UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CÂMPUS TOLEDO
COTSI - CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

GUSTAVO RAÍ MALACARNE

UM ESTUDO COMPARATIVO ENTRE PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE SEGURO.

TRABALHO DE CONCLUSÃO DE CURSO

TOLEDO
2021

GUSTAVO RAÍ MALACARNE

UM ESTUDO COMPARATIVO ENTRE PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE SEGURO.

Trabalho de Conclusão de Curso apresentado ao COTSI - Curso de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná, como requisito parcial para a obtenção do título de Tecnólogo.

Orientador: Prof. Dr. Edson Tavares de Camargo
Universidade Tecnológica Federal do Paraná

TOLEDO
2021

GUSTAVO RAÍ MALACARNE

UM ESTUDO COMPARATIVO ENTRE PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE SEGURO

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Tecnólogo em Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 14 de maio de 2021

_____

Wesley Klewerton Guez Assunção
Doutor em Informática
Universidade Tecnológica Federal do Paraná

_____

Rosane Fátima Passarini
Doutora em Engenharia de Automação e Sistemas
Universidade Tecnológica Federal do Paraná

_____

Edson Tavares de Camargo
Doutor em Informática
Universidade Tecnológica Federal do Paraná

TOLEDO

2021

Dedico este trabalho a minha família e amigos que muito me apoiaram e forneceram o suporte para chegar até aqui. Dedico também aos professores, que, como grandes mestres, sempre me instigaram a nunca parar a busca pelo conhecimento.

AGRADECIMENTOS

Gostaria de agradecer meus pais pela preocupação de me oferecer a melhor educação que estava ao alcance deles. Agradeço também ao colegiado de Tecnologia de Sistemas para Internet pela capacitação e preparação não só para o mercado de trabalho, mas também para as experiências de vida. Um agradecimento especial ao professor Dr. Edson Tavares de Camargo tendo em vista seu imensurável comprometimento e responsabilidade em orientar a minha formação durante o curso.

"A ciência nunca resolve um problema sem criar pelo menos outros dez". (George Bernard Shaw).

RESUMO

MALACARNE, Gustavo. Um estudo comparativo entre processos de desenvolvimento de software seguro.. 2021. 51 f. Trabalho de Conclusão de Curso – COTSI - Curso de Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná. Toledo, 2021.

Segurança da informação é uma área ampla e complexa dentro da computação. No entanto, a preocupação de desenvolvedores em manter os sistemas protegidos contra ameaças externas e internas fez com que a indústria de softwares e pesquisadores voltassem a atenção para que requisitos de segurança fossem incluídos durante o desenvolvimento. Este trabalho tem como objetivo analisar processos de desenvolvimento que incorporam formalmente a segurança da informação e que aparecem mais frequentemente na literatura, para então definir parâmetros e realizar um estudo comparativo entre eles. O estudo selecionou 5 vulnerabilidades listadas no banco de dados CVE e buscou identificar em que momento do ciclo de vida de desenvolvimento de software, com base nas recomendações dos modelos Microsoft SDL e OWASP CLASP, a vulnerabilidade seria evitada ou mitigada.

Palavras-chave: Desenvolvimento seguro. Microsoft SDL. OWASP CLASP.

ABSTRACT

MALACARNE, Gustavo. A comparative study between secure software development processes.. 2021. 51 f. Trabalho de Conclusão de Curso – COTSI - Curso de Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná. Toledo, 2021.

Information security is a large and complex area within computer science. However, the concern of developers to keep systems protected from external and internal threats has caused the software industry and researchers to turn their attention to include security requirements during development. This work aims to analyze development processes that formally incorporate information security and that appear more frequently in the literature, to then define parameters and develop a comparative study between them. This study selected 5 vulnerabilities listed in the CVE database and sought to identify at which point in the software development lifecycle, based on the recommendations of the Microsoft SDL and OWASP CLASP models, the vulnerability would be avoided or mitigated.

Keywords: Secure development. Microsoft SDL. OWASP CLASP.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

APTCA	*Allow Partially Trusted Callers Attribute*
ATL	*Active Template Library*
BSIMM	*Building Security In Maturity Model*
CLASP	*Comprehensive, Lightweight Application Security Process*
COM	*Component Object Model*
COTS	*Commercial Off-The-Shelf*
CSRF	*Cross-site Request Forgery*
CVDS	Ciclo de Vida de Desenvolvimento de Software
CVE	*Common Vulnerabilities and Exposures*
CVSS	*Common Vulnerability Scoring System*
CWE	*Common Weakness Enumeration*
DBO	*Database Owner*
DLL	*Dynamic-link library*
DOM	*Document Object Model*
DoS	*Denial of Service*
DSS	*Data Security Standard*
FSR	*Final Secure Review*
HIPAA	*Health Insurance Portability and Accountability Act*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
HTTPS	*HyperText Transfer Protocol Secure*
IMC	*Integrated Management Controller*
LINQ	*Language Integrated Query*
LOB	*Line-Of-Business*

NTLM	*NT LAN Manager*
NVD	*National Vulnerability Database*
OVA	*Open Virtual Appliance*
PCI	*Payment Card Industry*
RPS	*Relying Party Suite*
RTM	*Release to Manufacturing*
RTW	*Release to Web*
SA	*System Administrator*
SAL	*Standard Annotation Language*
SAMM	*Software Assurance Maturity Model*
SDLC	*Secure Development Lifecycle*
SLA	*Service-Level Agreement*
SQL	*Structured Query Language*
STRIDE	*Spoofing, Tampering, Repudiation, Information disclosure, Denial of service and Elevation of privilege*
UAC	*User Access Control*
URL	*Uniform Resource Locator*
UX	*User Experience*
XML	*Extensible Markup Language*
XSS	*Cross-Site Scripting*

SUMÁRIO

1 INTRODUÇÃO

Devido ao aumento expressivo de vulnerabilidades reportadas em sistemas Web (MITRE, 2019), muitas organizações, públicas e privadas, voltaram seus esforços para prevenir brechas que comprometam seus serviços. Para tanto, iniciou-se uma busca por um processo de desenvolvimento que englobe os requisitos necessários para que uma aplicação esteja segura e garanta os princípios básicos de segurança da informação: Disponibilidade, Integridade, Confidencialidade e Autenticidade.

Uma forma de atender essa demanda é estabelecer práticas de segurança que garantem ao código e aos processos envolvidos no desenvolvimento um nível de segurança mais avançado. Comumente, é muito mais barato criar um software seguro do que corrigir problemas de segurança após a conclusão e implantação dele, sendo esses custos ainda maiores no caso de uma violação de segurança (UTO, 2013).

Desenvolvimento seguro é uma prática para garantir que o código e os processos envolvidos no desenvolvimento de aplicativos sejam o mais seguro possível (ZENAH; AZIZ, 2011). O desenvolvimento seguro envolve a utilização de vários processos, incluindo a implementação de um ciclo de vida de desenvolvimento de segurança (*secure development lifecycle* ou SDLC) e a própria codificação segura. O SDLC é um processo de garantia de segurança de desenvolvimento de software que consiste em práticas de segurança agrupadas em fases nas quais cada uma corresponde a uma fase do ciclo de vida de desenvolvimento de software (CVDS).

É evidente que desenvolver uma aplicação que seja imune a falhas é uma tarefa árdua. No entanto, o uso ponderado da implementação de código é uma forma muito eficaz de retificar as falhas mais críticas que afetam as aplicações web. Ficar atento à validação de dados, usar criptografias complexas, fazer atualizações constantes das ferramentas usadas, não usar componentes vulneráveis, efetuar *backups* regulares e também limitar privilégios aos acessos são regras básicas para proteção.

No trabalho *Desenvolvimento Seguro de Sistemas Web: Uma Revisão Sistemática*, desenvolvi, em parceria com o Parque Tecnológico Itaipu, uma pesquisa científica com intuito de identificar e investigar a área de pesquisa de desenvolvimento de software aliada a segurança da informação. A revisão sistemática visou ressaltar a relevância de pesquisas e ferramentas que proporcionam formas seguras de desenvolver sistemas, buscando evitar as principais vulnerabilidades expostas (MALACARNE; CAMARGO; NOTH, 2019). Dessa forma foi possível destacar trabalhos que discorrem sobre processos de desenvolvimento seguro ou ainda que estabelecem um ciclo de vida de desenvolvimento de software que acopla requisitos de segurança em todas as suas fases. Como principais exemplos, estão citados mais frequentemente na literatura os modelos Microsoft SDL (MICROSOFT, 2012), CLASP (OWASP, 2006), BSIMM (MIGUES; WARE, 2019) e OPEN SAMM (OWASP, 2020).

Os modelos SDL da Microsoft e CLASP da OWASP são líderes no suporte de segurança

no cilo de vida de um software (RHAFFARI; ROUDIèS, 2014). Ambos consistem em um conjunto de atividades organizadas em fases chave, que são associadas ao desenvolvimento de software. No entanto, o SDL segue uma abordagem mais pesada, detalhista e voltada as funcionalidades de uma aplicação. Por outro lado, o CLASP tende a ser mais leve e focado nos papéis de cada profissional envolvido no desenvolvimento. O BSIMM se autodefine como um estudo sobre iniciativas de seguranças já existentes. Esse modelo quantifica as práticas de várias organizações diferentes e mostra um reflexo da segurança dos softwares usados por elas. Já o OPEN SAMM, tem como base as experiências e avaliações de mercado para criar e avaliar a eficácia das atividades. Esses dados passam por um processo de avaliação interna dos times de projeto da OWASP, para então serem incluídos nas sugestões do modelo.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

O objetivo deste trabalho é realizar um estudo comparativo dos processos de desenvolvimento seguros considerados mais populares atualmente. Para tanto, será necessário investigar os processos de desenvolvimento de software que incluem a segurança da informação no desenvolvimentos de sistemas Web e que possuem documentação aberta. Dessa forma, é possível compreender como os métodos analisados de desenvolvimento seguro impactam o produto final de organizações e empresas.

1.1.2 Objetivos Específicos

1. Identificar os processos de desenvolvimento de software seguro mais citados na literatura;

2. Descrever os processos encontrados;

3. Comparar os dois processos que focam no ciclo de desenvolvimento seguro Microsoft SDL e OWASP CLASP;

4. Realizar um estudo da relação dos métodos com vulnerabilidades críticas de aplicações web encontradas no CVE Details.

1.2 JUSTIFICATIVA

As vulnerabilidades reportadas pela comunidade internacional de cibersegurança no relatório do CVE (*Common Vulnerabilities and Exposures*), atingiram um pico relevante em 2017, passando a ser mais que o dobro do ano de 2016 (MITRE, 2019). Sendo muitas delas consideradas críticas, o documento do CVE mostra que elas cresceram significativamente nos últimos 5 anos, passando de zero registrado em 2013 para 2.070 no final do ano retrasado. Em 2018, esse número cresceu ainda mais, enfatizando o risco que o cenário atual está sofrendo.

Aplicações para Web são lançadas em grande número atualmente, visando atender a demanda que a concorrência do mercado exige. O número de aplicativos *mobile* lançados por dia já ultrapassa a marca de 6000 unidades (HASSAN; SHANG; HASSAN, 2017). Porém, nem sempre é possível prever todas as vulnerabilidades no cenário em que serão expostos. Devido a essa velocidade excessiva, processos de segurança nem sempre são adotados no ciclo de vida de desenvolvimento do software.

O SDLC garante que as fases do desenvolvimento do projeto estejam integradas com os requisitos seguros, para fornecer a segurança adequada no sistema ou aplicativo resultante. O SDLC deve ser documentado e as atividades de desenvolvimento do projeto devem estar em conformidade com esse documento. Os padrões e procedimentos escritos para cada fase devem guiar todo o processo, tratando do design, programação, teste, implementação, documentação e manutenção e devem ser flexíveis ao incorporar pontos de verificação de segurança para validar a adequação dos controles no sistema ou aplicativo. De modo geral, o design de segurança dos sistemas ou dispositivos é a chave para evitar vulnerabilidades antes que elas ocorram (OWASP, 2017b).

1.3 DIFERENCIAL TECNOLÓGICO

Este trabalho propõe uma análise dos modelos de Ciclo de Vida de Desenvolvimento Seguro mais usados, de acordo com o artigo *Are Companies Actually Using Secure Development Life Cycles?*: Microsoft SDL, BSIMM e CLASP-SAMM (GEER, 2010), e relaciona esses modelos com vulnerabilidades críticas listadas em relatórios CVE. Todos os modelos fornecem um guia tanto para empresas como para desenvolvedores individuais incluírem segurança ao criarem aplicações desde o início do projeto.

Sistemas Web estão ainda mais vulneráveis se comparados a sistemas *desktop* devido à sua conexão com a rede de Internet. É imprescindível que a indústria de desenvolvimento de software estruture e siga uma metodologia apropriada para criar segurança durante o Ciclo de Vida de Desenvolvimento de Software. A integração de práticas de segurança no ciclo de vida de desenvolvimento de software aliado a verificação da segurança das funcionalidades do aplicativo ajudam a reduzir os riscos de fontes internas e externas.

Portanto a prevenção ajuda a diminuir custos das empresas porque elas não precisam responder a incidentes ou lidar com danos à sua reputação. Incorporar um modelo que adeque requisitos de segurança ao software, como a integração de ferramentas e serviços de testes automatizados ajudam os desenvolvedores a encontrar e corrigir problemas de segurança, prevenindo problemas futuros.

1.4 ORGANIZAÇÃO DO TRABALHO

O trabalho está organizado de forma que o capítulo 2 traz a revisão da literatura com as definições dos termos recorrentes do trabalho, bem como a descrição dos quatro modelos de

desenvolvimento seguro de software selecionados, além dos trabalhos relacionados com esse estudo. O capítulo 3 apresenta a metodologia utilizada e o delineamento da pesquisa, para a coleta dos dados e avaliação das falhas em ambientes de software. Já os capítulos 4 e 5 trazem respectivamente a análise e discussão dos resultados e a conclusão do trabalho.

2 REVISÃO DE LITERATURA

2.1 DEFINIÇÕES

A segurança da informação é o conjunto de atividades que protegem o sistema de informação e os seus dados (KIM; SOLOMON, 2018). O desenvolvimento seguro de software é um processo que padroniza as práticas recomendadas de segurança em produtos e aplicações (MICROSOFT, 2012).

Na literatura, é notável um aumento ao decorrer dos anos da discussão sobre o desenvolvimento seguro de software. Contudo, os aspectos de evolução desse processo não recebem a mesma atenção. Esse fato é evidenciado no trabalho (FELDERER; KATT, 2015), em que os autores apresentam um processo de evolução de segurança para o ciclo de vida de desenvolvimento de software. A abordagem é baseada em um processo de desenvolvimento de segurança que inclui as fases de inicialização, análise de segurança, design de segurança, implementação de segurança, teste de segurança e implantação de segurança.

O ciclo de vida de desenvolvimento de software é definido como o período que começa com a decisão de desenvolver um produto de software e segue até o término de seu uso (IEEE. . . , 1990). Sua finalidade é definir as fases, atividades, entregas e responsabilidades de cada envolvido no processo de desenvolvimento. O que diferencia um processo de software do outro é a ordem em que as fases vão ocorrer, além do cronograma, atividades necessárias e o produto entregue em cada uma delas.

Um ciclo de desenvolvimento seguro de software visa diminuir ou evitar que vulnerabilidades sejam inseridas durante a construção do sistema ao adicionar atividades relacionadas a segurança da informação a cada fase de um ciclo de desenvolvimento de software (UTO, 2013).

A evolução dos artefatos nas diferentes fases do ciclo de vida de desenvolvimento de software está fortemente acoplada. Está afirmação está contida na pesquisa (FELDERER et al., 2015). Esse trabalho do estado da arte mostrou que a evolução da segurança é na verdade uma área de pesquisa fundada a partir da evolução de software e na engenharia de segurança, em que essa necessita ser mais explorada. A pesquisa de evolução de segurança para alguns artefatos do ciclo de vida de desenvolvimento de software ainda é rara e precisa de mais investigação.

O artigo (ZENAH; AZIZ, 2011) discorre sobre a importância de abordar requisitos de segurança em todo o processo de ciclo de vida do desenvolvimento do software e não somente no final ou numa fase específica. A segurança da informação deve ser lembrada por todos os profissionais que trabalham no projeto e até mesmo pelo usuário final. A partir disso, a confiabilidade pode ser aprimorada de forma geral no produto.

(MASOOD; JAVA, 2015) fala como a combinação das análises de vulnerabilidade estática e dinâmica fornecem uma avaliação de risco eficaz. A análise de código estático, por si

só, fornece uma excelente visão geral do sistema e por isso deve ser implementada na fase do CVDS. Outras metodologias como a análise em tempo de execução (análise dinâmica e teste de penetração) fornece um meio eficaz de mapear riscos e dependências. Práticas de segurança padrão como validação de entrada, codificação de saída, criptografia eficaz e modelagem de ameaças provaram ser ainda mais benéficas para o desenvolvimento de um software seguro.

O modelo de maturidade é considerado uma das ferramentas mais eficientes conhecidas para avaliar desempenho. A avaliação de segurança nas organizações deve ser baseada em seu nível de segurança e métricas apropriadas (GHAFFARI; ARABSORKHI, 2018). Portanto essa metodologia avalia cada processo com base nas melhores práticas e eficiência de processo. Por fim, o modelo de maturidade de segurança tenta identificar mecanismos, processos e procedimentos para melhorar a segurança da organização e propor um método de progressão baseado em níveis para atingir a segurança de modo geral.

Sendo assim, desenvolver um software seguro requer a adoção de um ciclo de desenvolvimento seguro de software que padroniza as práticas recomendadas de segurança. Esse processo de aplicar as recomendações de segurança em cada fase do ciclo é descrito como Ciclo de Vida de Desenvolvimento Seguro (neste trabalho sendo referenciado na sigla inglesa SDLC) (KIM; SOLOMON, 2018). As principais metodologias encontradas na literatura que abordam o SDLC serão descritas nas próximas seções.

2.2 OPEN SAMM

O *Software Assurance Maturity Model* (SAMM) é um *framework* de código aberto para ajudar as organizações a formular e implementar uma estratégia de segurança de software adaptada aos riscos específicos que a organização enfrenta. Os recursos fornecidos pelo SAMM ajudarão em: avaliar as práticas existentes de segurança de software em uma organização; construir um programa equilibrado de garantia de segurança de software em iterações bem definidas; demonstrar melhorias concretas em um programa de garantia de segurança; definir e medir atividades relacionadas à segurança em toda a organização.

O modelo SAMM define quatro principais categorias críticas chamadas de "Funções de Negócio". Essas categorias de atividades estão relacionadas aos detalhes do desenvolvimento de software, portanto qualquer organização envolvida no desenvolvimento de um software deve cumprir cada uma dessas Funções de Negócio.

Para cada Função de Negócio, são definidas três práticas de segurança, em que cada uma delas é uma área de atividades relacionadas à segurança que criam garantia para a Função de Negócio relacionada. No geral, existem doze práticas de segurança.

Para cada prática de segurança, existem três níveis de maturidade descritos como objetivos. Cada nível de uma prática de segurança é caracterizado por um objetivo cada vez mais avançado, definido por atividades específicas e métricas de avaliação mais rigorosas do que no nível anterior. Além disso, cada prática de segurança pode ser aprimorada independentemente, embora as atividades relacionadas possam necessitar de melhorias.

Cada uma das doze práticas de segurança possui três níveis de maturidade definidos e um ponto de partida implícito em zero. Os detalhes de cada nível diferem entre as práticas, mas geralmente representam: 0 como ponto de partida implícito que representa as atividades na prática sendo realizadas; 1 para entendimento inicial e fornecimento de práticas de segurança; 2 para aumentar a eficiência e/ou eficácia das práticas de segurança; e 3 para domínio abrangente das práticas de segurança em grande escala.

Nas subseções a seguir, serão detalhadas as quatro Funções de Negócio citadas no OPEN SAMM: Governança, Construção, Verificação e Desdobramento/Desenvolvimento.

2.2.1 Governança

A governança está centrada nos processos e atividades relacionados à forma como uma organização gerencia as atividades gerais de desenvolvimento de software. Mais especificamente, isso inclui preocupações que envolvem grupos de desenvolvimento, bem como processos de negócios estabelecidos no nível da organização.

- Estratégia e métricas: envolve a direção estratégica geral do programa de garantia de software e a instrumentação de processos e atividades para coletar métricas sobre a postura de segurança de uma organização.
- Política e conformidade: envolve a criação de uma estrutura de controle e auditoria de segurança e conformidade em toda a organização para obter maior segurança no software em construção e em operação.
- Educação e orientação: envolve o aumento do conhecimento de segurança entre o pessoal no desenvolvimento de software por meio de treinamento e orientação sobre tópicos de segurança relevantes para as funções individuais do trabalho.

2.2.2 Construção

A construção diz respeito aos processos e atividades relacionados à forma como uma organização define metas e cria o software nos projetos de desenvolvimento. Em geral, isso inclui gerenciamento de produtos, coleta de requisitos, especificação de arquitetura de alto nível, design detalhado e implementação.

- Avaliação de ameaças: envolve identificar e caracterizar com precisão os possíveis ataques ao software de uma organização, a fim de entender melhor os riscos e facilitar o gerenciamento de riscos.
- Requisitos de segurança: envolve a promoção da inclusão de requisitos relacionados à segurança durante o processo de desenvolvimento de software, a fim de especificar a funcionalidade correta desde o início.
- Arquitetura segura: envolve reforçar o processo de design com atividades para promover designs seguros por padrão e controle sobre tecnologias e estruturas nas quais o software é construído.

2.2.3 Verificação

A verificação está focada nos processos e atividades relacionados à forma como uma organização verifica e testa artefatos produzidos durante o desenvolvimento de software. Isso normalmente inclui trabalho de garantia de qualidade, como testes, mas também pode incluir outras atividades de revisão e avaliação.

- Revisão do Projeto: envolve a inspeção dos artefatos criados a partir do processo de design para garantir o fornecimento de mecanismos de segurança adequados e a aderência às expectativas de segurança de uma organização.
- Revisão de código: envolve a avaliação do código-fonte de uma organização para ajudar na descoberta de vulnerabilidades e atividades de mitigação relacionadas, além de estabelecer uma linha de base para expectativas seguras de codificação.
- Teste de segurança: envolve testar o software da organização em seu ambiente de tempo de execução, a fim de descobrir vulnerabilidades e estabelecer um padrão mínimo para versões de software.

2.2.4 Desdobramento/Desenvolvimento

A implantação envolve os processos e atividades relacionados à forma como uma organização gerencia a liberação do software que foi criado. Isso pode envolver o envio de produtos para usuários finais, a implantação de produtos em *hosts* internos ou externos e operações normais de software no ambiente de tempo de execução.

- Gerenciamento de vulnerabilidades: envolve o estabelecimento de processos consistentes para gerenciar relatórios de vulnerabilidades internas e externas para limitar a exposição e coletar dados para aprimorar o programa de garantia de segurança.
- Fortalecimento do ambiente: envolve a implementação de controles para o ambiente operacional em torno do software de uma organização para reforçar a postura de segurança dos aplicativos que foram implantados.
- Ativação Operacional: envolve a identificação e captura de informações relevantes à segurança necessárias para que um operador configure, implante e execute adequadamente o software de uma organização.

2.3 BSIMM10

O BSIMM é o resultado de um estudo de iniciativas de segurança de software do mundo real, pois foi construído diretamente a partir dos dados observados em 122 empresas. O modelo, que está em sua décima interação, compreende um único conjunto de atividades exclusivas, relacionadas a três níveis diferentes de atividades. Dessa forma, é possível distinguir a frequência relativa com a qual as atividades são observadas nas organizações. As atividades observadas com frequência são designadas como "nível 1", as atividades observadas com menos frequência são designadas como "nível 2" e as atividades observadas com pouca frequência são

designadas como "nível 3". No total, apresenta um conjunto de 119 atividades em uma estrutura de segurança de software na qual inclui doze práticas organizadas em quatro categorias, sendo elas: Governança, Inteligência, Interações com o SDLC e Implantação, descritas nas subseções a seguir.

2.3.1 Governança

Engloba as práticas que ajudam a organizar, gerenciar e medir uma iniciativa de segurança de software. O desenvolvimento pessoal também é uma prática de governança.

- Estratégias e métricas: Abrange planejamento, atribuição de funções e responsabilidades, identificação de metas de segurança de software, determinação de orçamentos e identificação de métricas e pontos de verificação.
- Conformidade e política: A prática de conformidade e política concentra-se na identificação de controles para regimes de conformidade, como PCI DSS e HIPAA[1], desenvolvendo controles contratuais, como acordos de nível de serviço (SLAs), para ajudar a controlar o risco de software COTS (*commercial off-the-shelf*), definir políticas de segurança de software organizacional e auditar essa política.
- Formação: O treinamento sempre desempenhou um papel importante na segurança de software, porque os desenvolvedores e arquitetos de software geralmente começam com pouco conhecimento sobre segurança.

2.3.2 Inteligência

Todas as atividades que resultam em conhecimento corporativo usadas na realização de atividades de segurança de software em toda a organização. Incluem orientações proativas de segurança e modelagem de ameaças organizacionais.

- Modelos de ataque: Os modelos de ataque capturam informações usadas para pensar como um invasor: modelagem de ameaças, desenvolvimento e refinamento de casos de abuso, classificação de dados e padrões de ataque específicos de tecnologia.
- Recursos e projeto de segurança: A prática de Recursos e Design de Segurança é responsável por criar padrões de segurança utilizáveis para os principais controles de segurança (atendendo aos padrões definidos na prática de Padrões e Requisitos), criar estruturas de *middleware*[2] para esses controles e criar e publicar diretrizes de segurança proativas.
- Normas e requisitos: A prática de Padrões e Requisitos envolve obter requisitos explícitos de segurança da organização, determinar quais COTS recomendar, criar padrões para os principais controles de segurança (como autenticação, validação de entrada etc.),

[1] Padrão de Segurança de Dados da Indústria de Pagamento com Cartão e Lei de Portabilidade e Responsabilização do Seguro Saúde

[2] Software que fornece serviços para softwares aplicativos além daqueles disponíveis pelo sistema operacional.

criar padrões de segurança para as tecnologias em uso e criar um conselho de revisão de padrões.

2.3.3 Interações com o SDLC

Práticas associadas à análise e garantia de artefatos e processos específicos de desenvolvimento de software. Todas as metodologias de segurança do software incluem essas práticas.

- Análise e arquitetura: A Análise de Arquitetura abrange a captura da arquitetura de software em diagramas concisos, a aplicação de listas de riscos e ameaças, a adoção de um processo para revisão (como STRIDE ou Análise de Riscos de Arquitetura) e a construção de um plano de avaliação e correção para a organização.
- Revisão de código: A prática da Revisão de código inclui o uso de ferramentas de revisão de código, desenvolvimento de regras personalizadas, perfis personalizados para uso da ferramenta por diferentes funções (por exemplo, desenvolvedores versus auditores), análise manual e acompanhamento/medição de resultados.
- Teste de segurança: A prática de Teste de segurança está relacionada ao teste de pré-lançamento, incluindo a integração da segurança nos processos padrão de controle de qualidade. A prática inclui o uso de ferramentas de segurança de caixa preta (incluindo teste de *fuzzing*[3]) como teste de fumaça no controle de qualidade, teste de caixa branca orientada a riscos, aplicação do modelo de ataque e análise de cobertura de código. O teste de segurança se concentra nas vulnerabilidades na construção.

2.3.4 Implantação

Práticas que fazem interface com as organizações tradicionais de segurança de rede e manutenção de software. Problemas de configuração, manutenção e outros problemas de ambiente têm impacto direto na segurança do software.

- Teste de invasão: A prática de teste de invasão envolve testes externos padrão, do tipo realizado por especialistas em segurança. Tais testes se concentram nas vulnerabilidades do software finalizado e fornece um relatório para o gerenciamento e mitigação das falhas.
- Ambiente de software: A prática do ambiente de software lida com o *patch* do SO e da plataforma (inclusive na nuvem), *firewalls* de aplicativos da web, documentação de instalação e configuração, contêiner, orquestração, monitoramento de aplicativos, gerenciamento de alterações e assinatura de código.
- Gerenciamento de configuração e gerenciamento de vulnerabilidade: A prática Gerenciamento de configuração e gerenciamento de vulnerabilidades se preocupa com a aplicação de *patches* e atualizações, controle de versão, rastreamento e correção de defeitos e manipulação de incidentes.

[3]Técnica de testes de software que envolve prover dados inválidos, inesperados e aleatórios como entradas para programas de computador.

2.4 MICROSOFT (SDL)

O Ciclo de Vida do Desenvolvimento Seguro da Microsoft (SDL) é uma iniciativa da empresa para formalizar o processo de garantia de segurança como uma política obrigatória desde 2004, chegando na sua versão 5.2 atualmente. O SDL vem desempenhando um papel muito importante na incorporação de segurança da informação tanto no software quanto na cultura da Microsoft.

O desenvolvimento seguro de software possui três principais elementos: práticas recomendadas, melhorias de processo e métricas. O documento publicado pela empresa se concentra principalmente nos dois primeiros elementos, e as métricas são derivadas da medição de como eles são aplicados.

Para entregar de forma confiável um software mais seguro, é necessário um processo abrangente. Dessa forma, para ajudar a determinar onde os esforços de segurança e privacidade são necessários a Microsoft definiu quatro categorias: Segurança no Design, Segurança Padrão, Segurança na Implantação e Comunicações.

A Segurança no Design trata da segurança na arquitetura, planejamento e estrutura do sistema. Além disso, lida com a modelagem e mitigação de ameaças, eliminação de vulnerabilidades e melhorias na segurança, como evitar o uso de protocolos e códigos descontinuados.

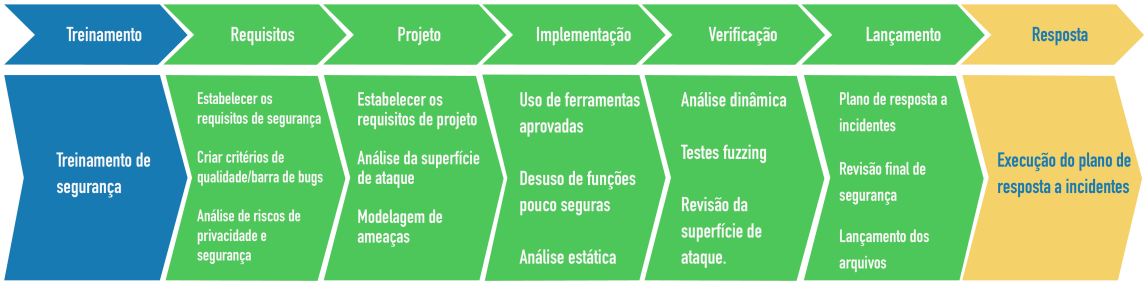
Equipes de desenvolvimento estão a todo momento utilizando ferramentas ou *frameworks* para auxiliá-los durante a construção do software. No entanto, muitos deles apresentam configurações padrão que, se não substituídas, apresentam risco a operação do sistema. Por isso a categoria de Segurança Padrão aborda conceitos como limitar privilégios, defesa em camadas e manter serviços pouco utilizados desativados por padrão.

Na implantação de um software, é necessário fornecer guias que descrevem como implantar cada recurso de um programa com segurança, incluindo o fornecimento de informações aos usuários que permitem avaliar o risco de segurança, ferramentas de análise e gerenciamento, bem como ferramentas para auxiliar na implantação de *patches* de atualização quando necessário. Essas questões são tratadas na Segurança na Implantação.

Por último, o contanto e comunicação com o usuário do sistema deve ser eficaz, oferecendo soluções de segurança sempre que possível. As equipes de desenvolvimento respondem prontamente aos relatórios de vulnerabilidades de segurança e comunicam informações sobre atualizações de segurança. Além disso, é necessário um bom envolvimento com a comunidade. As equipes precisam ser proativas com os usuários para responder a perguntas sobre vulnerabilidades de segurança, atualizações de segurança ou alterações no cenário de segurança.

Depois de adicionar as quatro etapas ao processo de desenvolvimento, o modelo de processo de desenvolvimento de software seguro será similar a Figura 1:

Figura 1 – Fases e etapas que formam o ciclo de vida do desenvolvimento seguro de software da Microsoft.



Fonte: Microsoft (2012)

2.4.1 Requisitos pré-SDL: treinamento em segurança

Educação e conscientização: Todos os membros das equipes de desenvolvimento de software devem receber treinamento apropriado para se manterem informados sobre os princípios básicos de segurança e as tendências recentes em segurança e privacidade. Os indivíduos que desenvolvem programas de software devem participar de pelo menos uma aula de treinamento de segurança a cada ano. O treinamento em segurança pode ajudar a introduzir nos costumes dos desenvolvedores que o software seja criado com segurança e privacidade e também pode ajudar as equipes de desenvolvimento a se manterem atualizadas sobre questões de segurança. Os membros da equipe do projeto devem ser encorajados a buscar educação adicional sobre segurança e privacidade, adequada às suas necessidades ou produtos.

2.4.2 Fase Um: Requisitos

A fase Requisitos do SDL inclui o início do projeto, momento fundamental para considerar a segurança e a privacidade. Também inclui uma análise de custos para determinar se os custos de desenvolvimento e suporte para melhorar a segurança e a privacidade são consistentes com as necessidades dos negócios.

Início do Projeto: A melhor oportunidade para criar software confiável é durante os estágios iniciais de planejamento de um novo lançamento ou uma nova versão, porque as equipes de desenvolvimento podem identificar os principais objetos e integrar segurança e privacidade, minimizando a interrupção de planos e cronogramas.

Análise de custos: Antes de investir tempo em design e implementação, é importante entender os custos e requisitos envolvidos no tratamento de dados para manter a privacidade. Os riscos à privacidade aumentam os custos de desenvolvimento e de suporte, portanto, melhorar a segurança e a privacidade pode ser consistente com as necessidades da empresa.

2.4.3 Fase Dois: Projeto

Nessa fase é criado o plano de como será conduzido o projeto durante o restante do processo de SDL, da implementação, à verificação e ao lançamento. Durante a fase de Design, são estabelecidas as práticas recomendadas a serem seguidas por meio de especificações funcionais e de design e realiza análises de risco para identificar ameaças e vulnerabilidades em seu software.

Estabelecer e seguir as melhores práticas de design: As especificações funcionais precisam descrever os recursos de segurança ou de privacidade diretamente expostos aos usuários, como exigir autenticação do usuário para acessar dados específicos ou consentimento do usuário antes do uso de um recurso de privacidade de alto risco. As especificações do projeto devem descrever como implementar esses recursos e como implementar todas as funcionalidades como recursos seguros. Recursos seguros são definidos como recursos com funcionalidade bem projetada em relação à segurança, como validação rigorosa de todos os dados antes de processá-los ou uso de criptografia robusta. É importante considerar as preocupações de segurança e privacidade com cuidado e antecedência ao projetar recursos e evitar tentativas de adicionar segurança e privacidade perto do final do desenvolvimento de um projeto.

Análise de risco: Durante a fase de design do desenvolvimento, é fundamental revisar cuidadosamente os requisitos e expectativas de segurança e privacidade para identificar preocupações e riscos. É mais eficiente identificar e tratar dessas preocupações e riscos durante a fase de projeto.

Por questões de segurança, a modelagem de ameaças é um processo crítico e sistemático usado para identificar ameaças e vulnerabilidades no software e deve ser concluída nessa fase. Uma equipe não pode criar um software seguro sem compreender os ativos que o projeto está tentando proteger, as ameaças e vulnerabilidades introduzidas pelo projeto e detalhes de como o projeto atenua essas ameaças. A modelagem de ameaças se aplica a todos os produtos e serviços, todos os tipos de código e todas as plataformas.

2.4.4 Fase Três: Implementação

Durante esta fase, cria-se a documentação e as ferramentas que o cliente usa para tomar decisões sobre como implantar seu software com segurança. Para esse fim, a fase de Implementação é quando estabelece as melhores práticas de desenvolvimento para detectar e remover problemas de segurança e privacidade no início do ciclo de desenvolvimento.

Criando documentação e ferramentas para usuários que abordam segurança e privacidade: Cada versão de um programa de software deve estar com seus requisitos de segurança garantidos em sua configuração padrão e na implantação. No entanto, as pessoas usam programas de maneiras diferentes e nem todos usam um programa em sua configuração padrão. Você precisa fornecer aos usuários informações de segurança suficientes para que eles possam tomar decisões sábias sobre como utilizar um programa com segurança. Como a segurança

e a usabilidade podem entrar em conflito, você também precisa educar os usuários sobre as ameaças existentes e o equilíbrio entre risco e funcionalidade ao decidir como implantar e operar programas de software.

Estabelecer e seguir as melhores práticas para o desenvolvimento: Para detectar e remover problemas de segurança no início do ciclo de desenvolvimento, é de importante ajuda estabelecer, comunicar e seguir práticas eficazes para o uso de código seguro. Vários recursos, ferramentas e processos estão disponíveis para ajudar a atingir esse objetivo. Investir tempo e esforço para aplicar práticas eficazes com antecedência ajudará a eliminar problemas de segurança e evitará responder a eles mais tarde, ou até mesmo após o lançamento, quando o custo é muito maior.

2.4.5 Fase Quatro: Verificação

A fase de verificação atua para garantir que o código atenda aos princípios de segurança e privacidade estabelecidos nas fases anteriores. Isso é feito por meio de testes de segurança e privacidade, como também pelo esforço em manter o foco de toda a equipe em atualizações de modelos de ameaças, revisão de código, testes, revisão e edição completa da documentação.

Testes de segurança e privacidade: O teste de segurança aborda a áreas de confidencialidade, integridade e disponibilidade do software e dos dados processados por ele. Essa área inclui todos os recursos e funcionalidades projetados para mitigar ameaças, conforme descrito no modelo de ameaça. Lidar com problemas que possam resultar em vulnerabilidades de segurança. Por exemplo, uma sobrecarga de *buffer* no código que analisa dados pode ser explorada de maneiras que tenham implicações de segurança.

O teste de segurança é importante para o ciclo de vida do desenvolvimento de segurança. Como Michael Howard e David LeBlanc observam, em (HOWARD; LEBLANC, 2002), "Os designers e as especificações podem descrever um design seguro, os desenvolvedores podem ser diligentes e escrever um código seguro, mas é o processo de teste que determina se o produto é seguro no mundo real."

Push de segurança: Um *push* de segurança é o foco de toda a equipe em atualizações de modelos de ameaças, revisão de código, testes e revisão e o complemento da documentação. Essa atividade não substitui a falta de disciplina de segurança. Em vez disso, é um esforço organizado para descobrir as mudanças que podem ter ocorrido durante o desenvolvimento, melhorar a segurança em qualquer código e identificar e corrigir as vulnerabilidades restantes. No entanto, deve-se notar que não é possível criar segurança no software apenas com essa etapa.

Um *push* de segurança ocorre depois que um produto entra no estágio de verificação (código/recurso alcançado concluído). Geralmente começa no momento em que o teste beta é iniciado. Como os resultados do *push* de segurança podem alterar a configuração e o comportamento padrão de um produto, você deve executar uma revisão final do teste beta após a conclusão do *push* de segurança e após a resolução de todos os problemas e alterações

necessárias.

É importante observar que o objetivo de um *push* de segurança é encontrar vulnerabilidades, e não as corrigir. O tempo para corrigir as vulnerabilidades é após a conclusão do envio de segurança.

2.4.6 Fase Cinco: Lançamento

Nesse momento, acontece a preparação do software para o uso público como também prepara a equipe para o que acontece quando o software está nas mãos do usuário. Um dos principais conceitos da fase de lançamento é o planejamento: mapeamento de um plano de ação, caso sejam descobertas vulnerabilidades de segurança ou privacidade em sua versão. Isso também é transferido para o pós-lançamento, em termos de execução de resposta. Para esse fim, é necessária uma revisão final da segurança e uma revisão da privacidade antes do lançamento.

Revisão de Privacidade de Lançamento: Antes de qualquer versão pública (incluindo as versões alfa e beta), atualize o Questionário de privacidade SDL apropriado para quaisquer alterações significativas na privacidade que foram feitas durante a verificação da implementação. Alterações significativas incluem alterar o estilo de consentimento, alterar substancialmente o idioma de um aviso, coletar diferentes tipos de dados e exibir novos comportamentos. Embora os requisitos de privacidade devam ser tratados antes de qualquer liberação pública de código, os requisitos de segurança não precisam ser abordados antes da liberação pública. No entanto, deve-se concluir uma Revisão de segurança final antes do lançamento final.

Planejamento: Qualquer software pode ser lançado com problemas de segurança ou privacidade desconhecidos, apesar dos melhores esforços e intenções. Mesmo os programas sem vulnerabilidades conhecidas no momento do lançamento podem estar sujeitos a novas ameaças que surgem e podem exigir ação. Da mesma forma, aqueles que debatem sobre privacidade podem levantar novas preocupações sobre após o lançamento. Você deve se preparar antes da liberação para responder a possíveis incidentes de segurança e privacidade. Com o planejamento adequado, você poderá resolver muitos dos incidentes que podem ocorrer no curso de operações comerciais normais.

A equipe deve estar preparada para uma exploração de "dia zero" de uma vulnerabilidade, ou seja aquela para a qual não existe uma atualização de segurança. A equipe também deve estar preparada para responder a uma emergência de segurança de software. Criar um plano de resposta a emergências antes da liberação, economizará tempo, dinheiro e frustração quando uma resposta for necessária por motivos de segurança ou privacidade.

Revisão final de segurança e revisão de privacidade: À medida que o final do projeto de desenvolvimento de software se aproxima, precisa-se ter certeza de que o software é seguro o suficiente para ser entregue. A revisão final de segurança (*final security review* ou FSR) ajuda a determinar isso. A equipe de segurança designada ao projeto deve executar o FSR com a ajuda da equipe do produto para garantir que o software cumpra a Revisão final de segurança

e a Revisão de privacidade.

Liberar para fabricação/Liberar para Web: A liberação do software para fabricação (RTM) ou a liberação para a web (RTW) está condicionada à conclusão do processo do Ciclo de vida do desenvolvimento de segurança, conforme definido neste documento. O consultor de segurança designado para a liberação deve certificar que sua equipe cumpriu os requisitos de segurança. Da mesma forma, para todos os produtos que possuem pelo menos um componente com uma classificação de impacto na privacidade de P1, seu consultor de privacidade deve certificar que sua equipe cumpriu os requisitos de privacidade antes que o software possa ser enviado.

2.4.7 Requisito pós-SDL: resposta

Serviço de Segurança e Execução de Respostas: Depois que um programa de software é lançado, a equipe de desenvolvimento do produto deve estar disponível para responder a quaisquer possíveis vulnerabilidades de segurança ou problemas de privacidade que justifiquem uma resposta. Além disso, é necessário um plano de resposta que inclua os preparativos para possíveis problemas pós-lançamento.

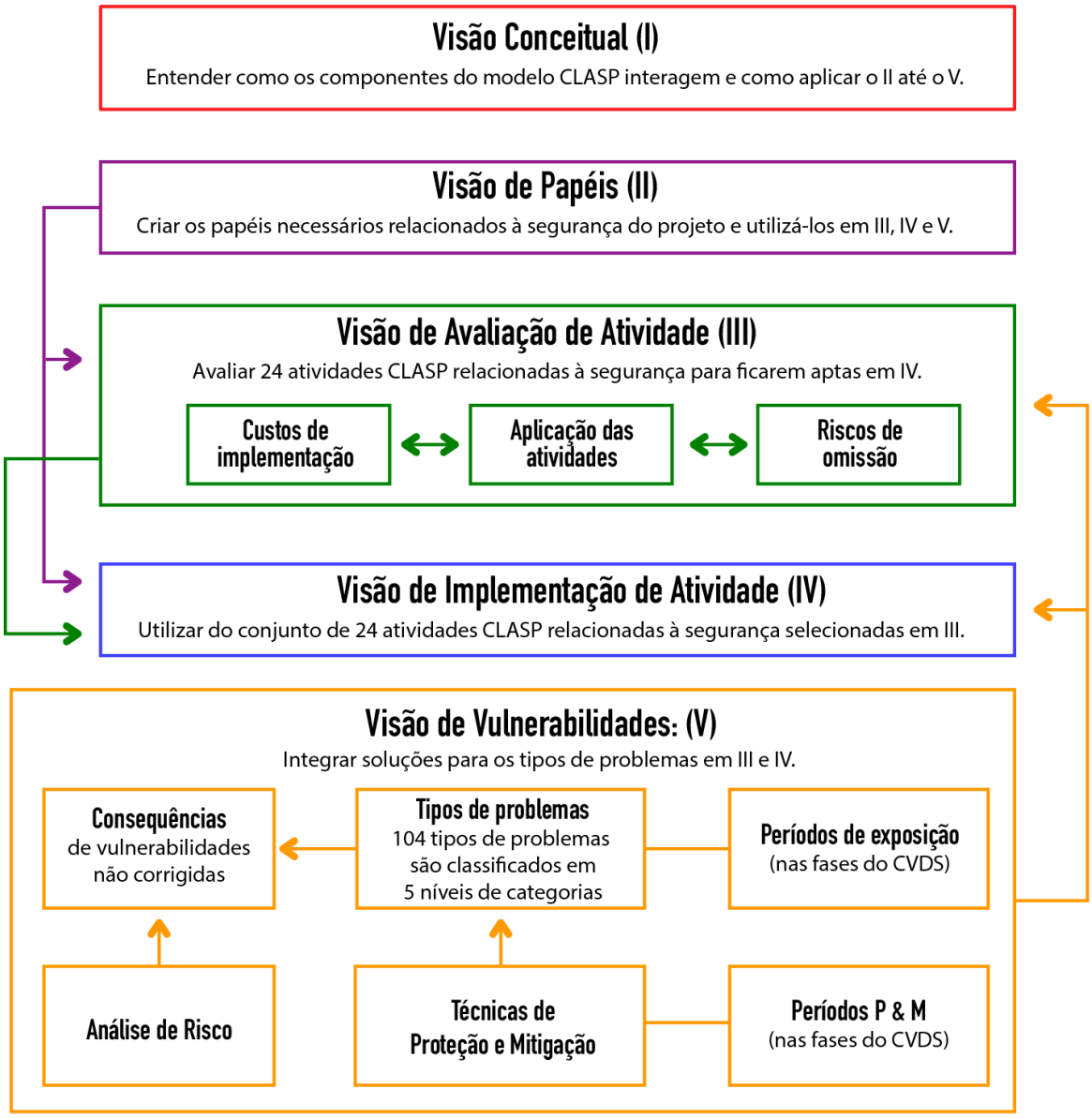
2.5 OWASP CLASP

O CLASP (*Comprehensive, Lightweight Application Security Process*) é uma metodologia de desenvolvimento seguro de software orientada a atividades e aos papéis de colaboração no projeto. Consiste em um conjunto de componentes com as melhores práticas de segurança formalizadas que cobrem todo o CVDS, não apenas o desenvolvimento, de modo que as preocupações com segurança sejam adotadas desde os estágios iniciais do CVDS. Dessa forma, pode ser aplicado em novos projetos ou para os que já estão em andamento, propondo 24 atividades divididas em componentes de processos discretos ligados a um ou mais papéis de um projeto. Tais papéis dependem da organização da equipe do projeto e nas funções na qual cada um tem um conjunto perfeitamente definido de atividades e responsabilidades das quais deve cuidar. Sendo assim, tem como finalidade fornecer um guia para os profissionais envolvidos em um projeto, como gerentes, auditores de segurança, desenvolvedores, arquitetos e testadores.

Além disso, há outros componentes que o CLASP oferece, como recursos que consistem em acesso a artefatos importantes, como exemplo a utilização de ferramentas para automatização do CLASP. Também fornece casos de uso de vulnerabilidades, que consistem na descrição de condições nas quais os serviços de segurança podem se tornar vulneráveis em aplicações.

A estrutura deste processo é dividida em cinco perspectivas, denominadas Visões CLASP. Cada Visão, por sua vez, é dividida em atividades, que contém os componentes do processo. As visões são descritas a seguir.

Figura 2 – Esquema do conjunto de visões do CLASP e as interações entre eles.



Fonte: OWASP (2006)

2.5.1 Visão Conceitual

A Visão Conceitual apresenta uma visão geral de como funciona o processo CLASP e como seus componentes interagem. São definidos os serviços básicos de segurança que devem ser atendidos para cada recurso: autorização, confidencialidade, autenticação, disponibilidade, responsabilidade e não repúdio. Nessa etapa são introduzida as melhores práticas, a interação entre o CLASP e as políticas de segurança, alguns conceitos de segurança e os componentes do processo.

2.5.2 Visão de Papéis (função)

A Visão de Papéis introduz as responsabilidades básicas de cada membro do projeto, mostrando como uma equipe de projeto deve executar questões de segurança, dependendo das responsabilidades específicas de cada função. São citadas as funções de gerente, arquiteto, especificador de requisitos, projetista, implementador, analista de testes e auditor de segurança. A partir disso, cada uma é relacionada com as atividades propostas, além de especificar quais são os requerimentos básicos para que cada função seja desempenhada.

2.5.3 Visão de Avaliação de Atividade

A Visão de Avaliação de Atividades descreve o propósito de cada atividade, bem como os responsáveis, contribuidores, a aplicabilidade, o impacto relativo, os riscos em caso de omissão da atividade, a frequência da atividade e propõe um valor aproximado para a relação trabalho/hora. É nesta etapa que são mapeadas as várias funções com as atividades de processo específicas relacionadas à segurança (há 24 delas) que devem ser implementadas.

2.5.4 Visão de Implementação de Atividade

A Visão de Implementação descreve o conteúdo das 24 atividades de segurança definidas pelo CLASP, detalhando cada uma delas, para então identificar os responsáveis pela implementação, bem como as atividades relacionadas.

2.5.5 Visão de Vulnerabilidades

A Visão de Vulnerabilidades possui um catálogo que descreve 104 tipos de vulnerabilidades no desenvolvimento de software, detalhando as consequências, tipos de problemas, períodos de exposição, técnicas de prevenção e mitigação de vulnerabilidades de segurança. Esta visão considera as categorias das vulnerabilidades, períodos de exposição, consequências, plataformas afetadas, recursos, avaliação de risco, períodos de prevenção e mitigação. Por fim, os 104 tipos são divididos em 5 categorias:

- Erros de Tipo e Limites de Tamanho;
- Problemas do Ambiente;
- Erros de Sincronização e Temporização;
- Erros de Protocolo;
- Erros Lógicos em Geral.

2.6 CONCLUSÃO

Tais metodologias abordam a importância em adotar um ciclo de vida de desenvolvimento seguro de um software, apontando seus benefícios e vantagens no produto final. Além disso, informam sobre ferramentas que podem ser usadas no projeto para garantir que os requi-

sitos de segurança sejam aplicados de forma correta em cada fase. Dessa forma, entender como a segurança é implementada nas fases do ciclo de vida, ajuda a mitigar falhas e vulnerabilidades e prevenir danos futuros. Portanto compreender um ciclo de vida de desenvolvimento seguro, bem como saber utilizar o benefício da aplicação de suas fases, garante um grau maior de qualidade de software.

Tanto BSIMM quanto OPEN SAMM são classificados como modelos de maturidade. No entanto há duas diferenças notáveis entre eles. Primeiramente, BSIMM é apresentado como um modelo descritivo, construído a partir das experiencias em segurança de software de mais de 160 empresas. Seu objetivo é documentar o que realmente é feito dentro dessas empresas por especialistas em segurança, e é devido a isso, que passa por atualizações regulares, sendo um reflexo das mudanças reais que acontecem no desenvolvimento de software. Por outro lado, o OPEN SAMM mostra um modelo prescritivo, apontando o que as empresas deveriam fazer na teoria. Embora construída por especialistas experientes, é uma estrutura genérica baseada em ideias ponderadas. De certa forma, o OPEN SAMM representa uma lista de recomendações de um pequeno grupo de atividades para o desenvolvimento de software.

A outra diferença está na comunidade ativa dos modelos. O BSIMM possui uma comunidade bastante engajada que mantém comunicação direta e conferências globais semestrais. Isso permite que as empresas que medem suas iniciativas com o BSIMM aprendam umas com as outras e colaborem para melhorar suas iniciativas de segurança de software (ISS). Os eventos da comunidade OPEN SAMM são mais raros e se concentram em mudanças incrementais no próprio modelo, não melhorando os recursos das ISS.

Por outro lado, os modelos SDL da Microsoft e o CLASP da OWASP fornecem um amplo conjunto de atividades cobrindo inteiramente o ciclo de vida do desenvolvimento. De acordo com a documentação, a Microsoft está usando o SDL internamente e o CLASP recebe contribuição juntamente com revisão de várias empresas de segurança líderes do consórcio OWASP. Além disso, para uma breve comparação do SDL e CLASP, o primeiro é considerado mais pesado e rigoroso, tornando-o mais adequado para grandes organizações. CLASP, por outro lado, é leve e mais acessível para pequenas organizações com demandas de segurança menos rígidas.

Foi possível observar que ambos os processos passaram por validação, já que são amplamente utilizados na produção de software. Sendo assim, para uma melhor observação de como os modelos lidam com vulnerabilidades comuns em softwares, uma comparação mais minuciosa será feita na discussão dos resultados.

3 METODOLOGIA

A fim de realizar as atividades necessárias à condução da pesquisa, o processo de criação do conhecimento científico empregado definido como mais adequado é a estrutura de paradigma fenomenológico (SORDI, 2017). Dessa forma, o estudo apresenta um contexto estrutural qualitativo, subjetivo e interpretativo.

3.1 DELINEAMENTO DA PESQUISA

Inicialmente será necessário realizar uma revisão dos trabalhos publicados que abordam obras que discorrem sobre processos de desenvolvimento de software seguro como temática principal. Em seguida, será descrito detalhadamente cada padrão considerado relevante, destacando suas características e sua metodologia.

É fundamental estabelecer parâmetros relevantes antes de classificar os processos encontrados para que possa ser feito um estudo comparativo entre eles. Tais parâmetros devem ser definidos com base nas recomendações oferecidas por cada modelo.

Para poder comparar metodologias de desenvolvimento seguro de software, primeiro é necessário entender sua estrutura, bem como suas respectivas fases. Portanto a análise minuciosa de cada fase é importante para definir cada atividade realizada durante o desenvolvimento.

Por fim, a análise dos dados pode ser feita aplicando os parâmetros estabelecidos, sendo possível classificar os processos distintivamente. O resultado poderá servir como base para o desenvolvimento de software incorporando a segurança da informação em suas fases do ciclo de vida para a mitigação de vulnerabilidades.

3.2 COLETA E TRATAMENTO DE DADOS

Devido ao fato de o BSIMM e o OPEN SAMM serem classificados como modelos de maturidade e não apresentarem uma associação bem definida com cada fase de um CVDS, para ter um foco maior no desenvolvimento de software, a comparação será feita entre os processos da Microsoft SDL e OWASP CLASP. Para estabelecer uma comparação entre esses dois modelos, foram avaliados como cada processo responde a uma vulnerabilidade comum em aplicações web. A "OWASP TOP 10" apresenta uma lista com as 10 principais ameaças a que aplicações web estão expostas. Sua última versão lançada foi no ano de 2017, em que listou as 10 ameaças mais comuns, selecionadas, priorizadas e combinadas com estimativas consensuais de exploração, detectabilidade e impacto, de acordo com dados que abrangem vulnerabilidades coletadas em centenas de organizações e mais de 100.000 aplicativos e APIs do mundo real (OWASP, 2017a).

Após identificar as vulnerabilidades que mais aparecem em sistemas web, a próxima etapa foi selecionar casos reportados de falhas em softwares. Para tanto, a busca e coleta

de dados foi efetuada no banco de dados do *Common Vulnerabilities and Exposures Details* (CVE). Nesse banco de dados há registros de vulnerabilidades reportadas e catalogadas, sendo possível buscar por fornecedores, produtos e versões para visualizar falhas que foram expostas relacionadas a eles.

Os dados de vulnerabilidade do CVE são obtidos diretamente do *National Vulnerability Database* (NVD) fornecidos pelo *National Institue of Standards and Technology* (NIST, 2021). As vulnerabilidades são classificadas usando uma palavras-chave e números do *Common Weakness Enumeration* (CWE) (CWE, 2021). Cada CVE reportado possui uma pontuação baseada no *Common Vulnerability Scoring System* (CVSS) (FIRST, 2020) fornecidas pelo NVD. Os dados de vulnerabilidade são atualizados diariamente.

O CVSS é uma iniciativa pública que fornece um modelo para avaliar e quantificar o impacto das vulnerabilidades em um software, de propriedade e gerenciado pela FIRST.Org. Seu objetivo é representar com precisão o impacto das vulnerabilidades em produtos de software. Atualmente o CVSS encontra-se na versão 3.1 e avalia a pontuação gerando um número que representa o nível de gravidade. Comumente são agrupados em 5 grandes intervalos: nenhum quando a pontuação for 0,0; baixo risco para pontuação entre 0,1 e 3,9; médio risco para pontuação entre 4,0 e 6,9; alto risco para pontuação entre 7,0 e 8,9; e risco crítico para pontuação entre 9.0 e 10.0.

As pontuações CVSS são compostos derivados das seguintes três categorias de métricas (FIRST, 2020):

- Base. Este grupo representa as propriedades de uma vulnerabilidade que não mudam ao longo do tempo, como complexidade de acesso, vetor de acesso, grau em que a vulnerabilidade compromete a confidencialidade, integridade e disponibilidade do sistema e requisito de autenticação para o sistema.
- Temporal. Este grupo mede as propriedades de uma vulnerabilidade que muda com o tempo, como a existência de um *patch* oficial ou código de exploração funcional.
- De Ambiente. Este grupo mede as propriedades de uma vulnerabilidade que são representativas dos ambientes de TI dos usuários, como prevalência de sistemas afetados e potencial de perda. Portanto adaptam as severidades Base e Temporal a um ambiente de computação específico, considerando fatores como a presença de mitigações no ambiente.

As métricas básicas representam as características imutáveis da vulnerabilidade, ou seja, propriedades que são constantes ao longo do tempo e entre os sistemas. É composta por outros dois conjuntos de métricas: as métricas de explorabilidade e as métricas de impacto.

As métricas de explorabilidade refletem a facilidade e os meios técnicos pelos quais a vulnerabilidade pode ser explorada. Ou seja, representa características do que é vulnerável (componente vulnerável). As métricas de impacto refletem a consequência direta de uma exploração bem-sucedida e representam a consequência para quem sofre o impacto (componente impactado). Eles produzem uma pontuação no intervalo de 0,0 a 10,0:

- Vetor de ataque: um invasor pode explorar a vulnerabilidade remotamente ou localmente?

- Privilégios necessários: um invasor deve se autenticar no sistema operacional, aplicativo ou serviço após obter acesso ao alvo para explorar a vulnerabilidade? As opções de pontuação são: é obrigatório ou não é obrigatório.
- Complexidade do ataque: é difícil explorar a vulnerabilidade (por exemplo, requer ação do usuário, como clicar em um URL malicioso?) As opções de pontuação são: altas ou baixas.
- Interação com o usuário: é necessário um outro usuário que não seja o invasor para explorar a vulnerabilidade? As opções são: é necessário ou não é necessário.
- Impacto da confidencialidade: qual é o impacto potencial do acesso não autorizado aos dados do sistema? As opções são: nenhuma, parcial ou completa.
- Impacto na integridade: qual é o impacto potencial da modificação ou destruição não autorizada dos arquivos ou dados do sistema? As opções são: nenhuma, parcial ou completa.
- Impacto na disponibilidade: qual é o impacto potencial se o sistema ou os dados não estiverem disponíveis? Novamente, as opções de pontuação são: nenhuma, parcial ou completa.
- Viés de impacto: para qual propriedade (se houver) a pontuação deve ter um peso maior: confidencialidade, integridade ou disponibilidade? Por exemplo, a confidencialidade pode ser o mais importante para software de criptografia e, portanto, o analista de pontuação atribuirá um viés de confidencialidade.

As métricas temporais representam as características de uma vulnerabilidade que podem mudar ao longo do tempo, mas não entre ambientes de usuário. Eles modificam a pontuação básica, diminuindo-a em até um terço:

- Explorabilidade: qual é o estado atual (ou probabilidade) da capacidade de exploração da vulnerabilidade? As opções são: não definido, prova de conceito, funcionais, sem provas ou altas.
- Nível de correção: que nível de controle de mitigação existe atualmente para a vulnerabilidade? As opções de pontuação são: correção oficial, correção temporária, solução alternativa, não definido ou indisponível.
- Relato da confiança: quão confiáveis são os detalhes da vulnerabilidade? As opções não são: confirmadas, razoável, não confirmadas ou desconhecidas.

As métricas ambientais representam as características de uma vulnerabilidade que é relevante e única a um ambiente de usuário. Tais métricas modificam o resultado para gerar uma pontuação final, variando de 0,0 (nenhum sistema afetado) a 10,0 (a maioria dos sistemas afetados com alto risco de danos catastróficos). Essa pontuação é mais importante porque fornece o contexto para vulnerabilidades dentro da organização:

- Requerimentos de segurança: permite que o analista personalize a pontuação CVSS, dependendo da importância do ativo de TI afetado para a organização, medida em termos de confidencialidade, integridade e disponibilidade. As opções são: nenhum, baixo, médio

ou alto.

- Métricas base modificadas: Essas métricas permitem que o analista substitua as métricas básicas individuais com base em características específicas do ambiente de um usuário.
- Potencial de dano colateral: qual é o grau de perda de informações, sistemas ou pessoas? As opções são: nenhum, baixo, médio ou alto.
- Distribuição de alvos: que porcentagem de sistemas a vulnerabilidade pode afetar? As opções são: nenhum, baixo, médio ou alto. Ter três pontuações para cada vulnerabilidade é eficiente e flexível porque todas as organizações podem usar a mesma pontuação básica para uma vulnerabilidade, mas cada organização pode personalizar a pontuação para seu próprio ambiente.

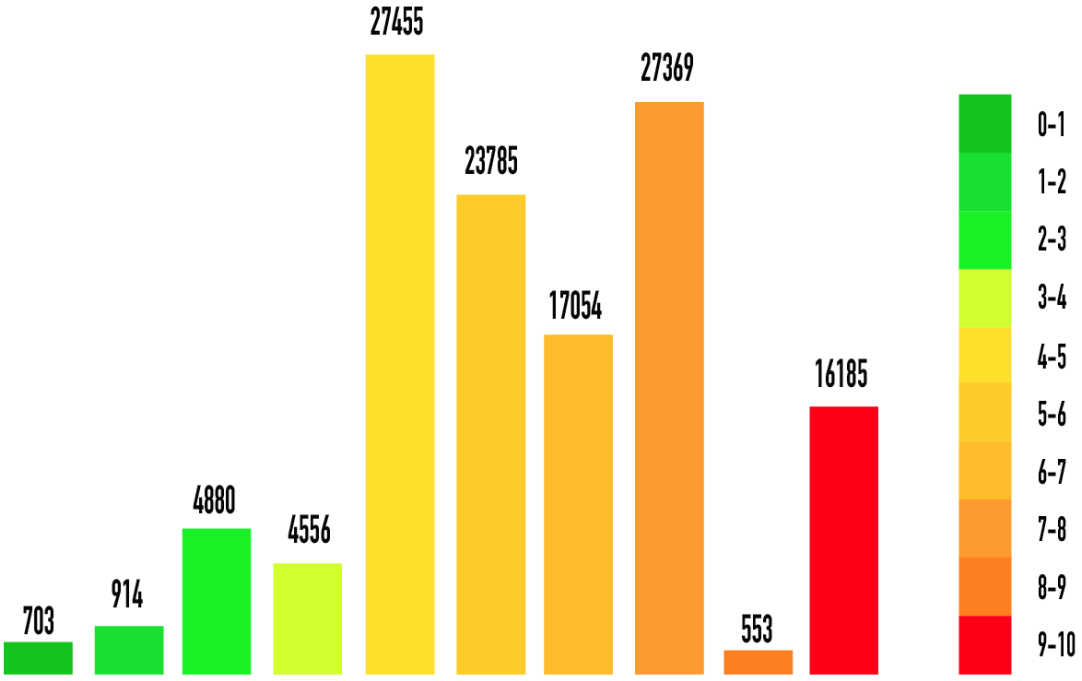
O banco do CVSS apresenta as pontuações a partir do ano de 1999, indo até o ano atual, 2021. No entanto, o último cálculo do total de vulnerabilidades mensuradas foi no ano de 2019. Sendo assim, para esse trabalho, foram escolhidas aleatoriamente cinco vulnerabilidades de sistemas web reportadas no ano de 2019 que continham uma falha comum para esse tipo de sistema e possuíam uma pontuação CVSS de risco crítico ou alto. Até o último ano mensurado, a média da pontuação era de 6,6, sendo 16185 vulnerabilidades de risco crítico, representando 13,10% e 27922 de alto risco, representando 22,60%, como mostram as figuras 3 e 4. O total de vulnerabilidades reportadas até 2019 é de 123454.

Figura 3 – Distribuição de todas as vulnerabilidades (1999-2019) de acordo com a pontuação CVSS em tabela com porcentagem representativa.

Pontuação CVSS	Número de Vulnerabilidades	Porcentagem
0-1	703	0.60
1-2	914	0.70
2-3	4880	4.00
3-4	4556	3.70
4-5	27455	22.20
5-6	23785	19.30
6-7	17054	13.80
7-8	27369	22.20
8-9	553	0.40
9-10	16185	13.10
TOTAL	123454	

Fonte: MITRE (2019)

Figura 4 – Distribuição de todas as vulnerabilidades (1999-2019) de acordo com a pontuação CVSS em gráfico.



Fonte: MITRE (2019)

4 ANÁLISE E DISCUSSÃO DOS RESULTADOS

Os resultados obtidos são apresentados da seguinte forma: inicialmente, uma tabela com os principais dados da CVE é construída para informar sobre os aspectos importantes relacionados a uma vulnerabilidade de software escolhida. Posteriormente, é detalhada a descrição da vulnerabilidade conforme apresentada no CVE, a análise dos impactos (em relação ao pilares de segurança), a complexidade de acesso (nível de conhecimento ou habilidade para explorar a falha), a forma de autenticação necessária para acessar a falha (quando houver) e por último, uma descrição geral da vulnerabilidade apresentada pelo CWE e pelos próprios modelos, quando citadas nestes. Por fim, há a comparação entre os modelos SDL da Microsoft e CLASP da OWASP, especificando como cada processo expõe as fases de introdução da vulnerabilidade no CVDS e quais são as recomendações de segurança para que essa falha não seja incluída durante esse processo. Ambos são exibidos em quadros separados.

Ao comparar, o parâmetro de análise refere-se aos diferentes modos de introdução da vulnerabilidade no desenvolvimento do software. Sendo assim, a comparação fornece informações sobre como e quando essa fraqueza pode ser introduzida em cada fase do CVDS. Geralmente será um ou mais das seguintes: especificação de requisitos, arquitetura e/ou projeto, implementação e, por fim, implantação.

Diferentemente do modelo SDL da Microsoft, o CLASP apresenta a perspectiva da visão de vulnerabilidades. Nessa visão do modelo, são catalogadas 104 falhas em ambientes de software, junto com as consequências de uma exploração bem sucedida, possíveis mitigações, plataformas vulneráveis, recursos de ataque, análise de risco, entre outras informações relevantes. Devido a esta característica, outros quatro parâmetros do modelo, descritos na Tabela 1, são fornecidos para cada CVE, caso a falha seja abordada no modelo.

Tabela 1 – Parâmetros definidos para a Visão de Vulnerabilidades CLASP

1	Trata das consequências possíveis que o modelo pressupõe que ocorra no software caso a vulnerabilidade seja explorada. As consequências estão relacionadas com os pilares disponibilidade, autenticidade, integridade, confidencialidade e não repúdio.
2	Atribui as plataformas principais que correm risco por estarem mais expostas à vulnerabilidade. As plataformas podem ser apresentadas como uma linguagem de programação ou até mesmo um sistema operacional.
3	Especifica as principais mitigações apresentadas pelo modelo para evitar com que a falha seja inserida no software. Cada potencial mitigação está associada a uma fase do CVDS.
4	Traz outras observações e análises relevantes sobre a vulnerabilidade do modelo CLASP.

A seguir, a apresentação da seleção das 5 vulnerabilidades em softwares que estão listadas no CVE. Para cada software, uma breve descrição de sua funcionalidade, a versão indicada como vulnerável (em alguns casos poderá ser mais de uma), o tipo de vulnerabilidade que foi reportada como também sua identificação (ID) na CWE. Ainda em sequencia será

descrita a data de publicação e atualização de cada CVE para, por fim, informar a pontuação CVSS.

4.1 CVE-2019-1871: UNIFIED COMPUTING SYSTEM (CISCO)

O *Cisco Unified Computing System* (Cisco UCS), lançado em 2009, é uma infraestrutura de computação integrada para servidores *datacenters*. Possui gerenciamento intuitivo para automatizar e acelerar a implantação de aplicativos, incluindo virtualização e computação em nuvem, bem como computação de ponta que oferece suporte a locais remotos e grandes quantidades de dados da Internet das Coisas (IoT) (CISCO, 2021). Logo em seguida, na Tabela 2, as informações da vulnerabilidade reportada neste software.

Tabela 2 – Informações CVE-2019-1871

Código de identificação	CVE-2019-1871
Aplicação afetada	Unified Computing System; Versão: 4.0(1c)hs3; Fornecedor: Cisco
Tipo(s) de vulnerabilidade(s)	Negação de serviço (DoS) e estouro de *buffer* (*buffer overflows*)
CWE ID	119 - Restrição imprópria de operações dentro dos limites de um *buffer* de memória.
Data de publicação	21/08/2019
Data da última atualização	09/10/2019
Pontuação CVSS	9,0 (Alto risco)

Fonte: MITRE (2019)

4.1.1 Descrição da vulnerabilidade no software

Uma vulnerabilidade no utilitário de configuração Import Cisco IMC do Cisco *Integrated Management Controller* (IMC) pode permitir que um invasor remoto autenticado cause uma condição de negação de serviço (DoS) e implemente eventuais comandos com privilégios de *root* em um dispositivo afetado. Tal vulnerabilidade ocorre devido à verificação inadequada de limites pelo processo *import-config*. Um invasor pode explorar esta vulnerabilidade enviando pacotes maliciosos a um dispositivo afetado. Quando os pacotes são processados, pode ocorrer uma condição de estouro de *buffer* (*buffer overflow*) explorável. Uma exploração bem-sucedida pode permitir que o invasor implemente um código de sua escolha no dispositivo afetado utilizando de privilégios elevados.

4.1.2 Análise dos impactos

Confidencialidade: Completo (há divulgação total de informações, resultando na revelação de todos os arquivos do sistema). Integridade: Completo (há um comprometimento total da integridade do sistema. Há uma perda total da proteção do sistema, resultando no

comprometimento de todo o sistema). Disponibilidade: Completo (há um desligamento total do recurso afetado. O invasor pode tornar o recurso completamente indisponível).

4.1.3 Complexidade de acesso

Baixa (não existem condições de acesso especializadas ou circunstâncias atenuantes. É necessário muito pouco conhecimento ou habilidade para explorar).

4.1.4 Autenticação

Sistema único (a vulnerabilidade requer que um invasor esteja conectado ao sistema como em uma linha de comando ou por meio de uma sessão de *desktop* ou interface da web).

4.1.5 Descrição geral da vulnerabilidade

Uma condição de estouro de *buffer* ocorre quando um programa tenta colocar mais dados em um *buffer* do que pode conter ou quando um programa tenta colocar dados em uma área da memória após um *buffer*. Nesse caso, um *buffer* é uma seção sequencial de memória alocada para conter qualquer coisa, desde uma *string* de caracteres até um *array* de inteiros. Sendo assim, o invasor utiliza o software para executar operações em um *buffer* de memória, mas pode ler ou gravar em um local de memória que está fora do limite pretendido do *buffer*. Dessa forma o principal recurso afetado é a memória.

Certos idiomas permitem o endereçamento direto de locais de memória e não garantem automaticamente que esses locais sejam válidos para o *buffer* de memória que está sendo referenciado. Isso pode fazer com que as operações de leitura ou gravação sejam realizadas em locais de memória que podem estar associados a outras variáveis, estruturas de dados ou dados internos do programa.

Como resultado, um invasor pode ser capaz de executar código arbitrário, alterar o fluxo de controle pretendido, ler informações confidenciais ou causar o travamento do sistema.

4.1.6 Comparação entre os modelos SDL e CLASP

De acordo com a descrição da vulnerabilidade, o modelo SDL apresenta recomendações de segurança para as fases de projeto (Quadro 1) e implementação (Quadro 2). Já o modelo CLASP, recomenda formas de segurança para as fases do CVDS especificação de requisitos, projeto e implementação (Quadro 3).

Quadro 1 – Fases de introdução do CVE-2019-1871 e recomendações de segurança para o modelo SDL - parte 1.

SDL
Projeto: • Para serviços online, todos os novos lançamentos devem usar o *Relying Party Suite* (RPS) v4.0 SDK[1]. • Controle de conta de usuário (UAC) é um recurso de segurança indispensável. Destina-se a ajudar na transição para o uso regular de privilégios não administrativos por aplicativos clientes. • Cumprir as práticas recomendadas do UAC e garantir que seu aplicativo seja executado com o mínimo de privilégios sempre que possível. • Se os usuários de um programa exigirem uma porta aberta no *firewall*, o código que escuta na porta deve estar em conformidade com certos requisitos de qualidade. • Modelagem de ameaças completas para todas as funcionalidades identificadas durante a fase de análise de custos. • Certifique-se de que todas as modelagens de ameaças atendam aos requisitos mínimos de qualidade do modelo de ameaça. • Ter todas as modelagens de ameaças e mitigações referenciadas revisadas e aprovadas por pelo menos um desenvolvedor, um testador e um gerente de programa. • Confirme se os dados do modelo de ameaça e a documentação associada (especificações funcionais/de projeto) foram armazenados usando o sistema de controle de documentos usado pela equipe do produto.

Fonte: Microsoft (2012)

Quadro 2 – Fases de introdução do CVE-2019-1871 e recomendações de segurança para o modelo SDL - parte 2.

SDL
Implementação: • As equipes de gerenciamento de desenvolvimento, gerenciamento de programa e UX devem se reunir para identificar e discutir quais informações os usuários precisarão para usar o programa de software com segurança. • Disponibilizar informações sobre configurações seguras separadamente ou como parte da documentação padrão do produto e/ou arquivos de ajuda. • Os componentes não devem ter dependências rígidas do protocolo NTLM. Todos os usos explícitos do pacote NTLM para autenticação de rede devem ser substituídos pelo pacote *Negotiate*[2]. • Use o mínimo possível de programas para geração de código e bibliotecas. Para código C/C++ nativo não gerenciado, use Visual C++ 2010, pois ele oferece todos os compiladores e *linker flags* exigidos por SDL. • Use ferramentas de análise de código nas versões atualizadas para código nativo C e C++ ou gerenciado (C#) que estão disponíveis para as plataformas de destino. • Não use APIs com vulnerabilidades. Verifique se não há APIs proibidas no código de envio, incluindo o código de amostra. Os problemas surgem quando um invasor controla o *buffer* de entrada e o código usa dados desse *buffer* para determinar o comprimento máximo do *buffer* a ser copiado. • Requisitos mínimos de versão ATL e codificação COM segura. Os desenvolvedores que usam a *Active Template Library*[3] (ATL) para criar controles COM devem se certificar de que usam as versões de arcabouços de codificações ATL mais seguras. • Cumprir as recomendações mínimas de anotação de código da *Standard Annotation Language* (SAL). Anotar o código ajuda as ferramentas de análise de código existentes a identificar melhor os problemas de implementação e também ajuda a melhorar as ferramentas. • Use aritmética de número inteiro seguro para alocação de memória para o novo código. Todos os novos códigos que usam aritmética para determinar a quantidade de memória dinâmica a ser alocada devem estar protegidos de qualquer forma de estouro, estouro negativo ou truncamento.

Fonte: Microsoft (2012)

Quadro 3 – Fases de introdução do CVE-2019-1871 e recomendações de segurança para o modelo CLASP.

CLASP
Especificação de requisitos: A escolha pode ser feita para usar uma linguagem que não seja suscetível a esses problemas. Projeto: tecnologias atenuantes, como bibliotecas de strings seguras e abstrações de contêiner, podem ser introduzidas. Implementação: Muitos erros lógicos podem levar a essa condição. Pode ser agravado pela falta ou uso indevido de tecnologias de mitigação.

Fonte: OWASP (2006)

A Tabela 3 contém os quatro parâmetros da visão de vulnerabilidades retratada pelo modelo CLASP para o CVE-2019-1871.

Tabela 3 – Parâmetros CLASP para o CVE-2019-1871

Parâmetros	**Visão de Vulnerabilidade CLASP**
1	Disponibilidade: Estouros de *buffer* geralmente levam a travamentos. Outros ataques que levam à falta de disponibilidade são possíveis, incluindo colocar o programa em um *loop* infinito. Autenticidade (Controle de acesso do processamento de instrução): Os estouros de *buffer* muitas vezes podem ser usados para executar código arbitrário, que geralmente está fora do escopo da política de segurança implícita de um programa. Outro: quando a consequência é a execução arbitrária de código, isso pode frequentemente ser usado para subverter qualquer outro serviço de segurança.
2	Linguagens: C, C ++, Fortran, Assembly. Plataformas operacionais: Todas, embora medidas preventivas parciais possam ser implantadas, dependendo do ambiente.
3	Pré-projeto: Use uma linguagem ou compilador que execute a verificação automática de limites (*bounds*). Projeto: Use uma biblioteca de abstração para abstrair APIs arriscadas. Não é uma solução completa. Pré-projeto por meio de *Build*: compiladores baseados em mecanismos *canary*, como StackGuard, ProPolice e o sinalizador Microsoft Visual Studio/GS. A menos que isso forneça verificação automática de limites, não é uma solução completa. Operacional: É recomendado usar a funcionalidade preventiva no nível do sistema operacional. Não é uma solução completa.
4	Estouros de *buffer* são um dos tipos de problemas de segurança mais conhecidos. A melhor solução é a verificação forçada dos limites de tempo de execução do acesso ao *array*, mas muitos programadores C/C ++ consideram que isso é muito custoso ou não possuem a tecnologia disponível. Mesmo esse problema trata apenas de falhas no controle de acesso, já que um acesso fora dos limites ainda é uma condição de exceção e pode levar a um problema de disponibilidade se não for resolvido. Algumas plataformas estão introduzindo tecnologias atenuantes no nível do compilador ou do sistema operacional. Todas essas tecnologias até agora tratam apenas de um subconjunto de problemas de estouro de *buffer* e raramente fornecem proteção completa até mesmo contra esse subconjunto. É mais comum tornar a carga de trabalho de um invasor muito maior, por exemplo, fazendo com que o invasor precise adivinhar um valor desconhecido que muda a cada execução do programa.

Fonte: OWASP (2006)

[1]Servidor que fornece acesso a um aplicativo de software seguro.

[2]O Microsoft Negotiate é um provedor de suporte de segurança (SSP) que atua como uma camada de aplicação entre o *Security Support Provider Interface* (SSPI) e os outros SSPs.

[3]Conjunto de classes C++ baseadas em *templates* desenvolvidas pela Microsoft.

4.2 CVE-2016-10817: CPANEL (CPANEL)

O cPanel é um software de gestão de hospedagem web que possui uma interface gráfica online baseada em Linux (GUI). Usado como um painel de controle para simplificar o gerenciamento de sites e servidores, utiliza a porta TCP 2082 e 2083 para o modo seguro. Entre as suas principais funções, estão a possibilidade de publicar sites, gerenciar domínios, organizar arquivos da web e também criar contas de e-mail (CPANEL, 2021). Logo em seguida, na Tabela 4, as informações da vulnerabilidade reportada neste software.

Tabela 4 – Informações CVE-2016-10817

Código de identificação	CVE-2016-10817
Aplicação afetada	Cpanel; Última versão: 56.0.14 (44 outras versões afetadas); Fornecedor: Cpanel
Tipo(s) de vulnerabilidade(s)	Injeção SQL
CWE ID	89 - Neutralização imprópria de elementos especiais usados em um comando SQL (injeção de SQL)
Data de publicação	01/08/2019
Data da última atualização	05/08/2019
Pontuação CVSS	10,0 (Alto risco)

Fonte: MITRE (2019)

4.2.1 Descrição da vulnerabilidade no software

O cPanel anterior a versão 57.9999.54 permite SQL Injection por meio do arquivo de log ModSecurity TailWatch (SEC-123).

4.2.2 Análise dos impactos

Confidencialidade: Completo (há divulgação total de informações, resultando na revelação de todos os arquivos do sistema). Integridade: Completo (há um comprometimento total da integridade do sistema. Há uma perda total da proteção do sistema, resultando no comprometimento de todo o sistema.) Disponibilidade: Completo (há um desligamento total do recurso afetado. O invasor pode tornar o recurso completamente indisponível).

4.2.3 Complexidade de acesso

Baixa (não existem condições de acesso especializadas ou circunstâncias atenuantes. É necessário muito pouco conhecimento ou habilidade para explorar).

4.2.4 Autenticação

Não necessária (a autenticação não é necessária para explorar a vulnerabilidade).

4.2.5 Descrição geral da vulnerabilidade

Os ataques de injeção de SQL são um tipo de ataque de injeção, em que os comandos SQL são inseridos na entrada de campos de dados para efetuar a execução de comandos SQL predefinidos.

Sem um controle suficiente da sintaxe SQL nas entradas utilizáveis pelo usuário, a consulta SQL gerada pode fazer com que essas entradas sejam interpretadas como SQL em vez de dados comuns do usuário. Isso pode ser usado para alterar a lógica da consulta, para contornar verificações de segurança ou para inserir instruções adicionais que modificam o banco de dados back-end, possivelmente incluindo a execução de comandos do sistema.

A injeção de SQL se tornou um problema comum em sites que possuem banco de dados. A falha é facilmente detectada e explorada. Dessa forma, qualquer site ou software com uma base mínima de usuários provavelmente estará sujeito a uma tentativa de ataque desse tipo. Essa falha é resultante da falta de distinção real entre os planos de controle e de dados do SQL.

4.2.6 Comparação entre os modelos SDL e CLASP

De acordo com a descrição da vulnerabilidade, o modelo SDL apresenta recomendações de segurança para as fases de especificação de requisitos e implementação (Quadro 4). Já o modelo CLASP, também recomenda formas de segurança para as fases do CVDS especificação de requisitos e implementação (Quadro 5).

Quadro 4 – Fases de introdução do CVE-2016-10817 e recomendações de segurança para o modelo SDL.

SDL
Especificação de requisitos: • Certifique-se de que as ferramentas de relatório de falhas possam rastrear problemas de segurança e que um banco de dados contendo todas as falhas de segurança possa ser consultado a qualquer momento. O objetivo dessa consulta é examinar problemas de segurança não corrigidos na revisão final de segurança (FSR). O sistema de rastreamento de falhas do projeto deve armazenar o valor de classificação da barra de *bugs* registrado com cada falha. • Definir e documentar a barra de *bug*[4] de segurança do projeto. Este conjunto de critérios estabelece um nível mínimo de qualidade. Defini-lo no início do projeto melhora a compreensão dos riscos associados a problemas de segurança e permite que as equipes identifiquem e corrijam problemas de segurança durante o desenvolvimento. Uma avaliação de risco de segurança (SRA) é um exercício obrigatório para identificar os aspectos funcionais do software que podem exigir uma análise profunda da segurança. • Quais partes do projeto exigirão teste de invasão (*pentest*) por um grupo organizado que é externo à equipe do projeto. Qualquer parte do projeto que requer um teste de invasão deve resolver os problemas identificados durante o teste antes de ser aprovado para lançamento. Implementação: • Use métodos seguros para acessar bancos de dados. A criação de consultas dinâmicas usando concatenação de strings permite que um invasor execute uma consulta arbitrária por meio da aplicação. Verifique se todo o acesso ao banco de dados é realizado por meio de uma combinação de consultas sequenciais parametrizadas, procedimentos armazenados (ativados por meio de consultas parametrizadas ou LINQ) ou nomes de objeto passados para APIs de armazenamento do Windows Azure que não são baseadas em dados fornecidos pelo usuário. • Evite LINQ (*Language Integrated Query*) "ExecuteQuery". As consultas LINQ são normalmente convertidas pelo compilador/arcabouço em consultas parametrizadas a serem passadas para o banco de dados. No entanto, o método "ExecuteQuery" do LINQ permite que o desenvolvedor passe comandos SQL arbitrários que podem ter sido criados por meio da concatenação de strings. • O banco de dados deve ser acessado por meio de uma conta de privilégio mínimo com apenas o nível mínimo de acesso necessário para realizar as funções da aplicação. Esta conta nunca deve ter funções de administrador do sistema (SA), proprietário do banco de dados (DBO) ou outros privilégios administrativos. • Evite "EXEC" em procedimentos armazenados. Verifique se os procedimentos armazenados não contêm chamadas para "EXEC", "EXECUTE" ou "spexecutesql" (exceto para chamadas para outros procedimentos armazenados). • Para serviços online e/ou aplicativos *line-of-business*[5] (LOB) que acessam um banco de dados SQL, não use consultas SQL com uma finalidade explícita para evitar ataques de injeção de SQL.

Fonte: Microsoft (2012)

Quadro 5 – Fases de introdução do CVE-2016-10817 e recomendações de segurança para o modelo CLASP.

CLASP
Especificação de requisitos: Pode ser escolhido um banco de dados que não use SQL, pois não está sujeito a esta falha. Implementação: Se um banco SQL for usado, todas as falhas que resultam em problemas de injeção de SQL devem ser mitigadas no nível de implementação.

Fonte: OWASP (2006)

A Tabela 5 contém os quatro parâmetros da visão de vulnerabilidades retratada pelo modelo CLASP para o CVE-2019-10817.

Tabela 5 – Parâmetros CLASP para o CVE-2016-10817

Parâmetros	**Visão de Vulnerabilidade CLASP**
1	Confidencialidade: Como os bancos de dados SQL geralmente contêm dados confidenciais, a perda de confidencialidade é um problema frequente com vulnerabilidades de injeção de SQL. Autenticação: Se comandos SQL básicos forem usados para verificar nomes de usuário e senhas, pode ser possível conectar-se a um sistema se passando por outro usuário sem nenhum conhecimento prévio da senha. Autorização: Se as informações de autorização forem mantidas em um banco de dados SQL, pode ser possível alterar essas informações por meio da exploração bem-sucedida de uma vulnerabilidade de injeção de SQL. Integridade: Assim como pode ser possível ler informações confidenciais, também é possível fazer alterações ou mesmo excluir essas informações com um ataque de injeção de SQL.
2	SQL; Qualquer outra plataforma que necessite de interação com um banco de dados SQL.
3	Especificação de requisitos: Quando possível utilizar um banco de dados que não seja SQL; Implementação: Use uma verificação robusta que implemente uma *whitelist* para qualquer entrada do usuário que possa ser usada em um comando SQL. Em vez de evitar os metacaracteres, é mais seguro rejeitá-los totalmente. Isso se deve ao motivo de o uso posterior de dados inseridos no banco de dados pode deixar escapar metacaracteres antes do uso.
4	A injeção de SQL se tornou um problema comum em sites que utilizam banco de dados. A falha é facilmente detectada e explorada e, como tal, qualquer site ou pacote de software com uma base mínima de usuários provavelmente estará sujeito a uma tentativa de ataque desse tipo. Comumente, o ataque é realizado colocando um metacaractere na entrada de dados para, em seguida, colocar os comandos SQL no plano de controle, que não existia lá antes. Essa falha depende do fato de que o SQL não faz distinção real entre os planos de controle e de dados.

Fonte: OWASP (2006)

4.3 CVE-2019-3708: EMC ISILONSD MANAGEMENT SERVER (DELL)

Isilon, fundado em 2001 e adquirido em 2010 pela EMC, é uma plataforma de armazenamento *Network Attached Storage* (NAS) escalável. É composto por um *cluster* de nós independentes integrados usando o sistema operacional OneFS, capaz de oferecer suporte a até 50 petabytes de dados (FENTON, 2018). Logo em seguida, na Tabela 6, as informações da vulnerabilidade reportada neste software.

Tabela 6 – Informações CVE-2019-3708

Código de identificação	CVE-2019-3708
Aplicação afetada	Emc Isilonsd Management Server; Versão: 1.1.0; Fornecedor: Dell
Tipo(s) de vulnerabilidade(s)	Execução de Código e *Cross Site Scripting*
CWE ID	79 - Neutralização imprópria de entrada durante a geração da página da Web (*Cross-Site Scripting*)
Data de publicação	17/04/2019
Data da última atualização	09/10/2019
Pontuação CVSS	9,3 (Alto risco)

Fonte: MITRE (2019)

4.3.1 Descrição da vulnerabilidade no software

O IsilonSD Management Server 1.1.0 contém uma vulnerabilidade *Cross Site Scripting* durante o *upload* de um arquivo OVA. Um invasor remoto pode enganar um usuário administrador para explorar potencialmente esta vulnerabilidade para executar código HTML ou JavaScript malicioso no contexto do usuário administrador.

4.3.2 Análise dos impactos

Confidencialidade: Completo (há divulgação total de informações, resultando na revelação de todos os arquivos do sistema). Integridade: Completa (há um comprometimento total da integridade do sistema. Há uma perda total da proteção do sistema, resultando no comprometimento de todo o sistema.) Disponibilidade: Completa (há um desligamento total do recurso afetado. O invasor pode tornar o recurso completamente indisponível).

4.3.3 Complexidade de acesso

Média (as condições de acesso são em partes mais específicas. Algumas pré-condições devem ser satisfeitas para serem exploradas).

[4] Método de qualidade que é usado para definir os limites de gravidade das vulnerabilidades de segurança.
[5] Um aplicativo LOB faz parte do conjunto de aplicativos que são vitais para o funcionamento de uma empresa.

4.3.4 Autenticação

Não necessária (a autenticação não é necessária para explorar a vulnerabilidade).

4.3.5 Descrição geral da vulnerabilidade

Ataques XSS ocorrem nas situações em que um software não neutraliza ou neutraliza incorretamente entradas controláveis pelo usuário antes de serem colocadas como saída, geralmente como uma página da web.

Esse tipo de vulnerabilidade ocorre quando: Dados não confiáveis entram em um aplicativo da web, normalmente a partir de uma solicitação da web; O aplicativo da web gera dinamicamente uma página que contém esses dados não confiáveis; Durante a geração da página, o aplicativo não impede que os dados contenham conteúdo executável por um navegador da web, como JavaScript, tags HTML, atributos HTML, eventos de mouse, Flash, ActiveX, entre outros; A vítima visita a página da web gerada por meio de um navegador da web, que contém um *script* malicioso que foi injetado usando dados não confiáveis.

Como o *script* vem de uma página da web enviada pelo servidor, o navegador da vítima executa o *script* malicioso no contexto do domínio do servidor da web. Isso efetivamente viola a intenção da política de mesma origem do navegador, que afirma que os *scripts* em um domínio não devem ser capazes de acessar recursos ou executar código de um domínio diferente.

Existem três tipos principais de XSS: XSS refletido (ou não persistente), XSS armazenado (ou persistente) e XSS baseado em DOM. Em muitos casos, o ataque pode ser lançado sem que a vítima perceba. Mesmo com usuários cuidadosos, os invasores frequentemente usam uma variedade de métodos para codificar a parte maliciosa do ataque, como codificação de URL ou Unicode, para que a solicitação pareça menos suspeita.

4.3.6 Comparação entre os modelos SDL e CLASP

De acordo com a descrição da vulnerabilidade, o modelo SDL apresenta recomendações de segurança para as fases de especificação de requisitos e implementação (Quadro 6). Já o modelo CLASP, também recomenda formas de segurança para as fases do CVDS especificação de requisitos e implementação (Quadro 7).

Quadro 6 – Fases de introdução do CVE-2019-3708 e recomendações de segurança para o modelo SDL.

SDL
Implementação: • Os componentes não devem ter dependências rígidas do protocolo NTLM. Todos os usos explícitos do pacote NTLM para autenticação de rede devem ser substituídos pelo pacote *Negotiate*. • Use *cookies* "HTTPOnly"[6] para reduzir o risco de divulgação de informações com um ataque XSS. • Para serviços online e/ou aplicativos LOB, siga os requisitos de validação de entrada de dados e codificação de saída para resolver possíveis vulnerabilidades de XSS. Também use um analisador XML seguro. • Fortaleça ou desative a resolução de entidade XML. O código de análise XML pode ser vulnerável a ataques de expansão de entidade exponencial e ataques de resolução de entidade externa, causando negações de serviço e divulgação de dados confidenciais. Se a resolução de entidade XML não for exigida por seu aplicativo, desative-a. • Use um *cookie* seguro em HTTPS. Os *cookies* HTTP criados por HTTPS não devem ser visíveis para o mesmo site em texto sem criptografia via HTTP. Certifique-se de que todos os *cookies* configurados por HTTPS usam o atributo "seguro". As ferramentas sugeridas incluem: Watcher (Casaba Security).

Fonte: Microsoft (2012)

Quadro 7 – Fases de introdução do CVE-2019-3708 e recomendações de segurança para o modelo CLASP.

CLASP
Implementação: caso haja um local onde usuários compartilhem mensagens públicas e informações, o *cross-site scripting* pode ser impedido apenas no momento da implementação.

Fonte: OWASP (2006)

A Tabela 7 contém os quatro parâmetros da visão de vulnerabilidades retratada pelo modelo CLASP para o CVE-2019-3708.

Tabela 7 – Parâmetros CLASP para o CVE-2019-3708

Parâmetros	**Visão de Vulnerabilidade CLASP**
1	Confidencialidade: o ataque mais comum de *cross-site scripting* envolve a divulgação de informações armazenadas nos *cookies* do usuário. Controle de acesso: em algumas circunstâncias, pode ser possível executar um código arbitrário no computador da vítima quando o *cross-site scripting* é combinado com outras falhas.
2	Qualquer linguagem de programação está sujeita ao ataque; Qualquer plataforma que necessite de interação com um servidor web que suporta conteúdo dinâmico.
3	Implementação: Fazer uso de uma rotina de análise que implementa uma *whitelist*, para garantir que nenhum conteúdo postado contenha *tags* de *script*.
4	Esse tipo de ataque pode ocorrer sempre que um usuário não confiável tiver a capacidade de publicar conteúdo em um site confiável. É comum que um usuário mal-intencionado crie um *script* do lado do cliente, que, quando analisado por um navegador da web, executa alguma atividade (como enviar todos os *cookies* do site para um determinado endereço de e-mail). Se essa entrada não estiver sendo monitorada, este *script* será carregado e executado por cada usuário que visitar o site. Como o site que solicita a execução do *script* tem acesso aos *cookies* em questão, o *script* malicioso também tem. Todos os ataques de *cross-site scripting* são facilmente evitados garantindo-se que nenhuma *tag* de *script*, ou até mesmo nenhuma *tag* HTML, seja permitida nos dados a serem postados publicamente.

Fonte: OWASP (2006)

4.4 CVE-2019-10673: ULTIMATE MEMBER - WORDPRESS (ULTIMATEMEMBER)

Ultimate Member é um *plugin* de perfil de usuário gratuito para WordPress. Facilita a criação de comunidades online e sites de membros no WordPress (MEMBER, 2021). Logo em seguida, na Tabela 8, as informações da vulnerabilidade reportada neste software.

Tabela 8 – Informações CVE-2019-10673

Código de identificação	CVE-2019-10673
Aplicação afetada	Ultimate Member (Wordpress); Última versão: 2.0.24 (214 outras versões afetadas); Fornecedor: Ultimatemember
Tipo(s) de vulnerabilidade(s)	Execução de código e *cross-site request forgery*
CWE ID	352 - *Cross-Site Request Forgery* (CSRF)
Data de publicação	03/04/2019
Data da última atualização	03/04/2019
Pontuação CVSS	9,3 (Alto risco)

Fonte: MITRE (2019)

[6] *Cookies* "HTTPOnly" são inacessíveis para a API JavaScript e são enviados só para o servidor.

4.4.1 Descrição da vulnerabilidade no software

Uma vulnerabilidade CSRF em um formulário de edição de perfil de usuário conectado no *plugin* Ultimate Member antes de 2.0.40 para WordPress permite que os invasores se tornem administradores e subsequentemente extraiam informações confidenciais e executem códigos arbitrários. Isso ocorre porque o invasor pode alterar o endereço de e-mail no perfil do administrador e, em seguida, pode redefinir a senha do administrador usando o formulário "esqueci a senha"do WordPress.

4.4.2 Análise dos impactos

Confidencialidade: Completo (há divulgação total de informações, resultando na revelação de todos os arquivos do sistema). Integridade: Completa (há um comprometimento total da integridade do sistema. Há uma perda total da proteção do sistema, resultando no comprometimento de todo o sistema). Disponibilidade: Completa (há um desligamento total do recurso afetado. O invasor pode tornar o recurso completamente indisponível).

4.4.3 Complexidade de acesso

Média (as condições de acesso são em partes mais específicas. Algumas pré-condições devem ser satisfeitas para serem exploradas).

4.4.4 Autenticação

Não necessária (a autenticação não é necessária para explorar a vulnerabilidade).

4.4.5 Descrição geral da vulnerabilidade

Vulnerabilidades CSRF ocorrem quando uma aplicação web não verifica ou não pode verificar suficientemente se uma solicitação válida e consistente foi fornecida intencionalmente pelo usuário que enviou a solicitação. Quando um servidor web é projetado para receber uma solicitação de um cliente sem qualquer mecanismo para verificar se foi enviada intencionalmente, pode ser possível que um invasor engane um cliente para fazer uma solicitação não intencional ao servidor, que será tratada como um pedido autêntico. Isso pode ser feito por meio de uma URL, carregamento de imagem, XMLHttpRequest, entre outras formas. Tem como resultado a exposição de dados ou execução de código não intencional.

4.4.6 Comparação entre os modelos SDL e CLASP

De acordo com a descrição da vulnerabilidade, o modelo SDL apresenta recomendações de segurança para as fases de projeto e implementação (Quadro 8). Não há no modelo CLASP qualquer menção da vulnerabilidade CSRF. Portanto não será especificada as fases de introdução

da vulnerabilidade reportada no CVE-2019-10673 e recomendações de segurança desse modelo. Também não será possível exibir a tabela com a visão de vulnerabilidades do modelo CLASP.

Quadro 8 – Fases de introdução do CVE-2019-10673 e recomendações de segurança para o modelo SDL.

SDL
Projeto: • Modelagem de ameaças completas para todas as funcionalidades identificadas durante a fase de análise de custos. • Certifique-se de que todas as modelagens de ameaças atendam aos requisitos mínimos de qualidade do modelo de ameaça. • Ter todas as modelagens de ameaças e mitigações referenciadas revisadas e aprovadas por pelo menos um desenvolvedor, um testador e um gerente de programa. • Confirme se os dados do modelo de ameaça e a documentação associada (especificações funcionais / de design) foram armazenados usando o sistema de controle de documentos usado pela equipe do produto. Implementação: • Reduza a falsificação de solicitação entre sites (CSRF). A falsificação de solicitação entre sites é um tipo de ataque em que um site malicioso explora o navegador do usuário para enviar comandos a outro site. Este ataque explora a confiança do site da vítima no navegador do usuário, geralmente usando *cookies* ou sessão do usuário. • Um *token* secundário é enviado com cada solicitação. Idealmente, o *token* deve ser exclusivo por usuário, embora *tokens* exclusivos de sessão sejam suficientes. • Os *tokens* de validação não são enviados automaticamente (como por meio de *cookie*). Incluir um valor de entrada oculto em um POST é o método mais recomendado. • O *token* não é previsível. Se o invasor puder prever o *token*, ele não está protegido. • Carregar DLLs com segurança. Aplicativos e bibliotecas de vínculo dinâmico (DLLs) devem carregar DLLs com segurança para evitar vulnerabilidades de pré-carregamento de DLL. • Defesa de retransmissão ou reflexão e autenticação. Esta nova recomendação foi projetada para ajudar a combater kits de ferramentas sofisticados que estão disponíveis para implementar ataques de reflexão e retransmissão. Espera-se que ajude os desenvolvedores a utilizar as defesas disponíveis contra ataques de retransmissão de reflexão e autenticação. • Redirecionamento seguro, apenas online. O redirecionamento automático do usuário (por meio de "Response.Redirect", por exemplo) para qualquer local arbitrário especificado na solicitação (como um parâmetro de *string* de consulta) pode abrir o usuário a ataques de *phishing*. Portanto, é recomendável não permitir redirecionamentos HTTP para domínios arbitrários definidos pelo usuário.

Fonte: Microsoft (2012)

4.5 CVE-2019-3974: NESSUS (TENABLE)

Nessus é uma ferramenta de verificação de falhas e vulnerabilidades de segurança em softwares. Construído com base na arquitetura cliente/servidor, implementa a parte de

verificação (*scanning*) pelo servidor (TENABLE, 2021). Logo em seguida, na Tabela 9, as informações da vulnerabilidade reportada neste software.

Tabela 9 – Informações CVE-2019-3974

Código de identificação	CVE-2019-3974
Aplicação afetada	Nessus; Versão: 8.5.2; Fornecedor: Tenable
Tipo(s) de vulnerabilidade(s)	Negação de serviço
CWE ID	284 - Controle de acesso inapropriado
Data de publicação	15/08/2019
Data da última atualização	27/08/2019
Pontuação CVSS	8,5 (Alto risco)

Fonte: MITRE (2019)

4.5.1 Descrição da vulnerabilidade no software

Foram encontrados no Nessus 8.5.2 e anteriores em plataformas Windows problemas em que determinados arquivos do sistema podiam ser substituídos arbitrariamente, criando potencialmente uma condição de negação de serviço.

4.5.2 Análise dos impactos

Confidencialidade: Nenhum (não há impacto na confidencialidade do sistema). Integridade: Completa (há um comprometimento total da integridade do sistema. Há uma perda total da proteção do sistema, resultando no comprometimento de todo o sistema). Disponibilidade: Completa (há um desligamento total do recurso afetado. O invasor pode tornar o recurso completamente indisponível).

4.5.3 Complexidade de acesso

Baixa (não existem condições de acesso especializadas ou circunstâncias atenuantes. É necessário muito pouco conhecimento ou habilidade para explorar).

4.5.4 Autenticação

Sistema único (a vulnerabilidade requer que um invasor esteja conectado ao sistema (como em uma linha de comando ou por meio de uma sessão de desktop ou interface da web).

4.5.5 Descrição geral da vulnerabilidade

Ataques DoS são qualquer ataque que afete a disponibilidade de um serviço. Falhas de confiabilidade que fazem um serviço travar ou entrar em algum tipo de estado vegetativo também são problemas potenciais de negação de serviço.

Essa vulnerabilidade se deve principalmente quando um software não restringe ou restringe incorretamente o acesso a um recurso de um usuário não autorizado. O controle de acesso envolve o uso de vários mecanismos de proteção, tais como: Autenticação (comprovando a identidade de um usuário) Autorização (garantindo que um determinado usuário possa acessar um recurso), e Prestação de contas (acompanhamento das atividades realizadas).

Quando qualquer mecanismo não é aplicado ou falha de outra forma, os invasores podem comprometer a segurança do software, ganhando privilégios, lendo informações confidenciais, executando comandos, evitando a detecção, e até mesmo levar o sistema à indisponibilidade.

4.5.6 Comparação entre os modelos SDL e CLASP

De acordo com a descrição da vulnerabilidade, o modelo SDL apresenta recomendações de segurança para as fases de projeto e implementação (Quadro 9). Já o modelo CLASP, também recomenda formas de segurança para as fases do CVDS projeto e implementação (Quadro 10).

Quadro 9 – Fases de introdução do CVE-2019-3974 e recomendações de segurança para o modelo SDL.

SDL
Projeto: • O "AllowPartiallyTrustedCallersAttribute"[7] (APTCA) permite que um *assembly* seja chamado por código não confiável. O comportamento do APTCA é diferente dependendo da versão da estrutura, mas o efeito geral é o mesmo: a funcionalidade potencialmente perigosa está exposta a um código parcialmente confiável. • Para serviços online, todos os novos lançamentos devem usar o *Relying Party Suite* (RPS) v4.0 SDK. • Controle de conta de usuário (UAC) é um recurso de segurança indispensável. Destina-se a ajudar na transição para o uso regular de privilégios não administrativos por aplicativos clientes. • Cumprir as práticas recomendadas do UAC e garantir que seu aplicativo seja executado com o mínimo de privilégios sempre que possível. Implementação: • As equipes de gerenciamento de desenvolvimento, gerenciamento de programa e UX devem se reunir para identificar e discutir quais informações os usuários precisarão para usar o programa de software com segurança. • Fortaleça ou desative a resolução de entidade XML. O código de análise XML pode ser vulnerável a ataques de expansão de entidade exponencial e ataques de resolução de entidade externa, causando negações de serviço e divulgação de dados confidenciais. • Corrija problemas identificados por ferramentas de análise de código para código gerenciado. Execute a ferramenta de análise de código FxCop em todo o código gerenciado e corrija todas as violações das regras de "Segurança" para a versão do FXCop usada. • Use o mínimo possível de programas para geração de código e bibliotecas. Para código C /C++ nativo não gerenciado, use Visual C++ 2010, pois ele oferece todos os compiladores e *linker flags* exigidos por SDL. • Use ferramentas de análise de código nas versões atualizadas para código nativo C e C++ ou gerenciado (C#) que estão disponíveis para as plataformas de destino. • Não use APIs com vulnerabilidades. Os problemas surgem quando um invasor controla o *buffer* de entrada e o código usa dados do *buffer* para determinar o comprimento máximo do *buffer* a ser copiado. Verifique se não há APIs proibidas no código de envio, incluindo o código de amostra.

Fonte: Microsoft (2012)

Quadro 10 – Fases de introdução do CVE-2019-3974 e recomendações de segurança para o modelo CLASP.

CLASP
Projeto: Problemas na arquitetura do sistema e no protocolo do projeto podem tornar os sistemas mais sujeitos a ataques de exaustão de recursos. Implementação: A falta de atenção na programação de baixo nível contribui para o problema.

Fonte: OWASP (2006)

A Tabela 10 contém os quatro parâmetros da visão de vulnerabilidades retratada pelo modelo CLASP para o CVE-2019-3974.

Tabela 10 – Parâmetros CLASP para o CVE-2019-3974

Parâmetros	Visão de Vulnerabilidade CLASP
1	Disponibilidade: o resultado mais comum da exaustão de recursos é a negação de serviço. Controle de acesso: Em alguns casos, pode ser possível forçar um sistema a "expor" outras falhas no caso de esgotamento de recursos.
2	Todas as plataformas e linguagens estão sujeitas a esta vulnerabilidade.
3	Projeto: projete mecanismos de limitação na arquitetura do sistema. Projeto: Certifique-se de que os protocolos tenham limites específicos de escala colocados neles. Implementação: Certifique-se de que todas as falhas na alocação de recursos colocam o sistema em uma posição segura. Implementação: implemente uma falha de segurança no caso de ocorrer uma exaustão de recursos.
4	Os problemas de exaustão de recursos são geralmente facilmente compreendidos, mas são muito mais difíceis de prevenir com sucesso. Os recursos podem ser explorados simplesmente garantindo que a máquina de destino deve fazer muito mais trabalho e consumir mais recursos para atender a uma solicitação do que o invasor deve fazer para iniciar uma solicitação. A prevenção desses ataques requer que o sistema de destino: • reconheça o ataque e negue a esse usuário acesso adicional por um determinado período de tempo; • ou limita uniformemente todas as solicitações para dificultar o consumo de recursos mais rapidamente do que eles possam ser liberados novamente.

Fonte: OWASP (2006)

4.6 DISCUSSÃO SOBRE OS RESULTADOS

É pertinente destacar que existem várias diferenças entre as duas metodologias. Primeiramente, o CLASP tem um foco maior nas métricas de segurança, ajudando a encontrar pontos fracos no projeto. Em geral, possui uma estrutura mais limitada, porém com atividades bem especificadas. O SDL, por sua vez, enfatiza a importância do suporte organizacional para o desenvolvimento de software seguro, definindo uma série de funções para membros ou equipes e como eles devem interagir. Essas funções, quando aplicadas corretamente, ajudam a evitar falhas através de um trabalho conjunto.

Ambos os modelos tentam trazer requerimentos e recomendações para todo o desenvolvimento de software, embora abordem esses pontos de forma distinta no decorrer do processo. O CLASP define papéis individualmente e mais precisamente para os membros de

[7] Permite que um assembly seja chamado por código parcialmente confiável. Sem esta declaração, somente os chamadores totalmente confiáveis podem usar o assembly.

uma equipe. Já o SDL explica em detalhes a interação dos papéis em um projeto, organizando sequencialmente em fases. A questão é que, no geral, o foco do modelo da Microsoft é a qualidade de software, enquanto o modelo da OWASP foca estritamente em segurança.

5 CONCLUSÃO

É possível observar um aumento nos estudos da área de segurança da informação, sendo em grande parte, um reflexo dos desafios de empresas e organizações em manter a confidencialidade, integridade e disponibilidade. No entanto, considerando o número de ataques, vale lembrar que apenas um pequeno número de incidentes de segurança é relatado. Dessa forma, isso pode levar a uma falsa sensação de segurança, já que é difícil calcular a probabilidade de sofrer um ataque cibernético.

Para contornar as consequências, por vezes catastróficas, da exploração de uma vulnerabilidade de uma aplicação, utilizar de processos de desenvolvimento seguro de software ajuda a abordar a segurança desde o inicio e mais frequentemente na elaboração de um software.

Sendo assim, este trabalho buscou analisar metodologias de desenvolvimento de software seguro que são reiteradamente citadas em trabalhos científicos. Através da busca foi possível notar que o BSIMM, OPEN SAMM, SDL e CLASP são relevantes para a segurança da informação. Dessas quatro metodologias, as duas primeiras são classificadas como modelos de maturidade, enquanto as duas últimas se caracterizam como processos de desenvolvimento seguro com foco no CVDS.

Em vista disso, o objetivo principal desta pesquisa foi comparar o OWASP CLASP e Microsoft SDL para encontrar as formas de suporte e as vantagens ao desenvolver um software utilizando as recomendações de segurança descritas neles. Para tanto, 5 vulnerabilidades que são comuns em sistemas web e que foram reportadas no CVE foram escolhidas para observar como é a abordagem de cada processo frente a uma falha.

O CLASP é um processo leve para a construção de software seguro. Inclui um conjunto de 24 atividades bem elaboradas e recursos adicionais, que podem ser ajustados para o processo de desenvolvimento. Além disso, define os papéis e funções da equipe envolvida no desenvolvimento, fortalecendo a postura de segurança do produto de software ao atribuir atividades a cada função.

Um recurso bastante relevante do CLASP é a lista de 104 vulnerabilidades de segurança conhecidas no código-fonte de uma aplicação. Essa lista pode servir de guia ao ser usada como parâmetro durante as revisões de código, prevenindo falhas críticas e a presença de vulnerabilidades no software. No entanto, como mostrado nos resultados, a lista não cobre todas as vulnerabilidades comuns atuais, estando defasada para os riscos que ameaçam aplicações web modernas.

Em contrapartida, o SDL traz um conjunto de atividades voltadas para tratar de questões de segurança que complementam o processo de desenvolvimento, estando mais atualizado para o cenário da tecnologia atual. O objetivo principal do SDL é aumentar a qualidade do software focando em suas funcionalidades, melhorando sua postura de segurança.

O processo SDL é bem organizado e as atividades relacionadas são agrupadas em

fases. Embora essas fases sejam específicas da segurança, é simples associá-las com as fases padrão de desenvolvimento de software. Além disso, várias atividades têm um caráter contínuo no processo SDL, incluindo modelagem de ameaças e o aprendizado de segurança.

Em suma, a capacidade de adaptação de processos de desenvolvimento ao CLASP é mais simples. Isso se deve principalmente ao guia oferecido pelo modelo aos participantes do projeto, citando formas de trabalho e melhorias incrementais. Também há um amplo dicionário de vulnerabilidades que ajuda equipes de desenvolvimento a evitar erros de projeto ou de código que poderiam levar a falhas de segurança.

Não obstante, o modelo da Microsoft faz um bom trabalho ao especificar o método que deve ser usado para executar as atividades, que, na maioria dos casos, são concretas e muitas vezes um tanto objetivas. Mesmo sem uma visão focada em vulnerabilidades, o SDL compensa com uma série de requerimentos e recomendações de segurança. Contudo, é importante destacar que seguir a risca todas as etapas de cada proposta do modelo leva ao efeito de criar uma linha do tempo rígida, geralmente não adequada em ambientes ágeis.

5.1 TRABALHOS FUTUROS

Esse estudo comparou apenas duas das metodologias de segurança mais citadas na literatura: o Microsoft SDL e o OWASP CLASP. Além disso, para uma próxima pesquisa, seria relevante colocar sob essa visão comparativa os modelos de maturidade BSIMM e OPEN SAMM. Embora similares, possuem abordagens distintas na aplicação de segurança em um software. Outro fator importante para levantar em um estudo posterior é como ambos os modelos de maturidade atuam na mitigação e detecção de falhas exploráveis durante e/ou após o desenvolvimento de software.

5.2 CONSIDERAÇÕES FINAIS

Analisando modelos de desenvolvimento seguro, fica claro a vantagem de incluir os métodos neles citados para garantir a proteção de uma aplicação web. Independente do modelo escolhido para ser aplicado no desenvolvimento, a segurança não pode ser um trabalho de apenas um profissional, mas sim um objetivo de toda a equipe. Em vista disso, é mister começar a criar uma mentalidade de segurança nas equipes envolvidas no desenvolvimento de software através de treinamentos de segurança e programas de conscientização. O fator humano deve ser contabilizado para que se possa aprimorar métodos de avaliação de riscos de segurança. Para isso, compreender os dados probabilísticos e a apuração da opinião de especialistas são desafios a serem superados pelas organizações públicas e privadas.

Sendo o SDLC um processo que padroniza as melhores práticas de segurança, ele ajudará a integrar os requisitos de segurança em todo seu ciclo de desenvolvimento, o que resultará naturalmente em um produto seguro por padrão. Dessa forma, para compreender a implementação do desenvolvimento seguro, é necessário estabelecer uma metodologia das

formas de segurança, implementando a estrutura mais adequada ao software. Como resultado, a segurança não pode ser separada do desenvolvimento de um software.

Referências

CISCO. **Cisco Unified Computing System Solution Overview**. 2021. Disponível em: <https://www.cisco.com/c/en/us/products/collateral/servers-unified-computing/solution-overview-c22-744677.html>. Acesso em: 23 de março de 2021. Citado na página 27.

CPANEL. **cPanel Technical Manuals**. 2021. Disponível em: <https://docs.cpanel.net/cpanel/>. Acesso em: 16 de março de 2021. Citado na página 32.

CWE. **A community-developed list of common software and hardware weakness types**. 2021. Disponível em: <https://cwe.mitre.org/about/index.html>. Acesso em: 21 de março de 2021. Citado na página 21.

FELDERER, M.; KATT, B. A process for mastering security evolution in the development lifecycle. **International Journal on Software Tools for Technology Transfer**, v. 17, n. 3, p. 245–250, Jun 2015. ISSN 1433-2787. Disponível em: <https://doi.org/10.1007/s10009-015-0371-4>. Acesso em: 09 de novembro de 2020. Citado na página 5.

FELDERER, M. et al. Evolution of security engineering artifacts: A state of the art survey. **International Journal of Secure Software Engineering**, v. 5, p. 48–98, 01 2015. Citado na página 5.

FENTON, T. **A Closer Look at the Dell EMC Isilon NAS Storage Platform**. 2018. Disponível em: <https://virtualizationreview.com/articles/2018/05/15/dell-emc-isilon-nas-storage-platform.aspx>. Acesso em: 16 de março de 2021. Citado na página 36.

FIRST. **Common Vulnerability Scoring System v3.1**. https://www.first.org/cvss/v3.1/specification-document, 2020. Citado na página 21.

GEER, D. Are companies actually using secure development life cycles? **Computer**, v. 43, n. 6, p. 12–16, June 2010. Citado na página 3.

GHAFFARI, F.; ARABSORKHI, A. A new adaptive cyber-security capability maturity model. In: **2018 9th International Symposium on Telecommunications (IST)**. [S.l.: s.n.], 2018. p. 298–304. Citado na página 6.

HASSAN, S.; SHANG, W.; HASSAN, A. E. An empirical study of emergency updates for top android mobile apps. **Empirical Software Engineering**, v. 22, n. 1, p. 505–546, Feb 2017. ISSN 1573-7616. Disponível em: <https://doi.org/10.1007/s10664-016-9435-7>. Acesso em: 09 de novembro de 2020. Citado na página 3.

HOWARD, M.; LEBLANC, D. E. **Writing Secure Code**. 2nd. ed. USA: Microsoft Press, 2002. ISBN 0735617228. Citado na página 14.

IEEE Standard Glossary of Software Engineering Terminology. **IEEE Std 610.12-1990**, p. 1–84, Dec 1990. Citado na página 5.

KIM, D.; SOLOMON, M. G. **Fundamentals of Information Systems Security**. [S.l.]: Jones & Bartlett Learning, 2018. Citado 2 vezes nas páginas 5 e 6.

MALACARNE, G. R.; CAMARGO, E. T.; NOTH, F. **Desenvolvimento Seguro de Sistemas Web: Uma Revisão Sistemática**. [S.l.], 2019. Citado na página 1.

MASOOD, A.; JAVA, J. Static analysis for web service security - tools & techniques for a secure development life cycle. **2015 IEEE International Symposium on Technologies for Homeland Security (HST)**, p. 1–6, 2015. Citado na página 5.

MEMBER, U. **Documentation for Ultimate Member**. 2021. Disponível em: <https://docs.ultimatemember.com/>. Acesso em: 16 de março de 2021. Citado na página 39.

MICROSOFT, C. **Security Development Lifecycle**. 5.2. ed. [S.l.], 2012. Citado 9 vezes nas páginas 1, 5, 12, 29, 30, 34, 38, 41 e 44.

MIGUES, J. S. S.; WARE, M. **BSIMM10 Maturity Model**. 10. [S.l.], 2019. Citado na página 1.

MITRE. **CVE, Common Vulnerabilities and Exposures**. 2019. Available from MITRE. Disponível em: <https://cve.mitre.org/data/downloads/index.html>. Acesso em: 25 de fevereiro de 2021. Citado 9 vezes nas páginas 1, 2, 24, 25, 27, 32, 36, 39 e 42.

NIST. **The National Institute of Standards and Technology (NIST)**. 2021. Disponível em: <https://www.nist.gov>. Acesso em: 21 de março de 2021. Citado na página 21.

OWASP, F. **CLASP**. 1.2. [S.l.], 2006. Citado 9 vezes nas páginas 1, 17, 30, 31, 35, 38, 39, 44 e 45.

OWASP, F. **OpemSAMM**. 2.0. [S.l.], 2020. Citado na página 1.

OWASP, T. O. W. A. S. P. **OWASP Top 10 - 2017**. [S.l.], 2017. Disponível em: <https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf>. Acesso em: 17 de novembro de 2020. Citado na página 20.

OWASP, T. O. W. A. S. P. **Secure Software Development Lifecycle Project(S-SDLC)**. [S.l.], 2017. Disponível em: <https://www.owasp.org/index.php/OWASP_Secure_Software_Development_Lifecycle_Project#tab=FAQs>. Acesso em: 24 de novembro de 2020. Citado na página 3.

RHAFFARI, I. E.; ROUDIèS, O. Benchmarking sdl and clasp lifecycle. In: **2014 9th International Conference on Intelligent Systems: Theories and Applications (SITA-14)**. [S.l.: s.n.], 2014. p. 1–6. Citado na página 2.

SORDI, J. D. **Elaboração de pesquisa científica**. Saraiva Educação S.A., 2017. ISBN 9788502210349. Disponível em: <https://books.google.com.br/books?id=NDInDwAAQBAJ>. Acesso em: 18 de fevereiro de 2021. Citado na página 20.

TENABLE. **Nessus Visão Geral**. 2021. Disponível em: <https://pt-br.tenable.com/products/nessus>. Acesso em: 16 de março de 2021. Citado na página 42.

UTO, N. **Teste de Invasão de Aplicações Web**. [S.l.]: Escola Superior de Redes, 2013. Citado 2 vezes nas páginas 1 e 5.

ZENAH, N. H. Z.; AZIZ, N. A. Secure coding in software development. In: **2011 Malaysian Conference in Software Engineering**. [S.l.: s.n.], 2011. p. 458–464. Citado 2 vezes nas páginas 1 e 5.