilidade em relação a segurança, porém deve-se questionar, é a melhor solução para o projeto? Como também um *software* com poucos usuários pode representar um ataque de engenharia social
What is the software license?	Qual é a licença do *software*?	Mesmo não sendo uma questão de Segurança, ela traz impactos regulatórios para a empresa.
If it is important, what is your own evaluation of it?	Se é importante, qual é a sua própria avaliação?	Pensando no desenvolvedor que está analisando ferramentas, ele precisa ter uma opinião própria sobre esta solução.
such as rigorous input validation of untrusted input and the use of prepared statements?	Há uma rigorosa validação de entrada para entrada não confiável e o uso de declarações preparadas?	A questão de credenciais de acesso é de suma importância, e deve ser analisada ao código se não há nenhuma vulnerabilidade para acesso indevido ao *software*.
Is there evidence of insecure or woefully incomplete software?	Existem evidências do código estar inseguro ou incompleto?	Deverá analisar o risco de usar um *software* inseguro ou incompleto. E por fim se aceitará este risco ou não.
Is there evidence that the software is malicious?	Há evidências de que o *software* é malicioso?	Observar características comuns de uma aplicação maliciosa; verifique as rotinas de instalação; verifique se há extração e envio de dados como ~/. variáveis *ssh* ou ambiente); procure valores codificados que acabam sendo executados

Fonte: (OpenSSF, 2022, *online)*

O Guia também aponta que sempre que realizar esta análise e teste do *software,* que criará a dependência com a aplicação, devem ser realizados em um

ambiente seguro como um *SandBox*[29], para mitigar o risco de uma contaminação no ambiente de produção e poder detectar vulnerabilidades antes de aplicar a correção. Todavia, um *software* malicioso e bem programado consegue detectar que ele está sendo analisado e mudar o seu comportamento como uma aplicação segura.

Outra preocupação em relação as dependências de *software* são os pacotes de atualizações, que precisam levar em consideração o tempo e a segurança desta função. Para isto deverá haver gerenciadores de pacotes e testes automatizados, incluindo os testes negativos, que é para testar o que não deve acontecer no pior cenário possível.

2.2.4. Comunicar e detectar uma vulnerabilidade

Para um programa efetivo de resposta incidente é necessário que os usuários saibam detectar uma vulnerabilidade, comunicar sua existência e o desenvolvedor gerar o relatório desta vulnerabilidade. Seguindo as orientações da resposta a incidente, é importante comunicar a vulnerabilidade detectada com base na ISO 27002.

> Deve ser requerido aos funcionários e usuários que reportem o mais rápido possível todos os incidentes e deficiência à central de atendimento ou a uma pessoa de contato. Naturalmente, é do interesse de todos que a organização responda rapidamente (Hintzbergen et al, 2018 p. 202).

Para auxiliar os desenvolvedores de *o* a comunicar e relatar as vulnerabilidades, a OpenSSF formulou um *guia para divulgação coordenada de vulnerabilidades para projetos de software de código aberto*[30] e outro *guia para implementar um processo coordenado de divulgação de vulnerabilidades para projetos de código aberto*[31].

> Como entrar em contato com a equipe sobre uma possível vulnerabilidade de segurança; O relatório de vulnerabilidade pode ser mantido privado até que o projeto decida compartilhar mais amplamente; As expectativas do repórter sobre comunicação/colaboração em torno do assunto estão devidamente definidas (OpenSSF, 2022, online)

[29] Caixa de Areia em inglês, um ambiente considerado seguro para acesso e realização de teste com arquivo/*software*s desconhecidos; e evitar uma contaminação no ambiente principal.
[30] Disponível em https://github.com/ossf/oss-vulnerability-guide
[31] Disponível em: https://github.com/ossf/oss-vulnerability-guide/blob/main/guide.md

Os processos e ferramentas não precisam ser burocráticos, complexos e onerosos, mas a documentação tem como objetivo garantir que o relator[32] possa obter as informações necessárias do *bug* para auxiliar os desenvolvedores responsáveis e corrigir a vulnerabilidade.

As ferramentas de *scan* de vulnerabilidade do código, não são os únicos mecanismos, porém, auxilia o processo do código antes de ir para produção, pois as vulnerabilidades detectadas poderão ser corrigidas. Vale ressaltar que nenhuma ferramenta é perfeita, elas podem detectar falsos positivos e/ou falsos negativos. Como a *OpenSSF* pontua, mesmo que com estes possíveis erros, é melhor corrigir as vulnerabilidades e análise de risco com base nos relatórios[33]. Pois as ferramentas fazem parte do processo de desenvolvimento seguro e devem ser utilizadas a partir da fase de *design*/arquitetura.

Existem dois tipos principais de ferramentas, as de análise dinâmica e análise estática, que serão detalhadas melhor no próximo capítulo. Mas segundo a *OpenSSF* convém configurar as ferramentas para delimitar o relatório somente com as informações necessárias e a detecção de vulnerabilidade. É indicado implementar desde o início do projeto, pois terão um *feedback* constante de vulnerabilidade em todas as fases.

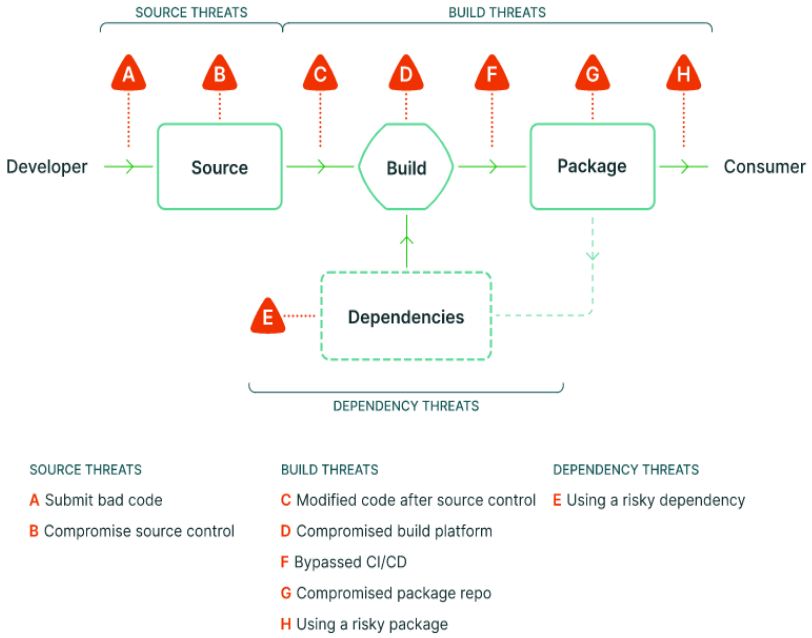
Outra forma de detectar vulnerabilidades é monitorar as já conhecidas e acompanhar os lançamentos de novas de acordo com a publicação da CVE *(Common Vulnerabilities and Exposures)*, e com os lançamentos do projeto.

2.2.5. *Supply chain levels of software artifacts* (slsa)

*Supply chain Levels of Software Artifacts (*SLSA) trata-se de níveis da cadeia de suprimentos para artefatos de *software*[34], de acordo com a slsa.dev[35], é pensar em uma estrutura de segurança com lista de verificação de padrões e controles para garantir a integridade, protegendo pacotes e infraestrutura dos projetos, ser resiliente em qualquer elo da cadeia, endurecendo a integridade do seu processo de construção e distribuição contra ataques.

[32] Pessoa Responsável pelo Relatório de Vulnerabilidade
[33] No português “Guia de ferramentas segura”, disponível em: https://github.com/ossf/wg-security-tooling/blob/main/guide.md
[34] Traduzido do inglês:Supply chain Levels for *Software* Artifacts
[35] Disponível em https://slsa.dev/

Figura 03 – Cadeia de suprimento para produção de *software* e pontos de ameaça.



Fonte: (SLSA, 2021, *online*)

A figura 03, representa a cadeia de suprimento de *software*, que é uma integração interna de atividades até a concepção do produto. Para a segurança desta cadeia de suprimento, é divido em 3 grandes grupos

- *source threats:* De acordo com a SLSA, são 2 as principais ameaças de código, que é o envio de um código fonte incorreto (A) para o ambiente de desenvolvimento; e o comprometimento do controle do código fonte (B) durante o desenvolvimento da aplicação;
- *build threats:* Ameaças que atinge a arquitetura do software após a implementação do código fonte. As principais são a modificação do código após controle de origem (C); comprometimento na estrutura da plataforma (D); ignorar CI/CD (F); atualização de um pacote que está comprometido (G); utilização de um pacote arriscado (H);
- *dependency threats:* O uso de dependências arriscadas é a maior ameaça desta categoria e foi discutida no capítulo 2.5 Dependências com outros softwares.

Os ataques podem ocorrer em qualquer elo de uma *suply chain* de *software*, o que representando um grande obstáculo para qualquer responsável envolvido dentro dos sistemas críticos de uma empresa. As SLSAs foram projetadas para garantir que

sejam de conhecimento comum, o que torna mais fácil a proteção. O slsa.dev[36] nos traz uma tabela referenciando um exemplo de exploração da vulnerabilidade e de como corrigir.

Tabela 03 – exemplo de ataque e mitigação na Suply Chain[37]

D	Ameaça	Exemplo conhecido	SLSA
	Enviar código incorreto para o repositório de origem	Compromissos hipócritas do Linux: O pesquisador tentou introduzir intencionalmente vulnerabilidades no kernel do Linux por meio de patches na lista de discussão.	revisão de duas pessoas pegou a maioria, mas não todas, as vulnerabilidades.
	Plataforma de controle de origem comprometida	PHP: O invasor comprometeu o servidor git auto hospedado do PHP e injetou dois *commits* maliciosos.	Uma plataforma de código fonte mais protegida teria sido um alvo muito mais difícil para os invasores.
	Compile com o processo oficial, mas a partir do código que não corresponde ao controle de origem	*Webmin*: O invasor modificou a infraestrutura de compilação para usar arquivos de origem que não correspondam ao controle de origem.	Um servidor de compilação compatível com SLSA teria produzido proveniência identificando as fontes reais usadas, permitindo que os consumidores detectassem tal adulteração.
	Plataforma de construção de compromisso	*SolarWinds*: O invasor comprometeu a plataforma de compilação e instalou um implante que injetou comportamento malicioso durante cada compilação.	Níveis mais altos de SLSA exigem controles de segurança mais fortes para a plataforma de compilação, tornando mais difícil comprometer e obter persistência.
	Use dependência arriscada (ou seja, AH, recursivamente)	*event-stream*: o invasor adicionou uma dependência inócua e depois atualizou a dependência para adicionar comportamento malicioso. A atualização não correspondeu ao	Aplicar o SLSA recursivamente a todas as dependências impediria esse vetor específico, porque a proveniência indicaria que ele não foi

[36] Disponível em: https://slsa.dev/spec/v0.1/threats
[37] Tradução livre

		código enviado ao *GitHub* (ou seja, ataque F).	construído a partir de um construtor adequado ou que a fonte não veio do *GitHub*.
	Carregar um artefato que não foi criado pelo sistema CI/CD	*CodeCov*: o invasor usou credenciais vazadas para fazer *upload* de um artefato malicioso para um *bucket* do GCS, do qual os usuários baixam diretamente.	A proveniência do artefato no *bucket* do GCS teria mostrado que o artefato não foi criado da maneira esperada a partir do repositório de origem esperado
	Repositório de pacotes comprometido	Ataques a espelhos de pacotes: O pesquisador executou espelhos para vários repositórios de pacotes populares, que poderiam ter sido usados para servir pacotes maliciosos.	Semelhante ao acima (F), a proveniência dos artefatos maliciosos teria mostrado que eles não foram construídos conforme o esperado ou a partir do repositório de origem esperado.
	Induzir o consumidor a usar um pacote ruim	*Browserify typosquatting*: O invasor carregou um pacote malicioso com um nome semelhante ao original.	O SLSA não aborda diretamente essa ameaça, mas a origem vinculada ao controle de origem pode habilitar e aprimorar outras soluções.

Fonte: (SLSA, 2021, *online*)

A aplicação do SLSA, permite a rastreabilidade do código com segurança de sua origem até o final. Basicamente protege o *software* desde a origem por meio de sua implantação, permitindo que os usuários tomem decisões automatizadas sobre a integridade dos artefatos que estão usando, evitando ataques possíveis por toda a cadeia de suprimentos.

Com isso fica claro que desde o início do projeto, se faz necessário trazer testes de segurança e entender como essas ferramentas específicas de testes atuam em cada etapa do desenvolvimento.

3. PROCESSOS *AST (APPLICATION SECURITY TESTING)*

Application Security Testing (AST) ou Teste de Segurança de Aplicativos é um conjunto de processos capazes de tornar *softwares* mais seguros às ameaças. Ele faz isso por meio da identificação de falhas de segurança e possíveis fragilidades no código fonte. É fato que todas as fases de elaboração, produção e implementação de um novo programa ou sistema demandam um delicado alinhamento entre as partes e os processos AST são os recursos-chave para garantir que o produto estará livre de *bugs*. O papel principal dos AST é detectar onde está o erro dentro do código, caso haja algum; o que é extremamente valioso nos dias de hoje, uma vez que a produção de *softwares* se torna a cada dia mais ágil, e a demanda, mais exigente em termos de eficácia e desempenho.

Ter uma estratégia competente de AST é o caminho mais curto para evitar gastos desnecessários com análise e refatoração de códigos, assim como previne danos à reputação das empresas de segurança da informação e de desenvolvimento de *softwares*. Os times que contam com esse tipo de método estão muito mais predispostos a receberem avaliações positivas por usuários e clientes do que aqueles que, por alguma razão, negligenciam essa etapa do planejamento de uma nova aplicação.

As tecnologias usadas nos processos AST são delineadas com o objetivo de testar, analisar e emitir relatórios sobre o nível de segurança de um aplicativo ao longo do ciclo de vida de seu desenvolvimento. Caso seja identificado um problema, ele é eliminado rapidamente; o resultado disso é um código fonte mais forte e que mantém os aplicativos muito mais seguros diante de ameaças internas e externas.

Bugs são comuns em *softwares*, tão comuns que eles são a principal motivação para o uso de ferramentas AST. A cada dia surgem novas opções delas, o que pode ser confuso para líderes, desenvolvedores e engenheiros de tecnologia da informação (TI). Para que a escolha seja adequada, é indicado que os diferentes tipos de ferramentas AST disponíveis sejam considerados, assim como suas especificidades e vantagens. Compreender cada contexto também é importante para ter resultados que aliem segurança e desempenho.

A segurança de aplicativos não é uma questão simples, como se bastasse um único processo para garantir máxima proteção. Trata-se muito mais de um compilado de camadas de segurança, uma acoplada à outra, com todas elas trabalhando sincronizadamente para reduzir o risco de falhas a um nível bem baixo. Isso é o que as organizações esperam: um *software* apto a ser o mais perfeito possível, ainda que não seja viável eliminar toda a chance de erro.

A principal motivação para usar as ferramentas AST é que o trabalho manual com os códigos e fases de testes é muito demorado e caro, além do fato de que depender da detecção manual de um erro é pouco confiável. Se pensarmos em um cenário de empresas rivais, por exemplo, em que uma delas usa o esquema tradicional de detecção manual e outra que possui processos AST, esta última terá a vantagem explícita por conta da maior eficiência e estabilidade diante de qualquer vulnerabilidade: falhas serão reparadas em bem menos tempo, com o envolvimento de menos pessoas e isso propiciará em uma entrega superior ao cliente.

As ferramentas AST podem ser configuradas para atuar em escala: uma vez que um teste seja desenvolvido, ele pode ser executado em muitas linhas de código sem que isso gere um aumento exponencial do custo. É possível fazer a triagem e classificação dos *bugs* encontrados, além de identificar padrões que poderiam se manifestar no futuro.

As ferramentas AST podem ser estáticas, dinâmicas ou interativas, assim como podem ser manuais, automatizadas ou uma combinação de ambos.

3.1. *Dynamic Application Security Testing* ou Teste de Segurança de Aplicativo Dinâmico (DAST)

As ferramentas de *Software* Dinâmico de Teste de Segurança de Aplicativos (DAST) geram o *fuzzing*: criam milhares de solicitações e bombardeiam o *software* com elas. Também conhecido como “teste de caixa preta”, porque quem testa não tem acesso ao código-fonte nem a maiores informações sobre o projeto, a ferramenta DAST testa as interfaces expostas de um aplicativo em busca de vulnerabilidades de fora para dentro.

A DAST é competente em encontrar vulnerabilidades visíveis, mas depende muito de especialistas para preparar os testes, o que dificulta o seu uso em grande

escala. Ela pode encontrar problemas que aparecem em páginas da *Web*, públicas, mas não consegue ver nada que esteja no código-fonte. As ferramentas DAST são executadas para detectar problemas com interfaces, solicitações, respostas, *scripts* (ou seja, j*avascript*), injeções de dados, sessões, autenticações e muito mais.

3.2. *Static Application Security Testing* ou Teste de Segurança de Aplicativo Estático (SAST)

SAST são ferramentas estáticas que tentam modelar o aplicativo como um todo, “adivinhando” como o código fonte, as bibliotecas, as estruturas e os componentes se encaixam e como funcionarão quando executados. Isso envolve analisar o código fonte procurando por condições de codificação e *design* que possuam vulnerabilidades.

As soluções SAST analisam um aplicativo de dentro para fora, quando ele não está em execução, tentando avaliar sua força de segurança. São ferramentas chamadas de “teste de caixa branca”, porque quem testa conhece informações sobre o sistema ou *software* que está sendo analisado, incluindo o diagrama de sua arquitetura, tem acesso ao código fonte etc.

Essas ferramentas são capazes de verificar defeitos como erros numéricos, validação de entrada, percursos, ponteiros e muito mais.

3.3. *Interactive Application Security Testing* ou Teste de Segurança de Aplicativo Interativo (IAST)

As IAST são abordagens híbridas, ou seja, elas misturam as técnicas dos exemplos anteriores. Estão disponíveis há muito tempo, mas recentemente foram categorizadas e receberam o termo IAST como denominação oficial. Sua técnica é combinar análise estática e dinâmica, testando se as vulnerabilidades conhecidas no código são ameaças em potencial para o aplicativo já em execução.

As ferramentas IAST usam o conhecimento do fluxo de aplicativos e do fluxo de dados para criar cenários avançados de ataque. À medida que uma verificação dinâmica está sendo executada, a ferramenta “aprende” coisas sobre o aplicativo com base em como ele responde.

Algumas dessas ferramentas usarão esse conhecimento para criar testes adicionais; isso fará com que elas sejam hábeis em reduzir o número de falsos positivos. Funcionam bem em ambientes *Agile* e *DevOps*, nos quais as ferramentas DAST e SAST tradicionais poderiam performar consumindo tempo demais durante o desenvolvimento.

3.4. DAST, SAST ou IAST

As vulnerabilidades das aplicações são a principal dor de cabeça para as organizações de TI e as abordagens tradicionais para o problema são muito lentas e propensas a erros diante de processos modernos de desenvolvimento de *software* de alta velocidade, como *Agile* e *DevOps*. Então, o que fazer?

Das opções pontuadas até aqui, somente uma é capaz de eliminar os falsos positivos, reduzindo drasticamente o gasto de tempo e recursos com testagem e refação. Trata-se das ferramentas IAST, que combinam processos dinâmicos e estáticos.

Recorrer às IAST é uma forma segura de analisar, de maneira totalmente automatizada, as vulnerabilidades desde o princípio, “lendo” informações do próprio aplicativo. Elas consideram o que as SAST e as DAST têm de melhor e usam essas estratégias para identificar erros com precisão, com a possibilidade de trabalhar em larga escala para proteger um *software* de ataques de todos os tipos.

Com a vantagem do escalonamento, as ferramentas IAST podem verificar uma grande quantidade de códigos-fonte, entregando relatórios com resultados precisos e projetando cenários de risco à medida que são utilizadas. Isso as torna ideais para ambientes *Agile*, *DevOps* e *DevSecOps*, pois permitem que o time de TI encontre e corrija falhas de segurança logo no início do desenvolvimento, tornando o processo mais fácil e barato.

Caso não haja a possibilidade imediata de recorrer a uma IAST, é preciso partir para uma técnica provisória inicial. Se o aplicativo for escrito internamente ou existir acesso ao código-fonte, um bom ponto de partida é executar uma ferramenta SAST e verificar problemas de codificação e de aderência aos padrões de codificação. As SAST são o ponto de partida mais comum para a análise inicial de um código. Se não existir acesso ao código-fonte, uma ferramenta DAST é a melhor escolha.

É importante notar que nenhuma ferramenta sozinha resolverá todos os problemas. Como dito, o objetivo é reduzir o risco e a exposição, e a melhor forma de fazer isso é combinando camadas de proteção.

As AST são usadas para corrigir vulnerabilidades enquanto os aplicativos ainda estão em desenvolvimento. A longo prazo, a incorporação dessas ferramentas economiza tempo e esforço no retrabalho, pois detecta problemas mais cedo.

Na prática, no entanto, a implementação de ferramentas AST requer algum investimento inicial de tempo e recursos antes que surjam resultados exponenciais. É preciso que líderes e equipes tenham isso em mente e que vejam as AST como um investimento de tempo e esforço em algo que futuramente os manterá a salvo de ataques que, de outra maneira, representariam prejuízos severos à organização e às suas carreiras. O objetivo das ferramentas AST é a otimização da performance tanto dos aplicativos quanto dos recursos humanos que trabalham em contextos de TI.

Em suma, as ferramentas DAST são um “teste de caixa preta” que procura vulnerabilidades de segurança e pontos fracos na arquitetura simulando ataques externos em um aplicativo enquanto ele está em execução. Como tal, elas não têm acesso ao código-fonte e são capazes de descobrir falhas apenas por meio de ataques externos.

As ferramentas SAST são um “teste de caixa branca” em que o código-fonte é analisado de dentro para fora enquanto os componentes ainda não estão “rodando”. Elas analisam o código-fonte do aplicativo em busca de falhas de codificação que sugiram possíveis vulnerabilidades de segurança.

As ferramentas IAST verificam o código-fonte de um aplicativo em um ambiente dinâmico, ou seja, híbrido entre estático e dinâmico. O teste ocorre em tempo real enquanto o aplicativo está “rodando”, geralmente em um ambiente de teste ou controle de qualidade. Quando uma IAST está analisando o código-fonte, ela é capaz de identificar a linha de código problemática e notificar o desenvolvedor para uma correção imediata.

Embora tanto as SAST quanto as IAST acessem diretamente os códigos-fonte, as ferramentas IAST o fazem após a compilação em um ambiente dinâmico por meio da instrumentação do código. Agentes e sensores são implantados no aplicativo, analisando o código para identificar vulnerabilidades. As IAST são altamente escaláveis e podem ser automatizadas ou executadas por um ser humano.

A verdade é que, no cenário atual, nenhuma ferramenta pode fazer tudo de uma vez. As empresas precisam de várias ferramentas para proteger seus aplicativos e sistemas e minimizar seus riscos, pois, caso uma falhe, outra atuará em seu lugar. A eficácia do conjunto de técnicas de segurança será consequência direta de uma excelente análise de riscos e da implantação de sucessivas barreiras de proteção, que, “empilhadas”, assegurem a integridade da arquitetura dos dados.

Com um bom diagnóstico do cenário em que o projeto deve rodar e uma combinação dessas ferramentas de segurança, será plausível salvaguardar *softwares* com a máxima precisão. Entretanto, é preciso estar consciente de que não existe uma solução que impeça absolutamente todos os ataques, pois as ameaças virtuais estão em constante aprimoramento.

Da mesma forma, as ferramentas AST devem seguir evoluindo, detectando padrões de intrusos e projetando, com a inferência dos dados disponíveis, como essas ameaças podem vir a se comportar.

A importância de testes de segurança em *softwares*, seguindo o uso das ferramentas AST em conjunto ou de forma unitária, de acordo com o contexto de sua aplicação, trará maior cobertura de segurança e qualidade do *software* e ao negócio, pois falhas podem ocasionar um prejuízo de grandes proporções à uma companhia. Entendendo toda a questão da criticidade que os testes de segurança no desenvolvimento devem ter, muitas dessas ferramentas trazem métricas, o que facilita e direciona a escolha por qual usar em um ambiente de desenvolvimento.

4. *BENCHMARK SAST - FORTIFY* E *SONAR*

No processo de desenvolvimento de uma aplicação, deve existir a necessidade de avaliar a qualidade do código do *software* com métricas, de uma maneira dividida ou particionada em menor complexidade, de forma que uma pessoa consiga ler e interpretar. Além de sinalizar o código que precisa ser refatorado ou quais têm mais probabilidade de possuírem *bugs* (SIMON; STEINBRUCKNER; LEWERENTZ, 2001).

Tipicamente, utilizam-se métricas de qualidade de *software* em desenvolvimentos mais complexos ou que sejam requerido tal processo, pois ainda é pouco difundido no desenvolvimento de *software* em geral.

As métricas podem ser extraídas através de ferramentas que possuam essa funcionalidade. Existem diversas ferramentas para tal uso, no entanto, não há uma ferramenta que seja dada como a melhor - as ferramentas possuem pontos fortes e fracos em validações diversas. Há também a questão do suporte às linguagens de programação e qual métrica é calculada. Assim a ferramenta a ser usada dependerá de qual linguagem é usada para desenvolvimento e quais métricas deseja obter.

As ferramentas apresentadas foram selecionadas a partir de uso em aplicações reais, de forma que abrange funções importantes do código.

4.1. *FORTIFY*

Fortify é uma ferramenta de análise estática de vulnerabilidades em código (SAST). Pode-se acessar o console da ferramenta através do DNS que é criado e é indicado o uso com VPN.

Como funciona a análise:

- Tradução: O código fonte a ser analisado é traduzido para uma linguagem intermediária utilizada pelo Fortify. Essa etapa é executada localmente por um cliente da solução. Neste estágio é necessário informar alguns parâmetros específicos da aplicação como: linguagem

de codificação, versão da linguagem, nome da aplicação, versão da aplicação, localização dos códigos fonte etc.;

- Envio: A tradução é enviada para que a varredura e análise sejam realizadas em um servidor do Fortify. Por esse motivo, são necessárias algumas informações sobre o servidor da solução nesta fase;
- Varredura e Análise: O Fortify realiza a varredura e análise da aplicação que pode ser acompanhada pela console web.

Há a limitação de linguagens de programação suportadas pela ferramenta: ABAP/BSP, *ActionScript*/MXML (Flex). ASP.NET, VB.NET, C# (.NET), C/C++, *Classic* ASP (w/VBScript), COBOL, *ColdFusion* CFML, Go, HTML, Java (incluindo *Android*), *JavaScript*/AJAX, JSP, Kotlin, *Objective*-C, PHP, PL/SQL, Python, T-SQL, *Ruby, Swift, Visual Basic, VBScrip*t, XML.

Realizando análises com integração contínua com alguma ferramenta específica de versionamento de código, trazendo a segurança cada vez mais para a esquerda (estratégia *shift-left*) no ciclo de desenvolvimento, a análise estática de código deve ser contínua. Todos os projetos deverão ser configurados em sua ferramenta de *CI (continuous integration)*.

As análises têm como objetivo trazer insumos ao time de engenharia e segurança da informação, no que diz respeito a potenciais vulnerabilidades e/ou utilização de práticas inadequadas de código seguro.

Deve-se construir um scanner que tenha os binários necessários para realizar a tradução e envio do código da aplicação, respectivamente chamados *source analyzer* e *cloudscan*. A utilização destes binários é relativamente simples, apenas sendo necessária a passagem de alguns parâmetros em um arquivo de configuração no projeto da aplicação.

Abaixo as variáveis comuns a todas as linguagens:

- FORTIFY_APPLICATION_LANGUAGE
    - Descrição: Linguagem utilizada pela aplicação
    - Valores aceitos: abap, asp, c, cpp, csharp, classicasp, golang, java, javascript, jsp, objectivec, php, psql, python, ruby, tsql, vbdotnet, vb6, vbscript, xml.
- FORTIFY_APPLICATION_TYPE

    - Descrição: Tipo da aplicação
    - Valores aceitos:
        - API: para APIs
        - WA: web application
        - WS: web service
        - RO: remote object exposure
        - ICI: interactive console
        - BP: batch processing console
        - GUI: graphical user interface
- FORTIFY_AUTHENTICATION_TYPE
    - Descrição: Tipo de autenticação utilizada na aplicação
    - Valores aceitos:
        - None: Nenhuma autenticação
        - BASIC: Básica
        - DIGEST: Baseada em servidor HTTP
        - FORM: Realizada por formulário
        - CLIENT-CERT: Baseada em certificados de cliente
        - SSO: Autenticação delegada a SSO
        - JAAS: Java Authentication and Authorization Service
        - Custom: Desenvolvida internamente ou outra
- FORTIFY_PATH
    - Descrição: Caminho contendo o código a ser analisado
    - Valores aceitos: Qualquer caminho válido de sistemas UNIX
- FORTIFY_PROJECTNAME
    - Descrição: Nome padronizado do projeto
    - Valores aceitos: Textos sem espaços ou caracteres especiais
- FORTIFY_PATH_EXCLUSIONS
    - Descrição: Lista de expressões regulares de arquivos a serem ignorados na análise, como arquivos de testes, etc.
    - Valores aceitos: Lista separada por ponto e vírgula de qualquer regex válida de sistemas UNIX que descrevam diretórios ou arquivos dentro do caminho FORTIFY_PATH

4.2. *SONARQUBE*

Sonarqube é uma ferramenta *open source* para inspeção contínua da qualidade de código, para executar revisões automáticas com análise estática de código para detectar *bugs*, *code smells* e vulnerabilidades de segurança em mais de 20 linguagens de programação.

Oferece também, relatórios sobre código duplicado, padrões de codificação, testes unitários, cobertura de código, complexidade de código, comentários, erros e vulnerabilidades de segurança. Registra o histórico de métricas e fornecer gráficos de evolução, fornece análise e integração totalmente automatizadas com as ferramentas.

O *SonarQube* oferece as ferramentas necessárias para escrever código limpo e seguro:

- *SonarLint:* É um produto complementar que funciona em seu editor dando *feedback* imediato para que você possa detectar e corrigir problemas antes que eles cheguem ao repositório.
- *Quality Gate*: O *Quality* Gate permite que você saiba se seu projeto está pronto para produção.
- *Clean as You Code*: *Clean as You Code* é uma abordagem para a qualidade do código que elimina muitos dos desafios que vêm com as abordagens tradicionais. Como desenvolvedor, você se concentra em manter altos padrões e assumir a responsabilidade especificamente no novo código em que está trabalhando.
- *Issues*: O SonarQube levanta problemas sempre que uma parte do seu código quebra uma regra de codificação, seja um erro que quebrará seu código (*bug*), um ponto em seu código aberto a ataques (vulnerabilidade) ou um problema de manutenção (*code smell*).
- *Security Hostpots*: O SonarQube destaca trechos de código sensíveis à segurança que precisam ser revisados. Após a análise, você descobrirá que não há ameaça ou precisa aplicar uma correção para proteger o código.

4.3. ANÁLISE DE CORRELAÇÃO

O *Fortify* classifica essencialmente os problemas de qualidade de código em termos de seu impacto de segurança na solução. Embora o Sonarqube seja mais uma ferramenta de análise de código estático, que também apresenta code *smell*, ele também lista as vulnerabilidades como parte de sua análise.

No entanto, a maior diferença é em termos de custo. O Sonarqube é gratuito para uso (com suporte da comunidade), enquanto o *Fortify* precisa de uma licença, que tem o custo elevado.

Esse *benchmark* foi realizado, uma vez que elencado os produtos usados em mercado, de forma a avaliar como cada um funciona e o que propõe. Desta forma foi possível elaborar uma proposta de produto, que serve como um guia para orientar com as melhores práticas de segurança, o desenvolvimento de um *software*.

5. PRODUTO

A proposta de do nosso produto “Guia – *Security Development*” é ser um guia com as melhores práticas, de forma à auxiliar o desenvolvedor a implementar as técnicas de segurança durante o desenvolvimento de aplicações/*softwares* que utilizem código aberto.

A divisão dele foi realizada conforme sugerido pelo Microsoft SDL, pois traz momentos desde o de pré-projeto, até desenvolvimento, implantação e pós-projeto que são: orientação de uso, treinamento, requisitos, *design*, implementação, verificação e liberação. Vale ressaltar que mesmo que o foco seja as aplicações o*pen source*, este guia também pode ser adequado e utilizado para desenvolvimento de outras aplicações.

5.1. CAPA

A abertura do guia conta com algumas breves orientações de uso da planilha, para que os desenvolvedores possam realizar os projetos de aplicações com conhecimento das melhores práticas de segurança.

Figura 04 – Capa do *Framework*

Guia – Security Development

Objetivo:	Orientar aos desenvolvedores de aplicação de código aberto a implementar práticas de segurança
Orientação:	Cada aba desta planilha contém as principais práticas de segurança levantas e pode ser alterado conforme o roteiro de desenvolviento da aplicação.
Materia de Base:	Código Aberto: Guia com as melhores práticas para o desenvolvimento de aplicações seguras.
Desenvolvido por:	ANDERSON ALMEIDA DOS SANTOS ARIADYNE SILVA DE OLIVEIRA LAURA REGINA DE SANTANA LUCIANE HISSAE TANAKA MOREIRA MARCIA FLORENCIO LEITE

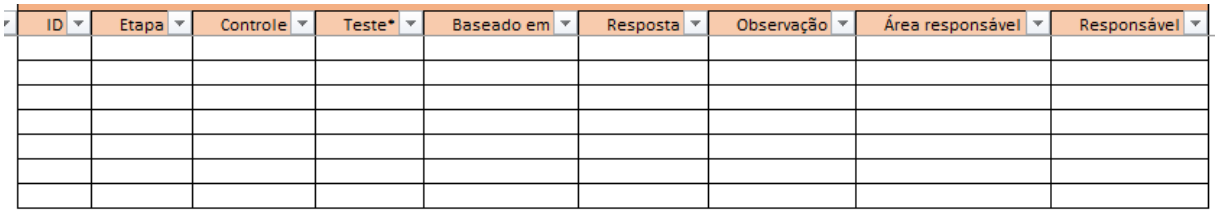
Fonte – Autores (2022)

Como base para a elaboração deste guia, foi usado a pesquisa bibliográfica desenvolvida neste trabalho de conclusão de curso, no qual foi levantado os materiais de apoio para construção do referencial teórico, buscando informações sobre o desenvolvimento seguro, de forma a propor o modelo do formato do guia.

5.2. FORMATO DO GUIA

A proposta do formato do guia, é que em cada aba da planilha há informações como demostrado na figura 05, e descrito a seguir:

Figura 05 – Cabeçalho do *Framework*



Fonte – Autores (2022)

- ID: Identificador para organizar os tópicos do projeto.
- Etapa: Tema principal a ser avaliado
- Controle: O tema que será avaliado
- Teste: Questionamento que será realizado para o projeto
- Baseado em: documentação/ referência bibliográfica que suporta este controle
- Resposta: múltipla escolha em – Atende, atende parcialmente, não atende e não aplicável
- Observação: Justificativa da resposta informada
- Área Responsável: Cada etapa terá um time como responsável
- Responsável: Ponto focal da atividade a ser realizada.

Diante de tantas vulnerabilidades apresentadas, formas de mitigar o risco de elas serem exploradas e baseado nas questões desenvolvidas neste trabalho, assim foi elaborado a criação do guia. Iremos apresentar nos subcapítulos seguintes, o que em cada etapa do ciclo de desenvolvimento, deve ser verificada.

Vale ressaltar que que os campos da planilha de “*Resposta, Observação, Área responsável, Responsável*” devem ser preenchidos pela equipe de desenvolvimento. Portanto iremos apresentar as partes de “*ID, Etapa, Controle, Teste e Baseado em*”, pois são os questionamentos formulados para a construção do produto.

5.3. TREINAMENTO

Com base no estudo realizado com as referências bibliográficas e companhia, observa-se que antes de iniciar qualquer projeto, é necessário que o desenvolvedor seja treinado de acordo com as ferramentas que utilizarão e com os princípios de segurança da informação e privacidade, além da modelagem das ameaças que circula uma aplicação.

Para isto dividimos nos seguintes tópicos:

- **Design de segurança** (Figura 06): É fundamental pensar em segurança desde o início do projeto (*security by design*), e para isto o desenvolver precisa ter conhecimento em relação as estruturas de segurança desde os acessos da informação; de como e quais dados necessitam de um nível de segurança adequado; conhecer a infraestrutura que suportará as APIs e até mesmo como será realizado o acesso à essas APIs;

Figura 06 – *Design* de segurança

ID	Etapa	Controle	Teste	Baseado em
TR-01	Design de segurança	Redução de superfície de ataque, defesa em profundidade, princípio do privilégio mínimo e padrões seguros	Os desenvolvedores realizaram treinamento da plataforma utilizada para o desenvolvimento?	SDL / OpenSSF
TR-01.01			Os desenvolvedores realizaram treinamento de gestão de acesso à plataforma?	SDL / OpenSSF
TR-01.02			Os desenvolvedores realizaram treinamento sobre os padrões de segurança da plataforma utilizada?	SDL / OpenSSF
TR-01.03			Os desenvolvedores realizaram treinamento sobre os principios de segurança da informação?	SDL / OpenSSF
TR-01.04			Os desenvolvedores realizaram treinamento das ferramentas utilizadas?	SDL / OpenSSF
TR-01.05			Os desenvolvedores realizaram treinamento de desenvolvimento seguro nos últimos 12 meses?	SDL / OpenSSF

Fonte – Autores (2022)

- **Modelagem de ameaças** (Figura 07): Conhecer os tipos de ameaças que possam causar danos a aplicação, é fundamental para que possa pensar e construir medidas de proteção no desenvolvimento do código.

Figura 07 – Modelagem de ameaças

ID	Etapa	Controle	Teste	Baseado em
TR-02	Modelagem de ameaças	Visão geral da modelagem de ameaças, implicações de design de um modelo de ameaças, codificação de restrições baseadas em um modelo de ameaças	Os desenvolvedores realizaram treinamento sobre as ameaças que possam ser exploradas na aplicação?	SDL / OpenSSF
TR-02.01			Os desenvolvedores realizaram treinamentos com base no resultado dos teste da aplicação?	SDL / OpenSSF
TR-02.02			Os desenvolvedores realizaram treinamento de como mitigar o risco das ameaças e explorarem as vulnerabilidade na aplicação?	SDL / OpenSSF
TR-02.03			Os desenvolvedores realizaram treinamentos de como verificar se o código está seguro?	SDL / OpenSSF
TR-02.04			Os desenvolvedores realizaram treinamentos de aplicação de criptografia na aplicação?	SDL / OpenSSF

Fonte – Autores (2022)

- **Codificação de segurança** (Figura 08): Seguindo a orientação da metodologia de *security by design* e considerando as ameaças novas que surgem a todo momento, o desenvolvedor precisa ter treinamentos de como detectar e pesquisar novas ameaças existentes.

Figura 08 – Codificação de segurança

ID	Etapa	Controle	Teste	Baseado em
TR-03	Codificação de segurança	Estouros de buffer, erros aritméticos inteiros, script entre sites, injeção de SQL, criptografia fraca	Foi realizado mapeamento de bibliotecas de codificação segura?	SDL
TR-03.01			Foi realizado mapeamento de públicações de CVE?	OWASP
TR-03.02			Os desenvolvedores realizaram treinamentos de codificação segura?	SDL
TR-03.03			Os desenvolvedores possuem treinamento de mitigar estas vulnerabilidade?	

Fonte – Autores (2022)

- ***Design* inseguro** (Figura 09): Além de aprender sobre os pontos do *design* seguro, é necessário reconhecer quais ações são inseguras para o desenvolvimento da aplicação.

Figura 09 – *Design* inseguro

ID	Etapa	Controle	Teste	Baseado em
TR-04	Design inseguro	Cultura e metodologia e não um complemento que será inserido na aplicação e sim consiste em todo um ciclo de vida de desenvolvimento seguro.	Foi estabelecido um ciclo de desenvolvimento seguro realizado em conjunto com profissionais do AppSec?	OWASP
TR-04.01			Há profissionais de segurança junto aos times de desenvolvedores?	OWASP
TR-04.02			Foi estabelecido na empresa uma cultura de cyber segurança?	OWASP
TR-04.03			Há integração de linguagens de segurança entre os usuários e desenvolvedores?	OWASP
TR-04.04			Há treinamento de compatibilidade de linguagens de programação ou outros softwares?	OpenSSF

Fonte – Autores (2022)

- **Teste de segurança** (Figura 10): Todas as aplicações precisam ser testadas e homologadas antes de serem implementadas em ambiente de produção. Os testes de segurança de uma aplicação devem ser realizados e os desenvolvedores precisam ter conhecimento de quais testes foram realizados.

Figura 10 – Teste de segurança

ID	Etapa	Controle	Teste	Baseado em
TR-05	Teste de segurança	Diferenças entre os testes de segurança e os testes funcionais, avaliação de riscos, métodos de teste de segurança	Há treinamento dos tipos de testes realizados na aplicação?	SDL
TR-05.01			Os desenvolvedores são treinados para realizar testes funcionais da aplicação?	OWASP
TR-05.02			Os desenvolvedores são treinados para métodos de teste de segurança?	OWASP

Fonte – Autores (2022)

- **Privacidade** (Figura 11): Devido as legislações mundiais vigentes (GDPR e a LGPD), é necessário saber reconhecer os tipos de dados que serão coletados na aplicação, qual o tratamento e o nível de segurança que será necessário.

Figura 11 – Privacidade

ID	Etapa	Controle	Teste	Baseado em
TR-06	Privacidade	Tipos de dados importantes de privacidade, práticas recomendadas de design de privacidade, avaliação de riscos, práticas recomendadas de desenvolvimento de privacidade, práticas recomendadas de teste de privacidade	Foi realizado nos últimos 12 meses treinamento de Privacidade aos desenvolvedores?	SDL
TR-06.01			Foi realizado o treinamento de LGPD sobre dados pessoais e sensíveis?	SDL
TR-06.02			Foi realizado o treinamento de LGPD sobre anonimização dos dados?	SDL
TR-06.03			Foi realizado o treinamento de LGPD sobre as melhores práticas para o *privacy by designe* ?	SDL

Fonte – Autores (2022)

5.4. REQUISITOS

Após o treinamento, ainda em fase de “pré-projeto”, é fundamental levantar os requisitos do projeto para criar a cultura do *security by design*, pois o desenvolvedor terá mapeado as operações realizadas pela aplicação e terá a visibilidade de quais dados serão coletados e qual a medida de segurança necessária, entre outros.

Como sugestão de identificar todos os pontos principais de segurança do projeto, elaboramos os seguintes tópicos para levantar estas informações:

- **Requisitos do projeto** (Figura 12): Neste ponto, o desenvolvedor pensará no nível operacional da aplicação.

Figura 12 – Requisitos do projeto

ID	Etapa	Controle	Teste	Baseado em
RE-01	Requisitos do projeto	Cada negócio exige um tipo de operação para que a máxima performance seja atingida, assim como seja minimizado o risco de erros	Levantamento e listagem de requisitos da aplicação para funcionamento	AZEVEDO JUNIOR e CAMPOS
RE-01.01			O aplicativo coletará dados confidenciais?	SDL
RE-01.02			O aplicativo coleta ou armazena dados que exigem a adesão a padrões do setor e programas de conformidade como o FFIEC (Federal Financial Institution Examination Council) ou o PCI DSS (Padrão de Segurança de Dados do Setor de Cartões de Pagamento)?	SDL
RE-01.03			Se o aplicativo estiver disponível para o público, como você protegerá os dados que poderão ser coletados contra uso indevido?	SDL

Fonte – Autores (2022)

- **Requisitos de segurança** (Figura 13): Este tópico foi formulado basicamente pelos questionamentos encontrados no *Microsoft Docs*, que aponta sobre além de questões relacionados à segurança, se faz necessário identificar os dados relacionados à privacidade do usuário.

- **Padrões de qualidade** (Figura 14): Pensando na otimização de tempo para configuração de ambiente e na trilha de auditoria, o ideal é definir nesta etapa, qual o nível de acesso de cada desenvolvedor e em qual será o ambiente de atuação.

Figura 13 – Requisitos de segurança

ID	Etapa	Controle	Teste	Baseado em
RE-02	Requisitos de segurança	Essa definição inicial de requisitos permite que as equipes de desenvolvimento identifiquem as etapas e os resultados finais principais e permite a integração da segurança e da privacidade de uma forma que diminua qualquer interrupção de planos e cronogramas	O aplicativo coleta ou contém dados confidenciais pessoais ou dos clientes que podem ser usados, por conta própria ou com outras informações para identificar, contatar ou localizar uma pessoa?	SDL
RE-02.01			O aplicativo coleta ou contém dados que podem ser usados para acessar informações médicas, acadêmicas, financeiras ou trabalhistas de um indivíduo?	SDL
RE-02.02			Onde e como os dados coletados na aplicação são armazenados?	SDL
RE-02.03			Será possível influenciar o registro em log para coletar dados mais detalhados e analisar um problema detalhadamente?	SDL
RE-02.04			O aplicativo estará disponível para o público (na Internet) ou apenas internamente?	SDL
RE-02.05			Se o aplicativo estiver disponível apenas internamente, considere quem em sua organização deve ter acesso a ele e por quanto tempo.	SDL
RE-02.06			Você entende o seu modelo de identidade antes de começar a projetar o aplicativo?	SDL
RE-02.07			Como você determinará que os usuários são quem dizem ser e o que um usuário está autorizado a fazer?	SDL
RE-02.08			O meu aplicativo executa tarefas confidenciais ou importantes (como transferir dinheiro, destrancar portas ou entregar remédios)?	SDL
RE-02.09			Considere como você validará que o usuário executando uma tarefa confidencial está autorizado a executar a tarefa e como você autenticará que a pessoa é quem diz ser.	SDL
RE-02.10			Meu aplicativo executa atividades de software arriscadas, como permitir que usuários carreguem ou baixem arquivos ou outros dados?	SDL
RE-02.11			Se o aplicativo executar atividades arriscadas, considere como o aplicativo protegerá os usuários de lidar com arquivos ou dados mal-intencionados.	SDL
RE-02.12			Como detectarei um ataque?	SDL / OWASP
RE-02.13			O que farei se houver um ataque ou uma violação?	SDL / OWASP
RE-02.14			Como vou me recuperar do ataque, como vazamento de dados ou adulteração?	SDL / OWASP

Fonte – Autores (2022)

Figura 14 – Padrões de qualidade

ID	Etapa	Controle	Teste	Baseado em
RE-03	Padrões de qualidade	A definição desses critérios no início de um projeto melhora o entendimento dos riscos associados aos problemas de segurança e permite que as equipes identifiquem e corrijam os erros de segurança durante o desenvolvimento	Os portões de qualidade e as barras de erros são usados para estabelecer níveis mínimos aceitáveis de qualidade de segurança e de privacidade?	SDL
RE-03.01			Foi levantado quais as configurações necessárias do ambiente de desenvolvimento / teste / produção?	OWASP
RE-03.02			Foi definido quais serão os níveis de acesso/privilégio de cada membro da equipe de desenvolvimento?	OWASP
RE-03.03			Foi levantado se haverá atualizações futuras daquela aplicação? Se sim, qual a periodicidade de atualização?	OWASP
RE-03.04			Foi definida a arquitetura de rede que será implementado a aplicação?	OWASP

Fonte – Autores (2022)

- **Avaliação de riscos de segurança e de privacidade** (Figura 15): Nesta etapa a equipe de desenvolvimento deverá realizar uma análise e

avaliar os riscos de segurança e privacidade que poderá conter na aplicação, com o objetivo de mitigar ou monitorar o risco.

Figura 15 – Avaliação de riscos de segurança e de privacidade

ID	Etapa	Controle	Teste	Baseado em
RE-04	Avaliação de riscos de segurança e de privacidade	As SRAS (Avaliações de riscos de segurança) e as PRAS (Avaliações de riscos de privacidade) são processos obrigatórios que identificam os aspectos funcionais do software	Quais partes do projeto requerem modelos de ameaças antes da liberação?	SDL
RE-04.01			Quais partes do projeto requerem revisões do design de segurança antes da liberação?	SDL

Fonte – Autores (2022)

5.5. DESIGN

Design é o momento em que a equipe do projeto dará “vida” à aplicação, o *hands-on*, porém é necessário validar se o que será desenvolvido vai ser feito de forma segura. Portanto trouxemos os seguintes tópicos:

- **Requisitos de *design*** (Figura 16): Uma observação que pode ser feita: Por que este tópico não se encontra na sessão 5.4 Requisitos? Consideramos que está etapa está mais próxima da escrita do código do que na visão macro do projeto.

- **Falha no controle de acesso** (Figura 17): Este é o momento em que a equipe de desenvolvimento do projeto deverá pensar e analisar quais serão os níveis de interação do usuário com a aplicação e evitar falhas no controle de acesso.

- **Redução de superfície de ataque** (Figura 18): Sempre que possível, o desenvolvedor deverá pensar em uma forma de reduzir as probabilidades de algum atacante explorar uma vulnerabilidade. Nesta etapa está focando em superfície de rede.

Figura 16 – Requisitos de *design*

ID	Etapa	Controle	Teste*	Baseado em
DE-01	Requisitos de design	O momento ideal para influenciar a confiabilidade do design de um projeto é o início de seu ciclo de vida. É muito importante considerar as questões de segurança e de privacidade cuidadosamente durante a fase de design	As especificações de design descrevem os recursos de segurança e de privacidade que serão expostos diretamente para o usuário?	SDL
DE-01.01			Será utilizado uma biblioteca de codificação segura e uma estrutura de software?	SDL
DE-01.02			Foi verificado se há componentes vulneráveis na aplicação?	SDL
DE-01.03			Será utilizado alguma modelagem de ameaças durante o design do aplicativo?	SDL
DE-01.04			Como reduzir a superfície de ataque?	SDL
DE-01.05			Será adotado uma política de identidade como o perímetro de segurança primário?	SDL
DE-01.06			Será exigido uma nova autenticação para transações importantes?	SDL
DE-01.07			Será exigido um duplo fator de autenticação para acesso a transações importantes?	SDL
DE-01.08			Será utilizado alguma uma solução de gerenciamento de chaves para proteger chaves, credenciais e outros segredos?	SDL
DE-01.09			Como será a proteção de dados confidenciais ou sensíveis?	SDL
DE-01.10			Será implementado medidas à prova de falhas?	SDL
DE-01.11			Será aproveitado o tratamento de erros e exceções?	SDL
DE-01.12			Utilizará o registro em log e os alertas?	SDL

Fonte – Autores (2022)

Figura 17– Falha no controle de acesso

ID	Etapa	Controle	Teste*	Baseado em
DE-02	Falha no controle de acesso	Delimitar o acesso e as interações com o sistema para cada usuário	Implementar os códigos em um servidor confiável e as APIs sem servidor	OWASP
DE-02.01			Como será o armazenamento dos logs (trilha de auditoria) e alertas da aplicação?	OWASP
DE-02.02			Será negado por padrão o acesso a todos os recursos não públicos?	OWASP
DE-02.03			Será implementado mecanismos de controle de acesso? Se sim, quais?	OWASP
DE-02.04			Será definido quais serão os tipos de permissão (leitura, escrita, execução) a cada tipo de usuário da aplicação?	OWASP
DE-02.05			Será desativado as listagens de diretórios web?	OWASP
DE-02.06			Foi certificado que os arquivos de dados e backup não estejam em acesso à web?	OWASP
DE-02.07			Será monitorado, alertado e registrado as falhas para os administradores?	OWASP
DE-02.08			Será definido a quantidade de acesso a API e a controladora?	OWASP
DE-02.09			Será utilizado tokens de curta duração?	OWASP
DE-02.10			Seguirá os procedimentos de AuthO para revogação de acesso?	OWASP

Fonte – Autores (2022)

Figura 18 – Redução de superfície de ataque

ID	Etapa	Controle	Teste*	Baseado em
DE-03	Redução de superfície de ataque	A redução da superfície de ataque está intimamente alinhada com a modelagem de ameaças, embora ela aborde as questões de segurança sob uma perspectiva um pouco diferente	A redução da superfície de ataque abrange o fechamento ou a restrição do acesso aos serviços do sistema, aplicando o princípio de privilégio mínimo e empregando defesas em camadas sempre que possível.	OWASP
DE-03.01			Foi examinado qual o tamanho da superfície de ataque?	SDL
DE-03.02			Foi removido do código recursos que você ainda não lançou/implementou?	SDL
DE-03.03			Foi removido do código de suporte à depuração?	SDL
DE-03.04			Foi removido os adaptadores de rede e protocolos que não são usados ou que foram preteridos?	SDL
DE-03.05			Foi removido as máquinas virtuais e outros recursos que você não está usando?	SDL

Fonte – Autores (2022)

- **Falha de criptografia** (Figura 19): O uso de criptografia é fundamental para garantir a integridade dos dados e deve ser aplicado em dados pessoais sensíveis e confidenciais/secretos. Portanto, o desenvolver precisa compreender qual o melhor tipo de criptografia a ser utilizada na aplicação.

Figura 19 – Falha de criptografia

ID	Etapa	Controle	Teste*	Baseado em
DE-04	Falha de criptografia	Definir quais dados que precisarão de criptografia em trânsito e em repouso, qual o nível da criptografia	Utilize chaves criptográficas atualizadas Utilização de protocolos de criptográficas para as aplicações Web	OWASP
DE-04.01			Algum dado é transmitido em texto simples?	SDL / OWASP
DE-04.02			Ainda é usado algum protocolo/algoritmo criptografado desatualizado ou fraco?	SDL / OWASP
DE-04.03			As chaves são reutilizadas? Há um gerenciamento ou rotação de chaves implementado?	SDL / OWASP
DE-04.04			Há algum parâmetro de segurança/critopgrafia de cabeçalho? Uso de HTTP/HTTPS, por exemplo.	SDL / OWASP
DE-04.05			Estão devidamente validados os certificados do servidor e a cadeia de confiança?	SDL / OWASP
DE-04.06			As chaves criptográficas foram projetadas e planejadas para atender os requisitos de SI e Privacidade da aplicação?	SDL / OWASP
DE-04.07			A aplicação possuí o uso de hash obsoletos? E utilizam hash com ou sem criptografia quando necessário?	SDL / OWASP
DE-04.08			Possuí monitoramento, alerta e registro de falha de criptografia?	SDL / OWASP

Fonte – Autores (2022)

- **Modelagem de ameaças** (Figura 20): Aplicar no código as medidas de modelagem de ameaças no ambiente de risco.

Figura 20 – Modelagem de ameaças

ID	Etapa	Controle	Teste*	Baseado em
DE-05	Modelagem de ameaças	A modelagem de ameaças é usada em ambientes onde há um risco de segurança significativo.	É uma prática que permite que as equipes de desenvolvimento considerem, documentem e discutam as implicações de segurança de designs no contexto de seu ambiente operacional planejado e de uma maneira estruturada	SDL

Fonte – Autores (2022)

- **Dependência de Software** (Figura 21): Com base na documentação do OpenSSF, inserimos as questões (citadas no capítulo 2.5 Dependências com outros softwares) dentro do guia, pois não adianta pensar em segurança no desenvolvimento do código e não validar a segurança das APIs e softwares, que forem se comunicar com a aplicação.

Figura 21 – Dependência de *Software*

ID	Etapa	Controle	Teste*	Baseado em
DE-06	Dependência de Software	Tudo o que o objeto precisa para funcionar é chamado de dependência. Cada dependência cria um caminho para vulnerabilidades (não intencionais e intencionais). Avaliação o software antes de selecioná-lo como uma dependência direta da aplicação que está sendo desenvolvida.	Ao instalar uma nova dependência a sua aplicação deve verificar o nome do pacote para combater erros de digitação para realizar a comunicação correta	OpenSSF
DE-06.01			Deve ser adicionado?	OpenSSF
DE-06.02			Foi revisado a última alteração do código?	OpenSSF
DE-06.03			É fácil de usar com segurança?	OpenSSF
DE-06.04			Há evidências de que os desenvolvedores atuam para tornar a aplicação segura?	OpenSSF
DE-06.05			Há evidências de que os desenvolvedores usam ferramentas para detectar defeitos e vulnerabilidades o mais cedo possível?	OpenSSF
DE-06.06			Há uma documentação explicando por que os desenvolvedores acreditam que é seguro ("caso de garantia")?	OpenSSF
DE-06.07			Há evidências de uma auditoria de segurança, e se detectado algum problema, foi corrigido?	OpenSSF
DE-06.08			Existem instruções sobre como relatar vulnerabilidades?	OpenSSF
DE-06.09			Este software tem atualizações?	OpenSSF
DE-06.10			Tem uso significativo [no mercado]?	OpenSSF
DE-06.11			Qual é a licença do software?	OpenSSF
DE-06.12			Se é importante, qual é a sua própria avaliação?	OpenSSF
DE-06.13			Há uma rigorosa validação de entrada para entrada não confiável e o uso de declarações preparadas?	OpenSSF
DE-06.14			Existem evidências do código estar inseguro ou incompleto?	OpenSSF
DE-06.15			Há evidências de que o software é malicioso?	OpenSSF

Fonte – Autores (2022)

5.6. IMPLEMENTAÇÃO

Momento que será validado toda a estrutura do projeto, pois nesta etapa serão realizados os testes operacionais e de segurança da aplicação relacionado ao código. Para isto dividimos nos seguintes tópicos:

- **Utilizar ferramentas aprovadas** (Figura 22): Todas as ferramentas que são utilizadas no projeto devem passar pela equipe de segurança para

serem testadas e homologadas o uso delas na companhia, pois a equipe de segurança do projeto deve averiguar os requisitos de segurança das ferramentas utilizada e se as mesmas não irão criar novas vulnerabilidades.

Figura 22 – Utilizar ferramentas aprovadas

ID	Etapa	Controle	Teste*	Baseado em
IM-01	Utilizar ferramentas aprovadas	Todas as equipes de desenvolvimento devem definir e publicar uma lista de ferramentas aprovadas e suas verificações de segurança associadas, como as opções e os avisos de compilador/vinculador.	As equipes de desenvolvimento devem se esforçar para usar a versão mais recente das ferramentas aprovadas para aproveitar as novas funcionalidades e proteções de análise de segurança.	SDL
IM-01.01			A equipe definiu quais ferramentas de desenvolvimento irão utilizar?	SDL
IM-01.02			Foi homologado pelo time de Segurança da Informação o uso das ferramentas?	SDL
IM-01.03			As ferramentas definidas são de terceiros? Se sim, o fornecedor garante suporte e requisitos mínimos de SI?	SDL
IM-01.04		SonarLint	É um produto complementar que funciona em seu editor dando feedback imediato para que você possa detectar e corrigir problemas antes que eles cheguem ao repositório.	SonarQube
IM-01.05		Clean as You Code	É uma abordagem para a qualidade do código que elimina muitos dos desafios que vêm com as abordagens tradicionais.	SonarQube

Fonte – Autores (2022)

- **Entrada / Saída de dados** (Figura 23): Entre toda a aplicação existe fluxo de dados sendo trafegado. Para isto o desenvolvedor deve pensar em como garantir a segurança nas entradas e saídas dos dados do *software*.

Figura 23 – Entrada / Saída de dados

ID	Etapa	Controle	Teste*	Baseado em
IM-02	Entrada / saída de dados	Validar e limpar todas as entradas do aplicativo	Foi validado a entrada no início do fluxo de dados para ter certeza de que apenas os dados formados corretamente entrarão no fluxo de trabalho	SDL
IM-02.01			Foi realizado uma lista de acessos bloqueados e de permitidos?	SDL
IM-02.02			A entrada para o seu aplicativo inclui parâmetros na URL, entradas do usuário, dados do banco de dados ou de uma API?	SDL
IM-02.03			Os dados estão sendo processando dados com segurança?	SDL
IM-02.04			Os envio de saída de dados estão de acordo com o requisito?	SDL

Fonte – Autores (2022)

- **Componentes vulneráveis e desatualizados** (Figura 24): além de monitorar os alertas de novas vulnerabilidades identificadas, a equipe de desenvolvimento da aplicação deve se atentar se as dependências da

aplicação estão seguindo as medidas de segurança e atualizações periódicas de correção de novas vulnerabilidades.

Figura 24 – Componentes vulneráveis e desatualizados

ID	Etapa	Controle	Teste*	Baseado em
IM-03	Componentes vulneráveis e desatualizados	Quando uma vulnerabilidade é reportada e torna-se pública, paths de segurança são disponibilizados pelo fabricante, porém nem todas as corporações irão efetuar a atualização imediatamente	Necessário conhecer todos os componentes de software que são utilizados, sempre analisando o lado cliente e servidor; para detectar se os softwares estão vulneráveis, desatualizados ou se já não possuem mais suporte pelo fabricante	OWASP
IM-03.01			A equipe de desenvolvimento detectou todos os componentes que são utilizados no software?	OWASP
IM-03.02			Foi verificado o monitoramento de publicação de novas vulnerabilidades são fundamentais e aplicação de correção do patches de segurança?	OWASP
IM-03.03			Foi realizado testes de compatibilidade com as bibliotecas?	OWASP
IM-03.04			Foi realizado as configurações corretas dos componentes?	OWASP

Fonte – Autores (2022)

- **Desaprovar funções não seguras** (Figura 25): Momento de realizar *hardening* do código, verificando todas as funções da aplicação e servidores que não sejam utilizadas estejam desativadas, e habilitar somente os recursos necessários para realizar somente a função destinada, isto para não criar portas a serem exploradas por vulnerabilidade.

Figura 25 – Desaprovar funções não seguras

ID	Etapa	Controle	Teste*	Baseado em
IM-04	Desaprovar funções não seguras	Muitas funções e APIs comumente usadas não são seguras frente ao ambiente atual de ameaças.	As equipes de projeto devem analisar todas as funções e APIs que serão usadas em conjunto com um projeto de desenvolvimento de software e proibir aquelas que estão determinadas como não seguras.	SDL
IM-04.01			Seguir as recomendações do item DE-06	OpenSSF

Fonte – Autores (2022)

- ***Software and Data Integrity Failures (Figura 26):*** Nesta etapa o desenvolvedor deverá garantir, durante a implantação, a integridade do *software* e dos dados que ali trafegam.

- ***Server-Side Request Forgery*** (Figura 27): Partindo do princípio do conceito de segurança em profundidade, é preciso validar se está sendo implementado os controles de segurança não somente da aplicação, como também na infraestrutura de rede que irá sustentar o *software*.

Figura 26 – *Software and Data Integrity Failures*

ID	Etapa	Controle	Teste*	Baseado em
IM-05	Software and Data Integrity Failures	Atualizações automáticas dos softwares. Momento em que a empresa desenvolvedora pode colocar um pedaço de script de um malware, sem perceber por usar a biblioteca pública de códigos abertos.	Foi utilizado assinaturas digitais ou mecanismos semelhantes para verificação do software ou dos dados são da fonte confiável e não foram alterados?	OWASP
IM-05.01			Foi validado que bibliotecas e dependências estejam integradas a repositórios confiáveis?	OWASP

Fonte – Autores (2022)

Figura 27 – *Server-Side Request Forgery* (SSRF)

ID	Etapa	Controle	Teste*	Baseado em
IM-06	Server-Side Request Forgery (SSRF)	Um aplicativo da Web está buscando um recurso remoto sem validar a URL	Implementar um controle de segurança em profundidade na camada de rede e aplicação	OWASP

Fonte – Autores (2022)

- **Análise estática** (Figura 28): Método onde é examinado o código fonte antes do programa ser executado. Neste momento o desenvolvedor deverá analisar o resultado do exame, para corrigir das vulnerabilidades detectadas. Nesta etapa pode-se utilizar as ferramentas SAST, conforme no capítulo 3.2 deste documento.

Figura 28 – Análise estática

ID	Etapa	Controle	Teste*	Baseado em
IM-07	Análise estática	A análise estática do código-fonte fornece uma capacidade escalável de revisão de código de segurança e pode ajudar a assegurar que as políticas de codificação seguras estejam sendo seguidas	Serão utilizadas ferramentas de analise estática? Se sim, quais?	OpenSSF
IM-07.01			Foi realizado a análise de fluxo de dados?	OWASP
IM-07.02			Foi identificação variáveis que foram 'contaminadas'?	OWASP
IM-07.03			Foi realizado a análise Lexical?	OWASP
IM-07.04			A verificação retornou com erros ou vulnerabilidades?	OWASP
IM-07.05			A análise de código estática sozinha é geralmente insuficiente para substituir uma revisão de código manual. A equipe de segurança e os consultores de segurança devem estar cientes das forças e fraquezas das ferramentas de análise estática e devem estar preparados para acrescentar às ferramentas de análise estática outras ferramentas ou revisão humana, como for apropriado	OWASP

Fonte – Autores (2022)

5.7. VERIFICAÇÃO

Etapa em que é realizado os testes de segurança e a última verificação antes de liberar para a implantação do produto. Para isto dividimos nos seguintes tópicos:

- **Análise de programa dinâmica** (Figura 29): Etapa de análise do código execução para verificar a integridade do fluxo de dados da aplicação.

Figura 29 – Análise de programa dinâmica

ID	Etapa	Controle	Teste*	Baseado em
VE-01	Análise de programa dinâmica	de software é necessária para garantir que a funcionalidade de um programa funcione como planejado. Essa tarefa de verificação deve especificar as ferramentas que monitoram o comportamento do aplicativo	Essa tarefa de verificação deve especificar as ferramentas que monitoram o comportamento do aplicativo quanto à corrupção da memória, problemas de privilégio do usuário e outros problemas de segurança críticos.	SDL

Fonte – Autores (2022)

- **Teste de *fuzzing*** (Figura 30): Semelhante a análise dinâmica das ferramentas DAST, o *software* é testado usando ferramentas automáticas para fornecimento de entradas invalidas na aplicação, que é a indução de falha para detecção de falha de acesso.

Figura 30 – Teste de *fuzzing*

ID	Etapa	Controle	Teste*	Baseado em
VE-02	Teste de fuzzing	O teste de fuzzing é uma forma especializada de análise dinâmica usada para induzir a falha do programa ao introduzir deliberadamente dados defeituosos ou aleatórios a um aplicativo.	A estratégia de teste de fuzzing é derivada do uso planejado do aplicativo e das especificações funcionais e de design para o aplicativo.	SDL
VE-03		Tradução: O código fonte a ser analisado é traduzido para uma linguagem intermediária utilizada pelo Fortify.	Executada localmente por um cliente da solução. Neste estágio é necessário informar alguns parâmetros específicos da aplicação como: linguagem de codificação, versão da linguagem, nome da aplicação, versão da aplicação, localização dos códigos fonte, etc;	Fortify
VE-04		Envio: A tradução é enviada para que a varredura e análise sejam realizada	A tradução é enviada para que a varredura e análise sejam realizadas em um servidor do Fortify. Por esse motivo, são necessárias algumas informações sobre o servidor da solução nesta fase;	Fortify
VE-05		Varredura e Análise	O Fortify realiza a varredura e análise da aplicação que pode ser acompanhada pela console web.	Fortify
VE-06		Security Hostpots	O SonarQube destaca trechos de código sensíveis à segurança que precisam ser revisados. Após a análise, você descobrirá que não há ameaça ou precisa aplicar uma correção para proteger o código.	SonarQube

Fonte – Autores (2022)

- ***Security Misconfiguration*** (Figura 31): Momento de avaliar e testar a eficácia das configurações utilizadas da aplicação, para que ela garanta a operabilidade do *software* com os requisitos de configurações de segurança.

Figura 31 – *Security Misconfiguration*

ID	Etapa	Controle	Teste*	Baseado em
VE-07	Security Misconfiguration	Falta de uma segurança robusta apropriada em qualquer parte da aplicação; permissões configuradas incorretamente em serviços em nuvem; instalação ou ativação de recursos desnecessário	há segregação do ambiente entre desenvolvimento, teste e produção?	OWASP
VE-07.01			Há um processo de teste da eficácia de proteção do ambiente, validando as configurações e ajustes em todo o ambiente?	OWASP
VE-07.02			Uma plataforma de desenvolvimento instalado somente os recursos necessários para a realização da função da aplicação?	OWASP
VE-07.03			Há implementado a revisão periódica de atualização de patches de segurança da aplicação/plataforma, para evitar ataques e exposições zero-day?	OWASP
VE-07.04			Há uma arquitetura de rede segmentada em contêiner ou grupos de segurança em nuvem?	OWASP
VE-07.05			Foi envio de diretrizes de segurança ao usuário/cliente?	OWASP

Fonte – Autores (2022)

- ***Injection*** (Figura 32): A técnica de injeção foi abordada no tópico 2.1.3 deste documento. Por ser uma técnica amplamente conhecida, chama-se a atenção ao desenvolvedor durante o projeto de desenvolvimento da aplicação a mitigar e corrigir possíveis meios de exploração.de códigos maliciosos sejam injetados no código e facilitando a invasão à aplicação ou à todo ambiente da empresa.

Figura 32 – *Injection*

ID	Etapa	Controle	Teste*	Baseado em
VE-08	Injection	A aplicação não realiza a validação, filtra ou higieniza os dados imputados pelo usuário	Manutenção dos dados separados dos locais de comandos e consultas.	OWASP
VE-08.01			Utilização de uma API segura.	OWASP
VE-08.02			utilizar a validação positiva para o lado do servidor	OWASP
VE-08.03			escapes específicos nas consultas dinâmicas residuais	OWASP
VE-08.04			use comandos para evitar consultas e vazamento em massa do SQL	OWASP
VE-08.05			não utilizar nomes das estruturas fornecidas pelo usuário	OWASP

Fonte – Autores (2022)

- **Falhas de identificação e autenticação** (Figura 33): Gestão e controle de acesso devem ser pensados a todo momento, mas nesta etapa é avaliado se a aplicação consegue realizar a gestão de acesso e garante que o usuário dono seja o mesmo que está acessando, trabalhando portanto os princípios de autenticação e autorização do acesso.

Figura 33 – Falhas de identificação e autenticação

ID	Etapa	Controle	Teste*	Baseado em
VE-09	Falhas de identificação e autenticação	Atinge aplicações Web, acesso a redes, sistemas e até locais fisicos. Uso de credenciais fracas, padrão ou conhecidas	Implementação de multifator de autenticação (MFA), de preferência que utilizem tokens ou aplicativos;	OWASP
VE-09.01			Use um gerenciador de sessão integrado	OWASP
VE-09.02			Não use credenciais padrões.	OWASP
VE-09.03			Politica de complexidade de senha.	OWASP

Fonte – Autores (2022)

- **Modelo de ameaças e revisão da superfície de ataque** (Figura 34): É abordado em outras etapas a preocupação com ameaças e superfície de ataque. Porém é necessária nesta etapa específica realizar a verificação se foi mitigado a superfície de ataque antes de liberar a aplicação para o ambiente de produção, garantindo assim que a aplicação cumpra com os requisitos de segurança necessários.

Figura 34 – Modelo de ameaças e revisão da superfície de ataque

ID	Etapa	Controle	Teste*	Baseado em
VE-10	Modelo de ameaças e revisão da superfície de ataque	Desviar significativamente das especificações funcionais e de design criadas durante as fases de requisitos e de design de um projeto de desenvolvimento de software	Essa revisão garante que as alterações de design ou de implementação ao sistema sejam consideradas e que os novos vetores de ataque criados como resultado das alterações sejam revisados e mitigados.	SDL

Fonte – Autores (2022)

- ***Security Logging and Monitoring Failures*** (Figura 35): Etapa para validação se a aplicação está monitorando e registrando qualquer tipo de interação com o*software*. Esta etapa é fundamental para garantir a rastreabilidade do usuário, armazenamento de *logs* de acesso, permitindo uma análise forense em caso de algum tipo de incidente, podendo detectar como e onde ocorreu. Estas informações são importantes para correção e desenvolvimento futuro, assim não cometendo o mesmo “erro” durante o desenvolvimento.

- **Teste de Penetração** (Figura 36): O teste de penetração ou pentest, deve ser realizado para detectar qualquer vulnerabilidade e falha de

acesso, que possa a vim ser explorada por um atacante. Considera o teste para que a aplicação seja “a prova” de uma invasão

Figura 35 – *Security Loggin and Monitoring Failures*

ID	Etapa	Controle	Teste*	Baseado em
VE-11	Security Logging and Monitoring Failures	Falha de registro e monitoramento de segurança, e vem para auxiliar ao analista detectar, escalar, responder a violação/ incidente que ocorrerem na aplicação	Todas as ações realizadas pelo usuário, a aplicação deve estar configurada para registrar este evento.	OWASP
VE-11.01			Foi garantido que todas as falhas de login, controle de acesso e validação de entrada, possam ser registradas com contexto de usuário suficiente para identificar contas suspeitas ou maliciosas?	OWASP
VE-11.02			Está previsto manter os registros por tempo suficiente para permitir análises forenses futuras? Se sim, quanto tempo.	OWASP
VE-11.03			Os logs gerados estão em um formato que as soluções de gerenciamento de log possam consumir facilmente?	OWASP
VE-11.04			Foi certificado que os dados de log estão codificados corretamente?	OWASP
VE-11.05			As transações de alto valor possuem uma trilha de auditoria com controles de integridade para evitar adulteração ou exclusão?	OWASP
VE-11.06			Foi estabelecido o monitoramento e alertas eficazes para que atividades suspeitas sejam detectadas e respondidas rapidamente?	OWASP
VE-11.07			Há um plano de resposta e recuperação de incidentes?	OWASP

Fonte – Autores (2022)

Figura 36 – Teste de Penetração

ID	Etapa	Controle	Teste*	Baseado em
VE-12	Teste de Penetração	Verificar se seu aplicativo é seguro é tão importante quanto testar qualquer outra funcionalidade.	Realizou testes em seus *endpoint* para revelar as 10 maiores vulnerabilidades do OWASP (Projeto de Segurança de Aplicativo Web Aberto)?	SDL
VE-12.01			Foi realizado exame de portas de seus pontos de extremidade e aplicações?	SDL
VE-12.02			Foi simulados ataques de (D)DoS na aplicação?	SDL
VE-12.03			Realização do teste fuzzing	SDL
VE-12.04			Tentativas de phishing ou outros ataques de engenharia social contra nossos funcionários?	SDL
VE-12.05			Efetuar teste de carga de seu aplicativo gerando o tráfego esperado durante o curso de negócios normal. Isso inclui testar a capacidade de lidar com surtos.	SDL

Fonte – Autores (2022)

5.8. LIBERAÇÃO

Após toda a verificação de testes de segurança e operacional, a aplicação pode-se liberar a aplicação para ser implementada no ambiente de produção. Porém ressalta-se a importância destes dois tópicos a seguir, para garantir a continuidade e disponibilidade do *software*:

- **Plano de resposta de incidentes (Figura 37):** Qualquer aplicação a ser implementada, deve ser mapeada previamente as medidas de

recuperação do ambiente em caso de um incidente de segurança. Pois em caso de incidente, o time de recuperação deve responder conforme o plano de *roll-back* definido pelo time de desenvolvimento e Segurança da Informação. Estas medidas são para minimizar o impacto do incidente na organização e usuários.

Figura 37 – Plano de resposta de incidentes

ID	Etapa	Controle	Teste*	Baseado em
LI-01	Plano de resposta de incidentes	Cada liberação de software sujeita aos requisitos do SDL deve incluir um plano de resposta de incidentes. Mesmo os programas sem vulnerabilidades conhecidas no momento da liberação podem estar sujeitos a novas ameaças que surgem com o tempo.	Cada liberação de software sujeita aos requisitos do SDL deve incluir um plano de resposta de incidentes. Mesmo os programas sem vulnerabilidades conhecidas no momento da liberação podem estar sujeitos a novas ameaças que surgem com o tempo.	SDL

Fonte – Autores (2022)

- **Revisão final de segurança (Figura 38):** Etapa final do Guia, é a revisão final de segurança permitindo a liberação da aplicação para ser implementada no ambiente de produção.

Figura 38 – Revisão final de segurança

ID	Etapa	Controle	Teste*	Baseado em
LI-02	Revisão final de segurança	A FSR (Revisão final de segurança) é um exame deliberado de todas as atividades de segurança realizadas em um aplicativo de software antes da liberação.	A FSR é realizada pelo consultor de segurança com a ajuda da equipe de desenvolvimento regular e os líderes das equipes de segurança e de privacidade. A FSR não é um exercício de "penetrar e corrigir", nem uma chance de realizar atividades de segurança que foram anteriormente ignorados ou esquecidos.	SDL

Fonte – Autores (2022)

CONSIDERAÇÕES FINAIS

O nosso objetivo de apresentar um roteiro para o desenvolvimento de *software* seguro em aplicações que utilizem do código aberto foi alcançado, ainda que não tenha sido possível testar o nosso produto, para avaliar se ele é funcional no ciclo de desenvolvimento de um *software*. No entanto, podemos apresentar boas práticas já validadas de forma apartada, a fim de mitigar problemas relacionados à qualidade do *software* e possíveis falhas na segurança que durante no processo de desenvolvimento possa a vim gerar.

Foi identificado no decorrer das pesquisas feitas, a ausência de documentos acadêmicos referente a *OpenSSF*, a ferramenta *Fortify* e a abrangência do tema de segurança em desenvolvimento de *software* em trabalhos que abordam o desenvolvimento de *software*.

Existem outros repositórios a serem consultados para levantamento de melhores práticas que não foram abordadas neste documento, porém a *OpenSSF* cita em seu material de orientação aos desenvolvedores *open*-source outros repositórios a serem consultados para o desenvolvimento seguro, como por exemplo: o CWE Top 25 que traz as principais vulnerabilidade publicamente conhecidas (CVE); família ISO 27000; *NIST, PCI, SafeCode, ISC2*, *Google SLSA* 2021, *Cloud Native Computing Foundation (CNCF), BSIMM, Building Security In Maturity Model (BSIMM), Business Software Alliance* (BSA). Diante de tantos repositórios, a escolha de qual será o melhor para o projeto se dará conforme o ecossistema do *software*, sua funcionalidade e aplicabilidade.

Sendo tantas as falhas de segurança em *softwares* e cada vez mais presentes e tão próximas, entendemos os conceitos abordados nessas pesquisas bibliográficas que realizamos, como boas práticas de segurança em desenvolvimento e não específicas para uma linguagem ou plataforma são aplicáveis à diferentes plataformas de *hardware*, sistemas operacionais, metodologias de desenvolvimento e linguagens de programação. De forma a não especificamente garantir que não exista uma falha ou vulnerabilidade, durante o processo de desenvolvimento, mas sim de mitigar ao máximo que exista essa possibilidade.

REFERÊNCIAS

AZEVEDO J, Delmir Peixoto de; CAMPOS, Renato de. **Definição de requisitos de *software* baseada numa arquitetura de modelagem de negócios**. Production, v. 18, p. 26-46, 2008.

BLAZQUEZ, Daniel. **SAST e DAST vs IAST:** tudo o que você precisa saber sobre as ferramentas AST. Hdiv. 2020. Disponível em: https://hdivsecurity.com/bornsecure/sast-dast-vs-iast-all-you-need-to-know-about-ast-tools/ <acessado em 10/05/2022>

DEMARTINI, Felipe. **Malware que rouba dados de cartão estava em biblioteca oficial de programação**. Reportagem – CanalTech. 02/08/2021. Disponível em: https://canaltech.com.br/seguranca/malware-que-rouba-dados-de-cartao-estava-em-biblioteca-oficial-de-programacao-191414/ <acessado em 10/11/2021>

ESPITIA, Diego Samuel. **Usando bibliotecas de desenvolvimento para implantar malware**. Blog Telefónica Tech. 01/12/2022. Disponível em: https://empresasbr.blogthinkbig.com/usando-bibliotecas-desenvolvimento-implantar-malware/ <acessado em 10/11/2021>

ESPOSTE, Arthur de Moura del; BEZERRA, Carlos Felipe Lima. **Cenário de Decisões Baseado em Métricas de *Software:*** Definição e Implementação de Cenários a partir de Métricas de *Design* e de Vulnerabilidade para Tomada de Decisão. Trabalho de Conclusão de Curso – Universidade de Brasília – UnB Faculdade UnB Gama - FGA, 2014.

FERNANDES, Jorge Henrique Cabral; HOLANDA, Maristela Terto. **Segurança No Desenvolvimento De Aplicações.** Desenvolvido em atendimento ao plano de trabalho do Programa de Formação de Especialistas para a Elaboração da Metodologia Brasileira de Gestão de Segurança da Informação e Comunicações – CEGSIC 2009-2011. Disponível em:

https://www.trf3.jus.br/documentos/rget/seguranca/CLRI/GSIC701_Seguranca_Desenvolvimento_Aplicacoes.pdf <acessado em 10/11/2021>

HACK_EDU. **SAST vs DAST vs IAST**. Disponível em: https://www.hackedu.com/blog/sast-vs-dast-vs-iast#:~:text=IAST%20Advantages&text=and%20configuration%20options%20to%20check,of%20a%20vulnerability%2C%20unlike%20DAST. <acessado em 15/05/2022>

HINTZBERGEN, J.; HINTZBERGEN, K.; AMULDERS, A. & BAARS, H. **Fundamentos de Segurança da Informação**: com base na ISO 27001 e na ISO 27002. Brasport Livros e Multimidia Ltda. 2018

IEEE. **Interactive Application Security Testing**. INSPEC Accession Number: 19155730. DOI: 10.1109/ICSGEA.2019.00131. Xiangtan, China, 10-11 Aug. 2019 Disponível em: https://ieeexplore.ieee.org/abstract/document/8901378 <acessado em 07/05/2022>

LEITE, Larissa Menune; LUCRÉDIO, Daniel. **Desenvolvimento de *software* utilizando o framework scrum**: um estudo de caso. Revista TIS, v. 3, n. 2, 2014.

LINUX FOUNDATION. Desenvolvendo Software Seguro (LFD121) disponível em: https://training.linuxfoundation.org/training/developing-secure-software-lfd121/ <acessado em 12/05/2022>

MICROSOFT. **A plataforma de identidade da Microsoft** para desenvolvedores consiste em um serviço de autenticação, bibliotecas de *softwar*e livre e ferramentas de gerenciamento de aplicativos. Disponível em: https://docs.microsoft.com/pt-br/azure/active-directory/develop/ <acessado em 10/11/2021>

_____. **Implantar aplicativos seguros no Azure**. Disponível em: https://docs.microsoft.com/pt-br/azure/security/develop/secure-deploy <acessado em 10/11/2021>

_____. **Ciclo de desenvolvimento seguro** Disponível em: https://docs.microsoft.com/en-us/windows/win32/ <acessado em 10/11/2021>

_____. **Criar aplicativos seguros no Azure**. Disponível em: https://docs.microsoft.com/pt-br/azure/security/develop/secure-design#ask-security-questions <acessado em 10/11/2021>

_____. **Desenvolver aplicativos seguros no Azure**. Disponível em: https://docs.microsoft.com/pt-br/azure/security/develop/secure-develop <acessado em 10/11/2021>

_____. **Práticas de segurança Azure**. Disponível em: https://azure.microsoft.com/pt-br/resources/security-best-practices-for-azure-solutions/ <acessado em 10/11/2021>

_____. **Recursos para desenvolver em conformidades a regulamento**. Disponível em: https://servicetrust.microsoft.com/ViewPage/BlueprintOverview <acessado em 10/11/2021>

_____. **Resumo Microsoft onde consultar sobre desenvolvimento seguro** Disponível em: https://docs.microsoft.com/pt-br/azure/security/develop/secure-dev-overview <acessado em 10/11/2021>

MAGINA, Rodrigo Agape Vieira. **Como reduzir o número de bugs e vulnerabilidades de uma aplicação. Técnicas e ferramentas para desenvolver um *software* seguro e estável**. Trabalho de Conclusão de Curso - Universidade Federal Fluminense. 2017.

Micro Focus. **Fortify Static Code Analyzer** - *Software* Version: 20.2.0. Disponível em: https://www.microfocus.com/documentation/fortify-static-code-analyzer-and-tools/2020/SCA_Guide_20.2.0.pdf <acessado em 16/05/2022>

MILLANI, Luís Felipe Garlet; BECK FILHO, Antonio Carlos Schneider. **Análise de correlação entre métricas de qualidade de *software* e métricas físicas**. Trabalho de conclusão de graduação - Universidade Federal do Rio Grande do Sul. 2013

OLIVEIRA, Fernando Gonçalves de; SEABRA, João Manuel Pimentel. **Metodologias de desenvolvimento de *software***: Uma análise no desenvolvimento de sistemas na web. Tecnologias em projeção, v. 6, n. 1, p. 20-34, 2015.

Open Source Security Foundation (OSSF). Disponível em: https://openssf.org <acessado em 11/05/2022>

_____. **Repositório *GitHub***. Disponível em: https://github.com/ossf <acessado em 11/05/2022>

Open Web Application Security Project (OWASP). Disponível em: https://owasp.org <acessado em 05/05/2022>

_____. **OWASP Top 10:2021** Disponível em: https://owasp.org/Top10/ <acessado em 05/05/2022>

_____. **A01:2021 –** *Broken Access Control.* Disponível em: https://owasp.org/Top10/A01_2021-Broken_Access_Control/ <acessado em 06/05/2022>

_____. **A02:2021 –** *Cryptographic Failures.* Disponível em: https://owasp.org/Top10/A02_2021-Cryptographic_Failures/ <acessado em 06/05/2022>

_____. **A03:2021 –** *Injection.* Disponível em: https://owasp.org/Top10/A03_2021-Injection/ <acessado em 07/05/2022>

_____. **A04:2021 –** *Insecure Design.* Disponível em: https://owasp.org/Top10/A04_2021-Insecure_Design/ <acessado em 07/05/2022>

_____. **A05:2021 –** *Security Misconfiguration.* Disponível em: https://owasp.org/Top10/A05_2021-Security_Misconfiguration/ <acessado em 08/05/2022>

_____. **A06:2021 –** *Vulnerable and Outdated Components.* Disponível em: https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/ <acessado em 08/05/2022>

_____. **A07:2021 –** *Identification and Authentication Failures.* Disponível em: https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/ <acessado em 09/05/2022>

_____. **A08:2021 –** *Software and Data Integrity Failures***.** Disponível em: https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/ <acessado em 09/05/2022>

_____. **A09:2021 –** *Security Logging and Monitoring Failures***.** Disponível em: https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/ <acessado em 10/05/2022>

_____. **A10:2021 –** *Server-Side Request Forgery (SSRF).* Disponível em: https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/ <acessado em 10/05/2022>

ORTEGA, Felipe Delboni; CÂMARA, Carlos Eduardo. **Um Estudo Aplicado a Segurança de Aplicações WEB**. Revista Ubiquidade, ISSN 2236-9031 – v.4, n.2 – jul. a dez. de 2021, p. 85 – 125

PRESSMAN, Roger. **Engenharia de *Software***. Rio de Janeiro, McGraw-Hill Brasil, 2002.

POSITIVE TECHNOLOGIES. **SAST, DAST, IAST, and RASP**: how to choose? Disponível em: https://www.ptsecurity.com/ww-en/analytics/knowledge-base/sast-dast-iast-and-rasp-how-to-choose/ <acessado em 13/05/2022>

REDHAT. **O que é uma API?** Disponível em: https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces <acessado em 29/05/2022>

ROSSINI J, Carlos Roberto; CAMOLESI, Almir Rogério. **O uso de conceitos de injeção de código no desenvolvimento de aplicações adaptativas**. PIBIC (Programas de Iniciação Científica e Tecnológica), pela Fundação Educacional do Município de Assis, 2015

SAMPAIO, Felipe Ferreira. **Uma análise prática das principais vulnerabilidades em aplicações web baseado no top 10 OWASP**. Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Redes de Computadores, Quixadá, 2021.

SLEPVO, Oleg. **DerScanner 3.11.0** - White Paper. Dersecur Ltd. Disponível em: https://www.researchgate.net/publication/359920964_SAST_Static_Application_Security_Testing_is_the_analysis_of_a_source_code_without_its_actual_execution_the_white_box_method_This_is_ideal_for_code_testing_integration_into_the_app_development_process_ <acessado em 15/05/2022>

SOARES, Michel dos Santos. M**etodologias ágeis extreme programming e scrum para o desenvolvimento de *software***. Revista Eletrônica de Sistemas de Informação, v. 3, n. 1, 2004.

SonarQube. **SonarQube Documentation** – Doc 9.4. Disponível em: https://docs.sonarqube.org/latest/ <acessado em 15/05/2022>

SLSA.DEV. **SLSA 101** - Supply chain threats. Disponível em: https://slsa.dev/spec/v0.1/#supply-chain-threats <acessado em 11/05/2022>

SYNOPSYS. **Static Application Security Testing**. Disponível em: https://www.synopsys.com/glossary/what-is-sast.html <acessado em 13/05/2022>

TRAPP, Dayana Fernanda. *Software* para Verificação De Conformidade De Sistemas À Norma Iso/Iec 15408. Trabalho de Conclusão de Curso II - Universidade Regional de Blumenau. 2010. Disponível em: http://campeche.inf.furb.br/tccs/2010-I/TCC2010-1-07-VF-DayanaFTrapp.pdf <acessado em 05/05/2022>

TRAUE, Thiago Graziani; KOBAYASHI, Guiou. **Um estudo sobre a relação entre processos de engenharia de *software* com dependabilidade e previsão de falhas em *software***. Dissertação - Universidade Federal do ABC programa de Pós-Graduação em Engenharia da Informação. 2012. Disponível em: http://biblioteca.ufabc.edu.br/index.php?codigo_sophia=47175 <acessado em 05/05/2022>