PROVAS DE TI
TUDO PARA VOCÊ PASSAR

Segurança no Desenvolvimento de Software

Prof. Rodrigo Macedo

Escopo do Curso

- Conceitos Iniciais
- Segurança no Desenvolvimento de Software
- SDL
- CLASP
- SAST
- DAST
- Sonarqube
- Questões de concursos

Conceitos Iniciais

- Software é componente fundamental na automação dos processos de sistemas de informação e processos da infraestrutura computacional.
- Um software deve sempre satisfazer as necessidades dos seus clientes e usuários. O software também deve ter um desempenho sem falhas por um longo período de tempo e deve ser de fácil modificação (PRESSMAN, 2006).
- “Software é um conjunto de programas de computador e a documentação associada. Software não é apenas o programa mas também toda a documentação associada e os dados de configuração necessários para fazer com que esses programas operem corretamente” (SOMMERVILLE, 2005).

Conceitos Iniciais

- Sommerville destaca os seguintes atributos essenciais de qualidade de um software:

1. **Facilidade de manutenção:** o software deve ser escrito de modo que possa evoluir para atender às necessidades mutáveis dos clientes e usuários.
2. **Nível de confiança compatível com o uso:** o nível de confiança envolve confiabilidade, proteção e segurança.
3. **Eficiência:** o software não deve desperdiçar os recursos do sistema computacional no qual é executado.
4. **Facilidade de uso:** o software deve ser de fácil utilização, deve ter uma interface apropriada e documentação adequada.

Segurança no Software

- A grande maioria dos demais incidentes de segurança da informação tem sua origem nas **vulnerabilidades** presentes no software.
- Os incidentes decorrentes da exploração dessas vulnerabilidades são geralmente relacionados com a indisponibilidade, a divulgação indevida e a perda de integridade da informação.
- Primitivas da Segurança da Informação.

Segurança no Software - STRIDE

- S - **Spoofing**: falsificação de identidade de um usuário;
- T - **Tampering**: adulteração de integridade da informação ou do sistema;
- R - **Repudiation**: negação da execução de ato cometido por um usuário;
- I - **Information Disclosure**: divulgação indevida de informação;
- D - **Denial of Service**: negação de serviço;
- E - **Elevation of Privilege**: elevação de privilégios indevidos por um usuário.

Esses efeitos podem produzir impactos de aixo alto valor, dependendo do tipo de aplicação um sistema computacional no qual o software está executando.

Segurança no Software - Primitivas

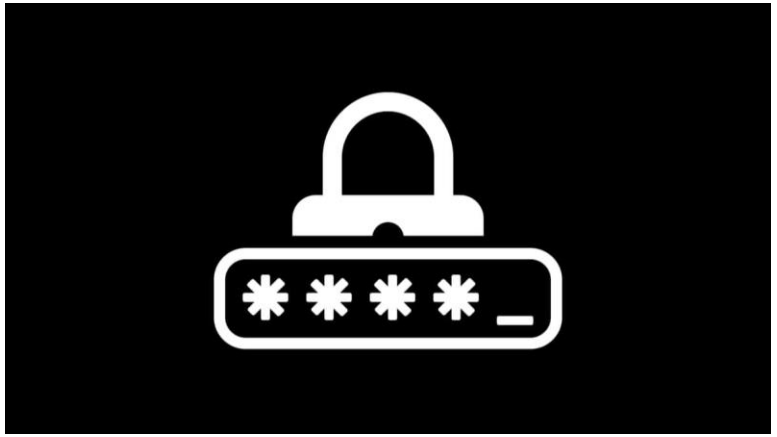
- Um software seguro é um software livre de vulnerabilidades, que funciona da maneira pretendida e que essa maneira pretendida não compromete a segurança de outras propriedades requeridas do software, seu ambiente e as informações manipuladas por ele (DSH, 2006).

1. **Disponibilidade:** o software deve estar sempre operacional e acessível para os usuários autorizados sempre que necessário.
2. **Integridade:** o software deve estar sempre protegido contra modificações não autorizadas.
3. **Confidencialidade:** no contexto da segurança do software, confidencialidade se aplica para o próprio software e para os dados que ele manipula.

Diretrizes - Software Seguro

- Algumas diretrizes a ser aplicadas para uma maior segurança no desenvolvimento de software:

1. **Senhas fortes:** Aplicações atuais buscam “obrigar” o usuário a cadastrar senhas que tenham parâmetros mínimos de segurança, conforme elencamos, além de considerar os tamanhos das senhas. Recomenda-se um tamanho mínimo de 8 caracteres, apesar de diversas aplicações aceitarem como quantidade razoável 6 caracteres. Além disso, algumas aplicações são capazes também de gerar senhas extremamente fortes para os usuários.



Diretrizes - Software Seguro

1. **Atualização de Aplicações:** As atualizações disponibilizadas pelos fabricantes não se restringem ao acréscimo de novas funcionalidades e recursos, mas também contemplam correções de bugs, falhas de segurança, entre outros. Assim, não basta que o software seja seguro por si próprio se softwares complementares e integrados ou sistemas operacionais não se encontram atualizados, com diversas brechas de segurança.

Diretrizes - Software Seguro

1. **Fuzzing:** Esta é uma técnica utilizada para testar erros em aplicações. É amplamente utilizado no processo de desenvolvimento de softwares seguros devido sua capacidade de detectar defeitos que usuários não descobrem com facilidade. Assim, caso este seja descoberto em ambiente de produção, pode gerar grandes danos aos usuários de determinada aplicação. A referida técnica consiste, basicamente, em enviar entradas randômicas para a aplicação. Por este motivo, também é conhecida como injeção de falhas, teste de validação robusta, teste de sintaxe ou teste de negação.

Assim, o Fuzzing injetará informações incomuns como tamanhos diferenciados, caracteres não utilizados e, paralelamente, monitorará o comportamento da aplicação, pois esta poderá travar ou vazar dados de forma indevida.

Diretrizes - Software Seguro

- Boas práticas de Código Seguro:

1. **Documentação:** A documentação pode ser extremamente importante no diagnóstico e resolução de forma mais fácil e rápida de problemas.
2. **Validação de Entrada:** Este processo consiste em inserir dados em pontos de entrada da aplicação e verificar se o comportamento está de acordo com o esperado pelo desenvolvedor, documentando todo o processo. Um típico exemplo é a utilização de máscaras que obrigam o usuário de inserir dados no formato esperado, como o CPF.

Diretrizes - Software Seguro

- Boas práticas de Código Seguro:

1. **Manipulação de Erros:** O tratamento de erros é um ponto muito importante no desenvolvimento de aplicações seguras. Essas aplicações sempre estarão sujeitas a erros e, por medida de segurança, é importante que haja um padrão de mensagem de erro para o usuário que não vaze informações a respeito da aplicação, evitando assim que um atacante obtenha essas informações para aprimorar seus ataques.

- **Baseline de Configuração de Aplicação:** As aplicações podem utilizar diversos componentes pelos quais possuem dependências para seu funcionamento. É importante identificar esses componentes e entender como as aplicações fazem uso dessas. A partir de então, pode-se trabalhar em cima dessas aplicações com configurações seguras que darão a devida base e sustentação da aplicação principal.

SDL (Ciclo de Vida do Desenvolvimento Seguro)

- O SDL é um processo de desenvolvimento de softwarecom segurança adotado pela Microsoft e descrito por Howard e Lipner (2006), Lipner e Howard (2005) e Microsoft (2010).
- O SDL envolve a adição de uma série de atividades e produtos concentrados na produção de software seguro, desenvolvidas em cada fase do processo de desenvolvimento de software.
- O processo de desenvolvimento de software seguro, globalmente aceito na Microsoft, segue, em termos gerais é composto pelas fases de treinamento, requisitos, design, implementação, verificação, liberação e resposta.

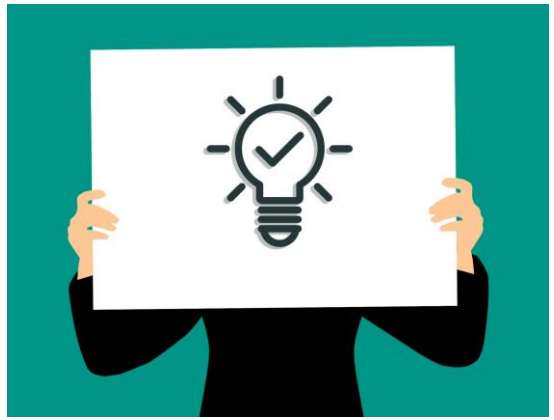
SDL (Ciclo de Vida do Desenvolvimento Seguro)

- Ajuda os desenvolvedores na construção e softwares mais seguros e na conformidade com requisitos de segurança, ao mesmo tempo que reduz custos.
- **Objetivos:** reduzir o número de defeitos de codificação e design relacionados à segurança e reduzir a gravidade de todos os demais defeitos.

SDL (Ciclo de Vida do Desenvolvimento Seguro)

O SDL faz uso de um conjunto de princípios de alto nível conhecido como SD3+C:

1. **Seguro por Projeto (Design):** a arquitetura, o projeto e a implementação do software devem ser executados de forma a protegê-lo e proteger as informações que ele processa, além de resistir a ataques.
2. **Seguro por Padrão (Default):** visto que na prática é impossível obter uma segurança perfeita os projetistas devem considerar a possibilidade de haver falhas de segurança.



SDL (Ciclo de Vida do Desenvolvimento Seguro)

O SDL faz uso de um conjunto de princípios de alto nível conhecido como SD3+C:

1. **Seguro na Implantação (Deployment):** o software deve conter ferramentas e orientação que ajudem os usuários finais e/ou administradores a usá-lo com segurança, assim como a implantação das atualizações deve ser fácil.
2. **Comunicações:** os desenvolvedores de software devem estar preparados para a descoberta de vulnerabilidades do produto e devem comunicar-se de maneira aberta e responsável com os usuários finais e/ou com os administradores para ajudá-los a tomar medidas de proteção (como instalar patches ou implantar soluções alternativas).

SDL - Treinamento de Segurança

- O treinamento de segurança envolve aplicação de treinamento básico e avançado aos membros da equipe de desenvolvimento de software, abordando aspectos como princípios, tendências em segurança e privacidade.

Tópicos abordados no treinamento:

1. O desenho seguro, envolvendo a **redução da superfície de ataque** (campos de entrada, portas e serviços em execução num computador), a **defesa em profundidade** (utilização de firewalls, vpn, ids, etc), o **princípio do privilégio mínimo** (controle de acesso bem definido).
2. A **modelagem de ameaças**, que consiste na realização de estudos visando: analisar como um adversário ou atacante em potencial vê uma aplicação;

SDL - Treinamento de Segurança

Tópicos abordados no treinamento:

1. A **codificação segura**, que envolve desenvolver capacidade construir código capaz de resistir a ataques como: estouro de buffers, erros de aritmética de inteiros, cross site scripting, SQL Injection, deficiências na criptografia e particularidades específicas da plataforma computacional utilizada para desenvolvimento das aplicações.
2. Teste de segurança, que envolve equilibrar testes funcionais com os testes de segurança, realizar testes baseados nos riscos e ameaças às quais a aplicação está sujeita, aplicar metodologias de teste de software e **automatizar os testes**.
3. Questões de **privacidade de informações pessoais** (LGPD).

SDL - Requisitos

- Nessa fase os requisitos de segurança e privacidade da aplicação são analisados em maior detalhe e é produzida uma **análise de custos** visando determinar se os **recursos** necessários para **melhorar a segurança e privacidade da aplicação** estão compatíveis com os **objetivos de negócio** do projeto de software.

Tópicos abordados nessa fase:

1. **Estabelecer requisitos de privacidade e segurança:** definição e **integração** cedo dos requisitos de segurança e privacidade ajudam a tornar mais fácil a **identificação** das princip
   e resultados

SDL - Requisitos

Tópicos abordados nessa fase:

1. **Criar portas de qualidade (Quality Gates) e Barras de defeitos (Bug Bars):** definição no início de **níveis mínimos aceitáveis** de **segurança** e de qualidade de **privacidade** ajuda a equipe a entender os riscos associados a questões de segurança, identificar e corrigir bugs de segurança durante o desenvolvimento.
2. **Realizar avaliações de riscos de segurança e privacidade:** examinar o design de software com base nos **custos** e requisitos regulamentares ajuda a equipe a identificar quais partes de um projeto exigirá revisões de **modelagem de ameaças** e design de segurança antes do lançamento e determinar a taxa de impacto de privacidade de um recurso.

SDL - Design

- Nessa fase se realiza análise da superfície de ataque da aplicação e se produz um modelo de ameaças da aplicação. A análise de superfície de ataque consiste em fazer um levantamento de formas de acesso do aplicativo ou componente.

Tópicos abordados nessa fase:

1. **Estabelecer Requisitos de Projeto:** considerar as preocupações de segurança e privacidade ajuda a **minimizar o risco** de interrupções de cronograma e a **reduzir as despesas** de um projeto.

SDL - Design

Tópicos abordados nessa fase:

1. **Redução / Análise da Superfície de Ataque:** reduzir as oportunidades para os atacantes explorarem um potencial ponto fraco ou vulnerabilidade requer a análise cuidadosa da superfície de ataque global e inclui **desabilitar** ou **restringir** o acesso aos serviços do sistema, aplicando o princípio do Menor Privilégio e empregando a defesa em camadas sempre que possível.
2. **Utilizar Modelagem de Ameaças:** aplicar uma abordagem estruturada para cenários de ameaça durante o projeto ajuda a uma equipe **identificar** de forma mais **eficaz** e menos dispendiosa vulnerabilidades de segurança, determinar os **riscos** contra essas **ameaças** e estabelecer atuneações apropriadas.

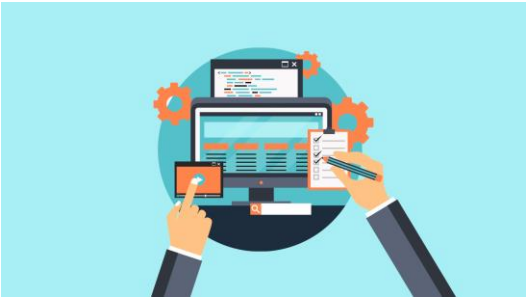
SDL - Implementação

- A implementação consiste na produção de código executável, usando-se uma ou mais linguagens de programação, sejam elas interpretadas ou compiladas. É preciso empregar técnicas de **programação defensiva** para desenvolver código que resista ao ataque de usuários hackers.
- A forma mais comum de garantir que será produzido código seguro é por meio de aderência ao uso de **padrões de codificação** que reduzem a ocorrência de vulnerabilidades de código, chamados padrões de **codificação segura**, e da verificação de que os programadores a estão adotando. Tais padrões tendem a evitar a injeção de código e outros ataques.
- Também se recomenda, para verificação, o uso de ferramentas de **análise estática de código**, que identificam formas de codificação propensas à introdução de vulnerabilidades.

SDL - Implementação

Tópicos abordados nessa fase:

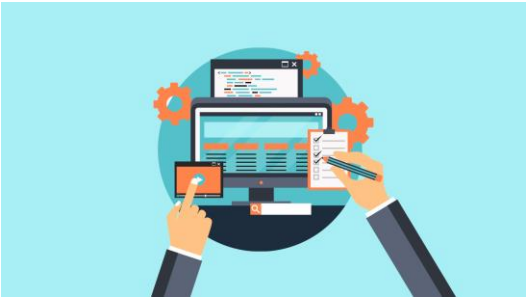
1. **Utilizar Ferramentas Aprovadas:** a publicação de uma lista de ferramentas aprovadas e verificações de segurança associadas ajuda a **automatizar** e aplicar facilmente **práticas de segurança** a um custo baixo. Manter a lista atualizada regularmente permite a inclusão de novas funcionalidades de análise de segurança e proteção.
2. **Executar Análise Estática**: Analisar o código-fonte antes da compilação fornece um método escalável de **revisão** de segurança do **código** e ajuda a garantir que as políticas de codificação seguras estão sendo seguidas.



SDL - Implementação

Tópicos abordados nessa fase:

1. **Depreciar Funções Inseguras:** analisar todas as funções e APIs do projeto e proibir aquelas inseguras ajuda a **reduzir erros potenciais de segurança** com um custo muito pequeno de engenharia. As ações específicas incluem o uso de cabeçalhos, compiladores mais recentes, ou ferramentas de leitura de código para verificar o código em busca de funções na lista de funções proibidas, e, em seguida, subsitituí-las por alternativas mais seguras.



SDL - Verificação

- Envolve a realização de testes, inspeções de código e análise de documentação do software, por meio de ferramentas automatizadas, como de análise estática de código, ou de técnicas manuais como inspeções de código e auditoria de configuração, dentre outras.

Tópicos abordados nessa fase:

1. **Teste de Fuzz:** induzir falhas no programa ao **introduzir** deliberadamente **dados** malformados ou **aleatórios** para uma aplicação ajuda a revelar possíveis problemas de segurança antes da liberação.

SDL - Verificação

Tópicos abordados nessa fase:

1. **Executar Análise Dinâmica:** executar verificações em tempo de execução de funcionalidades utilizando ferramentas que **monitoram** o **comportamento** de aplicações em busca de memória corrompida, problemas de privilégio de usuário e outros problemas de segurança críticos.
2. **Revisão de Superfície de Ataque:** revisar a **medição de superfície de ataque** após a conclusão do código ajuda a garantir que qualquer mudança de projeto ou implementação na aplicação ou sistema foram levados em consideração e que quaisquer novos vetores de ataque criadas como resultado das mudanças foram revistos e mitigados, incluindo modelos de ameaças.

SDL - Liberação

- Durante a fase de liberação é produzido um plano de ação descrevendo como poderá se dar a resposta da equipe de tratamento de incidentes de segurança da informação, a eventualidade de descoberta de uma vulnerabilidade de segurança da aplicação ou mesmo na ocorrência de um incidente de segurança.

Tópicos abordados nessa fase:

1. **Conduzir Revisão de Segurança Final:** revisar deliberadamente todas as atividades de segurança que foram executadas ajuda a assegurar que a release do software está pronta. Geralmente inclui examinar modelos de ameaças, saídas de ferramentas e **desempenho em relação a porta de qualidade e barras de defeito definidas** na fase de requisitos.

SDL - Liberação

Tópicos abordados nessa fase:

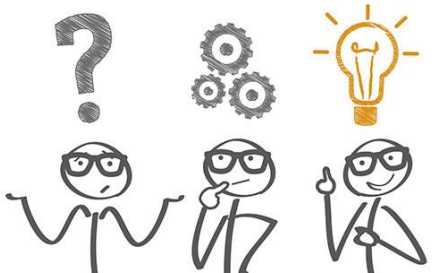
1. **Criar um PLano de Resposta a Incidentes:** preparar um plano de resposta a incidentes é crucial para **ajudar a lidar com as novas ameaças que podem surgir ao longo do tempo**. Ele inclui a identificação de contatos de emergência de segurança adequados e o estabelecimento de planos de serviço de segurança para código herdadas de outros grupos dentro da organização e para código licenciado de terceiros.
2. **Certificar Lançamento e Arquivamento:** atestar software previamente ao lançamento auxilia a **garantir que os requisitos de segurança e privacidade foram atendidos**. O arquivamento de todos os dados pertinentes é essencial para a execução de tarefas de manutenção pós-liberação e ajuda a reduzir os custos de longo prazo associados com a sustentação da engenharia de software.

SDL - Resposta

- Envolve o tratamento de incidentes relacionados à aplicação. No texto sobre Tratamento de Incidentes esse tema será abordado com profundidade. Responder a defeitos de segurança e trabalhar com pessoas que descobrem problemas de segurança no código. Aprender com os erros, através da análise e documentação da causa dos defeitos.

Tópicos abordados nessa fase:

1. **Executar Plano de Resposta a Incidentes:** ser capaz de **implementar** o **plano de resposta a incidentes** na fase de lançamento é essencial para ajudar a proteger os clientes de vulnerabilidades de segurança ou privacidade que podem emergir.



CLASP

- Inicialmente desenvolvido pela empresa Secure Software, hoje sob a responsabilidade da OWASP, o CLASP é um conjunto de componentes de processo dirigido por atividade e baseado em regras, que articula práticas para construção de software seguro.
- O CLASP é um conjunto de pedaços de processos que pode ser **integrado** a **qualquer processo** de desenvolvimento de software. Foi projetado para ser de fácil utilização.
- Tem um enfoque **prescritivo**, documentando as atividades que as organizações devem realizar, proporcionando uma ampla riqueza de recursos de segurança que facilitam a implementação dessas atividades.

CLASP - Estrutura

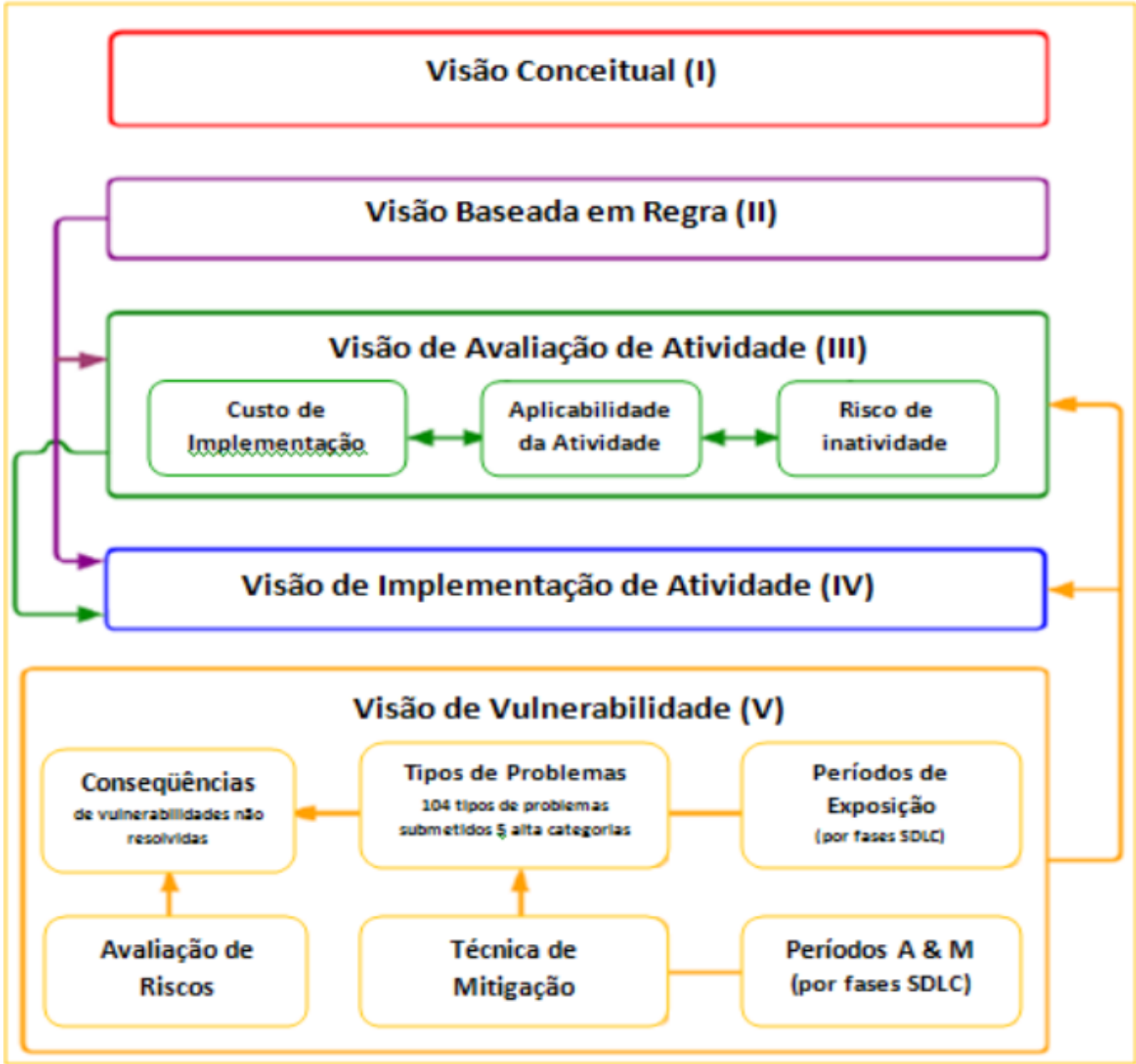
A estrutura do CLASP e as dependências entre os componentes do processo CLASP são organizados em:

1. Visões CLASP
2. Recursos CLASP
3. Caso de Uso e Vulnerabilidade

- São propostas 24 atividades divididas em componentes de processos discretos ligados a um ou mais papéis de um projeto. Desta forma, o CLASP provê um guia para participantes de um projeto: gerentes, auditores de segurança, desenvolvedores, arquitetos e testadores, entre outros.

CLASP - Visões

- Um processo de desenvolvimento de software seguro CLASP pode ser analisado através de perspectivas de alto nível, chamadas Visões CLASP, que são classificadas em: Visão de Conceitos; Visão baseada em Regras; Visão de Avaliação de Atividades; Visão de Implementação de Atividades e Visão de Vulnerabilidade.



CLASP - Visões

1. **Visão Conceitual:** apresenta uma visão geral de como funciona o **processo** CLASP e como seus **componentes interagem**. São introduzidas as melhores práticas, a interação entre o CLASP e as políticas de segurança, alguns conceitos de segurança e os componentes do processo.
2. **Visão Baseada em Regras:** introduz as **responsabilidades** básicas de cada **membro** do projeto (gerente, arquiteto, especificador de requisitos, projetista, implementador, analista de testes e auditor de segurança) relacionando-os com as **atividades propostas**, assim como a especificação de quais são os requerimentos básicos para que cada função.

CLASP - Visões

1. **Visão de Avaliação de Atividades:** descreve o propósito de cada atividade, bem como o custo de implementação, a aplicabilidade, o impacto relativo, os riscos em caso de não aplicar a atividade.
2. **Visão de Implementação:** descreve o conteúdo das 24 atividades de segurança definidas pelo CLASP e identifica os responsáveis pela implementação, bem como as atividades relacionadas.
3. **Visão de Vulnerabilidades:** possui um catálogo que descreve 104 tipos de vulnerabilidades no desenvolvimento de software, divididas em cinco categorias: Erros de Tipo e Limites de Tamanho; Problemas do Ambiente; Erros de Sincronização e Temporização; Erros de Protocolo e Erros Lógicos em Geral. Nessa atividade também é realizada técnicas de mitigação e avaliação de risco.

CLASP - Recursos

- O processo CLASP suporta planejamento, implementação e desempenho para atividades de desenvolvimento de software relacionado com segurança.
- Os recursos do CLASP fornecem acesso para os artefatos que são especialmente úteis se seu projeto está usando ferramentas para o processo CLASP.
- Os recursos CLASP são compostos por uma lista de onze artefatos.

CLASP - Recursos

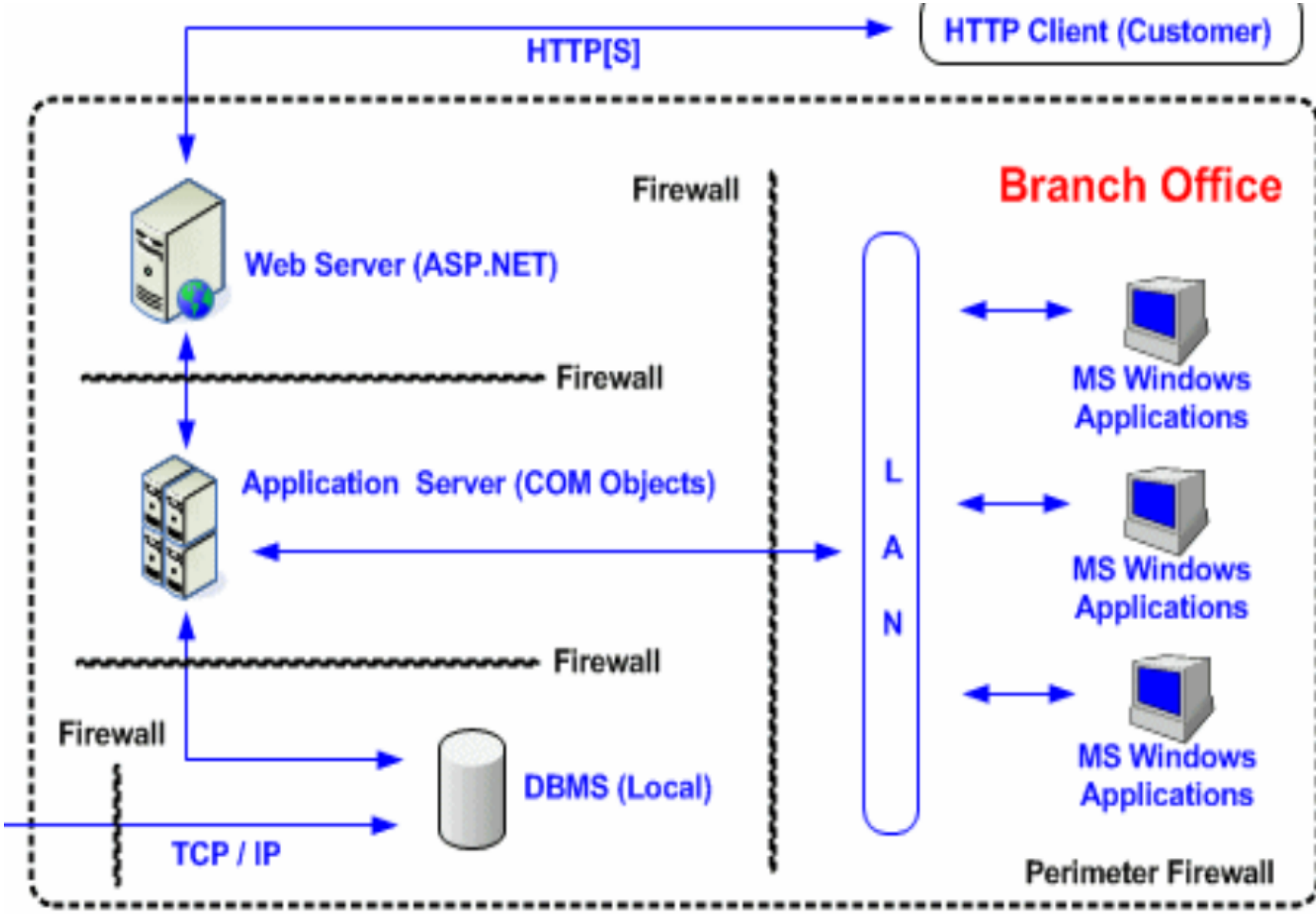
Artefatos e Visões
Princípios Básicos em Segurança de aplicações (todas as Visões)
Exemplo de Princípios Básicos: Validação de Entrada (todas as Visões)
Exemplo de princípios Básicos Violação: Modelo penetração-e-*patch* (todas as Visões)
Serviços Essenciais de Segurança (todas as Visões; especialmente III)
Planilhas em Guia com Codificação de Exemplo (Visões II, III e IV)
Planilhas de Avaliação de Sistema (Visões III e IV)
Mapa de Caminhos Exemplo: Projetos Legados (Visão III)
Mapa de Caminho Exemplo: Começo de Novo Projeto (Visão III)
Criação do Plano de Engenharia de Processo (Visão III)
Formação da Equipe de Engenharia de Processo (Visão III)
Glossário de Equipe de Segurança (todas as Visões)

CLASP - Caso de Uso de Vulnerabilidades

- Os Casos de Uso de Vulnerabilidade descrevem condições nas quais os serviços de segurança podem se tornar vulneráveis em aplicações de software.
- Os Casos de Uso fornecem aos usuários CLASP exemplos específicos, com fácil entendimento, e relacionamento de causa e efeito sobre a codificação do programa e seu design, além de possíveis resultados de exploração de vulnerabilidades em serviços de segurança básicos como autorização, autenticação, confidencialidade, disponibilidade, responsabilização e não-repúdio.
- Cada Use Case é composto pelo diagrama de arquitetura de componentes onde há descrição de cada componente, assim como outro diagrama contendo o fluxo do processo relacionado com a segurança do software.

CLASP - Caso de Uso de Vulnerabilidades

1. Inicialmente o cliente solicita autenticação para a filial. Office Branch;
2. O cliente é autenticado por certificação digital via HTTPs;
3. O cliente solicita uma aplicação específica utilizando HTTPS;
4. O cliente é autorizado a executar o aplicativo;
5. A página web solicita a execução de um objeto COM que está localizado em um servidor de aplicação local;
6. Acesso ao objeto COM é autorizado;
7. O objeto COM garante apenas os dados relevantes acessíveis/atualizados para o cliente autorizado;
8. A aplicação responde com os dados acessíveis e atualizados pela aplicação;
9. O objeto COM solicita acesso e atualização para os dados localizados no SGBD local para desenvolver funções de negócio específicas;
10. Acesso aos dados é autorizado fornecido às tabelas dos SGBDs locais são permitidos para acesso web.



SAST

- SAST é uma ferramenta de Segurança de aplicativos (AppSec), que verifica os códigos fonte, binário ou byte. Uma ferramenta de **teste de caixa branca**, que identifica a causa principal das vulnerabilidades e ajuda a corrigir as falhas de segurança subjacentes.
- As soluções SAST analisam um aplicativo “de dentro para fora” e não esperam o sistema entrar em execução para fazer a verificação.
- As ferramentas SAST garantem **feedback em tempo real** aos desenvolvedores, à medida que codificam, ajudando-os a corrigir falhas antes de transferirem o código para a próxima fase do SDLC.
- A metodologia também fornece **representações gráficas dos problemas encontrados**, desde o início até o coletor. Isso ajuda a navegar pelo código com mais facilidade.

SAST

- Algumas ferramentas apontam a localização exata das vulnerabilidades e **destacam o código arriscado**, fornecendo orientações detalhadas sobre como reparar problemas e o melhor local no código para corrigi-los, sem exigir profundo domínio em segurança.
- Com o Sast, também é possível criar relatórios personalizados, que podem ser exportados offline e rastreados através de painéis.
- É importante reforçar que as ferramentas SAST **devem ser executadas no aplicativo constantemente, durante compilações diárias ou mensais**, toda vez que o código é analisado ou durante o lançamento de um código.

SAST - Recomendações

1. **Escolha da Ferramenta:** Escolha uma ferramenta de análise estática que faça revisões de código para aplicativos escritos nas linguagens de programação que você usa.
2. **Customização da Ferramenta:** Alinhe a ferramenta para atender às necessidades da empresa. Por exemplo, **configurando-a para diminuir falsos positivos ou encontrar fragilidades de segurança adicionais, escrevendo novas regras ou corrigindo as existentes**.
3. **Integração de Aplicativos:** Este estágio envolve a **triagem dos resultados da varredura para eliminar falsos positivos**. Após o conjunto de problemas ser finalizado, eles devem ser rastreados e fornecidos às equipes de implantação para ser feita a correção adequada.

SAST - Motivações

- Nas organizações, a quantidade de desenvolvedores ultrapassa drasticamente o número de funcionários de segurança, tornando desafiador para uma empresa encontrar os recursos necessários para fazer revisões de código até mesmo em uma fração de seus aplicativos.
- Essas ferramentas varrem milhões de linhas de código em poucos minutos.
- Uma grande vantagem das ferramentas SAST é justamente a capacidade de analisar 100% da base de código.
- A metodologia identifica automaticamente vulnerabilidades críticas — como injeção de SQL, cross-site scripting, buffer overflows e outros.

DAST

- DAST, ou, Dynamic application security testing, são tecnologias projetadas para detectar indícios de vulnerabilidades numa aplicação enquanto ela está executando.
- É o processo de análise de um aplicativo da web que usa o front end para encontrar vulnerabilidades por meio de ataques simulados. Esse tipo de abordagem avalia o aplicativo “de fora para dentro”, atacando-o como um usuário mal-intencionado faria.
- Essas soluções não tem qualquer contato com as "entranhas" do código, mas somente com aquilo que os usuários percebem da aplicação, com suas interfaces e funcionalidades. Por esse motivo, também essas técnicas são conhecidas como testes "caixa-preta".

DAST

- O mecanismo se baseia em um banco de dados de vulnerabilidades conhecidas, imita um ataque e emite alertas em caso de sucesso do ataque.
- Além disso, um scanner DAST observa como o aplicativo responde durante um ataque, reduzindo assim os falsos positivos e fornecendo percepções críticas sobre o aprimoramento dos recursos e controles de segurança existentes.
- A maioria de soluções DAST testam somente as interfaces HTTP e HTML expostas pelas aplicações web. Algumas ferramentas mais especializadas, no entanto, são projetadas especificamente para aplições cliente servidor que utilizam protocolos de rede diferentes dos utilizados na web bem como problemas de integridade de dados trafegados (por exemplo remote procedure call, Session Initiation Protocol - utilizado em VoIP).

DAST - Motivações

Os mecanismos DAST são normalmente usados para identificar vulnerabilidades como:

1. Vulnerabilidades de validação de dados / entrada
2. Ataques de injeção
3. Gerenciamento de sessão quebrada
4. APIs inseguras

- A análise dos dados de vulnerabilidade do Fortify on Demand (FoD) mostrou que 94% dos mais de 11.000 aplicativos da web continham bugs nos recursos de segurança, e que as questões de qualidade do código e de abuso de APIs praticamente dobraram nos últimos 4 anos.

SAST x DAST

- Trata-se de duas ferramentas muito importantes e usadas para localizar vulnerabilidades em um software, automatizando algumas barreiras de segurança e evitando que o fluxo de trabalho se torne lento.
- O SAST **deve ser utilizado mais no início e preferencialmente nos arquivos que possuem o código-fonte**. Porém, o método não consegue identificar fragilidades que surgem com a aplicação em produção.
- Em contrapartida, o **DAST deve ser realizado com a aplicação rodando**, em ambiente igual ao de produção.
- Porém, esse é um teste mais complexo, e as vulnerabilidades localizadas tendem a ser direcionadas para serem corrigidas somente no próximo começo do ciclo de desenvolvimento.

SAST x DAST

SAST	DAST
Usada para teste de segurança de caixa branca	Usada para teste de segurança de caixa preta
Age analisando o código-fonte, sem ser necessária a execução do código	Age enquanto o aplicativo está em execução
As vulnerabilidades são encontradas antes do desenvolvimento e são menos caras para consertar	Encontra problemas que não podem ser identificados em uma análise estática
Não consegue identificar problemas relacionados ao tempo e ao ambiente	As vulnerabilidades são encontradas após o desenvolvimento e são mais caras para consertar
É capaz de dar suporte a todos os tipos de software	Consegue identificar problemas relacionados ao tempo e ao ambiente

IAST

- O teste interativo de segurança de aplicativos (IAST) combina técnicas SAST e DAST, permitindo verificações de segurança em vários estágios de desenvolvimento e implantação.
- Ao fazer isso, as ferramentas IAST monitoram continuamente os aplicativos para coletar informações sobre desempenho, funcionalidade e bugs.
- Uma análise de segurança abrangente envolve testar o código de acesso, informações de rastreamento de pilha, bibliotecas, controle de tempo de execução e informações de fluxo de dados.
- Em sua essência, o IAST combina os benefícios dos mecanismos DAST e SAST, identificando vulnerabilidades de tempo de execução e destacando linhas de código mal escritas.

RASP

- O RASP analisa o comportamento de um aplicativo da web e o contexto para detectar entradas maliciosas ou ameaças em tempo real.
- Essas ferramentas utilizam os recursos inatos do aplicativo para monitorar seu próprio comportamento, permitindo a detecção autônoma e a mitigação de ataques.
- Devido à sua metodologia flexível para rastrear vulnerabilidades, o RASP pode ser acomodado em todas as fases do SDLC e é preferido em uma ampla variedade de casos de uso.
- Depois que uma plataforma RASP é instalada em um servidor, ela incorpora segurança ao interceptar a comunicação entre o aplicativo e o usuário. O agente RASP executa funções de segurança como validação de dados e autenticação de usuário diretamente no aplicativo.

Análise Estática de Código Fonte

- A Análise Estática de Código-Fonte trata do processo de detecção de erros no código sem, de fato, executá-lo - pode ser considerado um processo de revisão automática de código.
- Analisadores estáticos ferramentas de software que varrem o código-fonte de um programa e detectam possíveis defeitos e anomalias.
- Eles complementam os recursos de detecção de erros providos pelo compilador da linguagem.
- O objetivo é chamar a atenção do inspetor para anomalias do programa, como variáveis usadas sem serem iniciadas, variáveis não usadas ou dados cujos valores poderiam ficar fora de extensão.
- Geralmente, é muito utilizada por desenvolvedores antes e durante o teste de componente e de integração e por projetistas durante a modelagem do software.

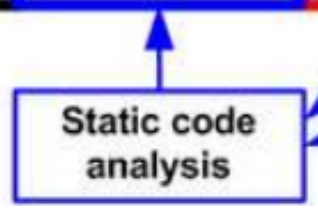
Análise Estática de Código Fonte - Vantagens

- Detecção de defeitos antes da execução do teste.
- Conhecimento antecipado sobre aspectos suspeitos no código através de métricas, por exemplo, na obtenção de uma medida da alta complexidade.
- Identificação de defeitos dificilmente encontrados por testes dinâmicos.
- Detecção de dependências e inconsistências em modelos de software, como links perdidos.
- Aprimoramento da manutenibilidade do código e construção.
- Detecção de vulnerabilidade na segurança.
- Detecção de inconsistência entre as interfaces dos módulos e componentes.

Análise Estática de Código Fonte - Vantagens

- Abordagem de prevenção de erros em vez de detecção de erros é mais eficiente no aprimoramento da confiabilidade de programa.
- De acordo com Steve McConnell, consertar um erro na fase de testes custa dez vezes mais do que na fase de implementação.

	Time Detected				
Time Introduced	Requirements	Architecture	Construction	System Test	Post-Release
Requirements	1	3	5-10	10	10-100
Architecture	-	1	10	15	25-100
Construction	-	-	1	10	10-25

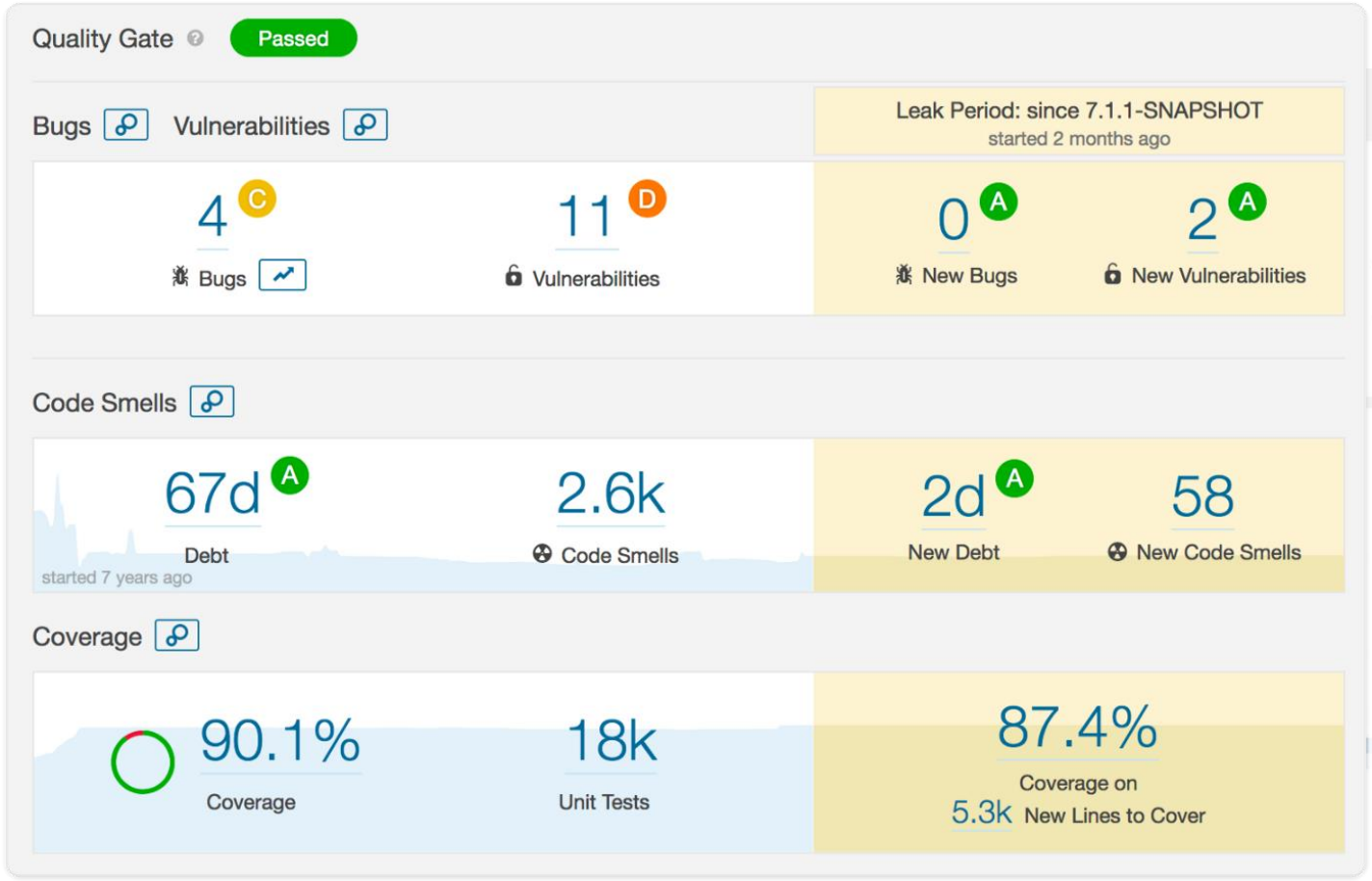


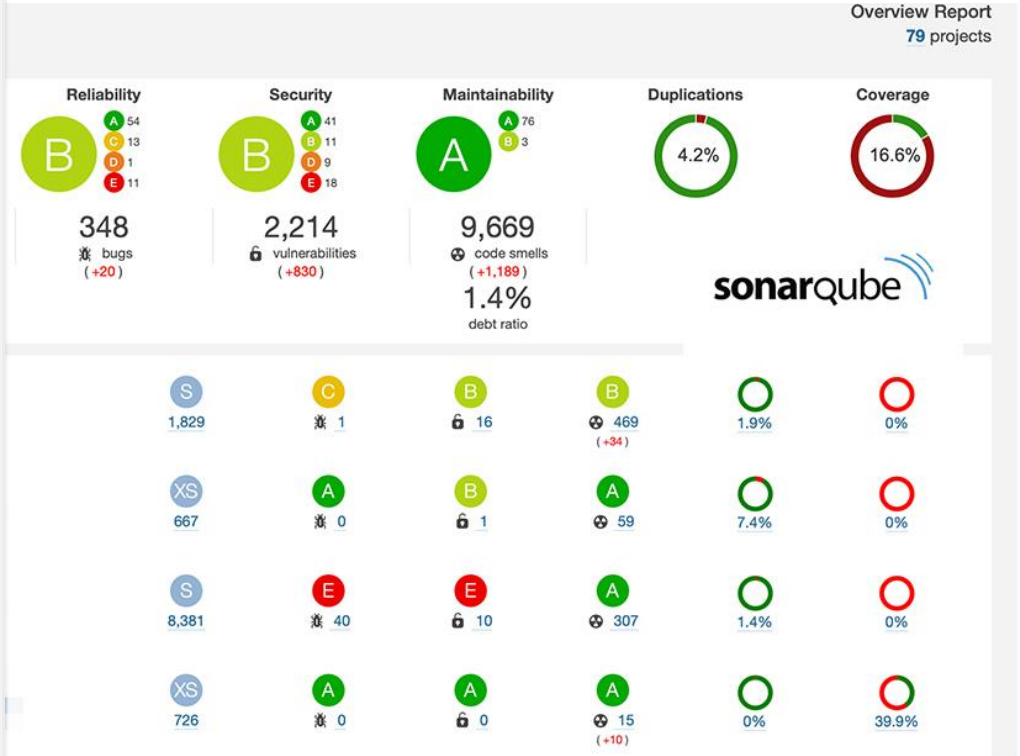
Sonarqube

- O SonarQube é uma ferramenta web open-source utilizada para gerenciar a qualidade do código - ele cobre sete categorias: arquitetura e design; comentários; duplicações de código; padrões de codificação; testes; cobertura ciclomática e bugs em potencial.
- O Débito técnico é igual juros de cartão de crédito: quanto mais demoramos a pagar, mais cara a conta final fica. E com isso derrubamos a **qualidade** do nosso código. E baixa qualidade trás: baixa velocidade no desenvolvimento da aplicação, problemas em produção, dificuldade em manter a base de código atual, confiabilidade no sistema entre outros.
- O Sonarqube é a ferramenta que vai nos ajudar em tempo de desenvolvimento a analisar, identificar e corrigir esses erros antes que seja tarde demais.

Sonarqube

- Em suma, podemos dizer que ele propõe a ser a central de qualidade do seu código-fonte, possibilitando o controle sobre um grande número de métricas de software, e ainda apontando uma série de possíveis bugs.





Sonarqube - Arquitetura

A Arquitetura do SonarQube é composta de três componentes:

1. **Banco de Dados:** Deve haver apenas um! Ele mantém os resultados da análise, os projetos e configuração global, mas também mantém uma análise histórica.
2. **Servidor Web:** Deve haver apenas um! Ele é responsável por fornecer aos usuários diversos painéis de qualidade de código-fonte e por configurar a instância do SonarQube.
3. **Analisadores**: Um conjunto de analisadores de código-fonte que são agrupados e acionados por demanda – eles utilizam a configuração armazenada no banco de dados

Por ele suporta diversas linguagens de programação, com o pré-requisito de ter instalado e configurado o JDK ou OpenJDK.

Q1) [ESAF 2015] O SDL (Security Development Lifecycle), processo de desenvolvimento de software com segurança proposto pela Microsoft, segue, em geral, um fluxo composto pelas fases de treinamento, requisitos, design, implementação, verificação, lançamento e resposta. Na fase de implementação são recomendados os seguintes elementos do SDL, exceto:

a)realizar revisões de código.

b)aplicar padrões de codificação e teste.

c)aplicar ferramentas de verificação de código de análise estática.

d)aplicar ferramentas de verificação de código de análise dinâmica.

e)aplicar ferramentas de testes de segurança, incluindo ferramentas de difusão.

Q1) [ESAF 2015] O SDL (Security Development Lifecycle), processo de desenvolvimento de software com segurança proposto pela Microsoft, segue, em geral, um fluxo composto pelas fases de treinamento, requisitos, design, implementação, verificação, lançamento e resposta. Na fase de implementação são recomendados os seguintes elementos do SDL, exceto:

a)realizar revisões de código.

b)aplicar padrões de codificação e teste.

c)aplicar ferramentas de verificação de código de análise estática.

d)aplicar ferramentas de verificação de código de análise dinâmica.

e)aplicar ferramentas de testes de segurança, incluindo ferramentas de difusão.

Q2) [CESPE 2013 PF) No que se refere a processos de desenvolvimento seguro de aplicações, julgue os itens subsecutivos.

O processo SDL (Secure Development Lifecycle) tem sido adotado pela Microsoft no desenvolvimento de alguns de seus produtos, como Windows Server, SQL Server e Exchange Server, reduzindo o número de vulnerabilidades encontradas nesses produtos em versões desenvolvidas sem o uso do SDL. Uma das características desse processo é que ele provê dois roteiros, sendo um com foco no suporte a desenvolvimento de novos sistemas com base em um processo iterativo, e outro que enfoca a manutenção de sistemas já existentes.

Q3) [CESPE TCE PA 2016] Acerca de segurança de banco de dados e de desenvolvimento de software, julgue o item subsecutivo.

Na metodologia de desenvolvimento seguro de software SDL (Security Development Lifecycle), a modelagem de ameaças é realizada na fase de requisitos.

Q2) [CESPE 2013 PF) No que se refere a processos de desenvolvimento seguro de aplicações, julgue os itens subsecutivos.

O processo SDL (Secure Development Lifecycle) tem sido adotado pela Microsoft no desenvolvimento de alguns de seus produtos, como Windows Server, SQL Server e Exchange Server, reduzindo o número de vulnerabilidades encontradas nesses produtos em versões desenvolvidas sem o uso do SDL. Uma das características desse processo é que ele provê dois roteiros, sendo um com foco no suporte a desenvolvimento de novos sistemas com base em um processo iterativo, e outro que enfoca a manutenção de sistemas já existentes. ERRADO.

Q3) [CESPE TCE PA 2016] Acerca de segurança de banco de dados e de desenvolvimento de software, julgue o item subsecutivo.

Na metodologia de desenvolvimento seguro de software SDL (Security Development Lifecycle), a modelagem de ameaças é realizada na fase de requisitos. ERRADO.

Q4) [IADES PC DF 2016] Acerca dos conceitos relacionados ao desenvolvimento seguro de aplicações, assinale a alternativa correta.

a) A responsabilidade de desenvolver aplicações seguras é dos desenvolvedores. Dessa forma, com treinamento adequado, todos os desenvolvedores podem encontrar vulnerabilidades no código e, assim, criar código seguro e resiliente.

b) Para que um software seja considerado seguro, é suficiente que possua certa resiliência, ou seja, que, ao ser intencionalmente forçado a falhar por agentes mal-intencionados, ele possa retornar ao seu estado inicial sem realizar operações não planejadas.

c) A fase de requisitos de software é de pouca ou nenhuma importância para a construção de software seguro, já que se concentra nos requisitos funcionais e não funcionais da aplicação. A responsabilidade de manter a segurança do software é do time de segurança da informação.

d) A aplicação de criptografia na informação gerenciada pela aplicação e o uso do SSL/TLS para a proteção da comunicação são suficientes para manter as aplicações web protegidas. Com essas medidas de proteção, ataques a aplicações como os descritos em guias como o OWASP Top 10 tornam-se secundários.

e) Um ciclo de desenvolvimento de software seguro é aquele em que atividades de segurança são aplicadas ao longo das etapas de requisitos, projeto, codificação, testes, operação e descarte. Essas atividades podem incluir revisão de segurança no projeto de arquitetura da aplicação e no respectivo código fonte, além de testes com foco em segurança, realizados pela equipe de garantia da qualidade.

Q4) [IADES PC DF 2016] Acerca dos conceitos relacionados ao desenvolvimento seguro de aplicações, assinale a alternativa correta.

a) A responsabilidade de desenvolver aplicações seguras é dos desenvolvedores. Dessa forma, com treinamento adequado, todos os desenvolvedores podem encontrar vulnerabilidades no código e, assim, criar código seguro e resiliente.

b) Para que um software seja considerado seguro, é suficiente que possua certa resiliência, ou seja, que, ao ser intencionalmente forçado a falhar por agentes mal-intencionados, ele possa retornar ao seu estado inicial sem realizar operações não planejadas.

c) A fase de requisitos de software é de pouca ou nenhuma importância para a construção de software seguro, já que se concentra nos requisitos funcionais e não funcionais da aplicação. A responsabilidade de manter a segurança do software é do time de segurança da informação.

d) A aplicação de criptografia na informação gerenciada pela aplicação e o uso do SSL/TLS para a proteção da comunicação são suficientes para manter as aplicações web protegidas. Com essas medidas de proteção, ataques a aplicações como os descritos em guias como o OWASP Top 10 tornam-se secundários.

e) Um ciclo de desenvolvimento de software seguro é aquele em que atividades de segurança são aplicadas ao longo das etapas de requisitos, projeto, codificação, testes, operação e descarte. Essas atividades podem incluir revisão de segurança no projeto de arquitetura da aplicação e no respectivo código fonte, além de testes com foco em segurança, realizados pela equipe de garantia da qualidade.

Q5) [FGV MPE AL 2018] Na prática de programação segura, a ação que pode ser adotada para mitigar ataques que exploram a inserção de comandos em campos de formulários dos sistemas, especialmente em sistemas web, como o ataque de "SQL Injection", é descrita como

a)codificação dos dados de entrada.

b)criptografia dos dados de entrada.

c)autenticação de usuários.

d)controle de acesso dos dados de entrada.

e)validação dos dados de entrada.

Q6) [CESPE CNJ 2013] Acerca de conceitos relacionados ao desenvolvimento de software seguro e segurança para web services, julgue os itens subsecutivos.

O SDL é um processo de desenvolvimento de software seguro, que envolve a adição de produtos e atividades, como o desenvolvimento de modelos de ameaças.

Q5) [FGV MPE AL 2018] Na prática de programação segura, a ação que pode ser adotada para mitigar ataques que exploram a inserção de comandos em campos de formulários dos sistemas, especialmente em sistemas web, como o ataque de "SQL Injection", é descrita como

a)codificação dos dados de entrada.

b)criptografia dos dados de entrada.

c)autenticação de usuários.

d)controle de acesso dos dados de entrada.

e)validação dos dados de entrada.

Q6) [CESPE CNJ 2013] Acerca de conceitos relacionados ao desenvolvimento de software seguro e segurança para web services, julgue os itens subsecutivos.

O SDL é um processo de desenvolvimento de software seguro, que envolve a adição de produtos e atividades, como o desenvolvimento de modelos de ameaças. CERTO.

Q7) [CESPE FUNPRESP 2016] Julgue o próximo item, relativo a desenvolvimento e qualidade de software.

No que se refere a softwares, uma programação segura deve dispor de mecanismos que, em quaisquer circunstâncias, rejeitem a entrada de dados que contenham caracteres considerados potencialmente perigosos.

Q8) [FAUGRS BANRISUL 2018] Considere as falhas de segurança abaixo.

I - Integer overflow II - Injeção de comandos III - Vazamento de informações sensíveis IV - Execução remota de comandos V - Ataques de força bruta

Quais podem ser evitadas por meio de boas práticas de programação segura?

a)Apenas II e IV.

b)Apenas I, III e V.

c)Apenas I, II, III e IV.

d)Apenas I, II, IV e V.

e)I, II, III, IV e V.

Q7) [CESPE FUNPRESP 2016] Julgue o próximo item, relativo a desenvolvimento e qualidade de software.

No que se refere a softwares, uma programação segura deve dispor de mecanismos que, em quaisquer circunstâncias, rejeitem a entrada de dados que contenham caracteres considerados potencialmente perigosos. ERRADO.

Q8) [FAUGRS BANRISUL 2018] Considere as falhas de segurança abaixo.

I - Integer overflow II - Injeção de comandos III - Vazamento de informações sensíveis IV - Execução remota de comandos V - Ataques de força bruta

Quais podem ser evitadas por meio de boas práticas de programação segura?

a)Apenas II e IV.

b)Apenas I, III e V.

c)Apenas I, II, III e IV.

d)Apenas I, II, IV e V.

e)I, II, III, IV e V.

Q9) [FCC TCE PR 2011] Sobre o processo de revisão de código é correto afirmar:

a)Desenvolver software, adotando uma prática de revisão de fato, eleva o número de defeitos detectados nas fases iniciais do ciclo de vida, o que auxilia o cumprimento de custo e prazo acordados com o cliente, assim como a aderência aos requisitos definidos e consequente satisfação com o produto entregue.

b)Revisões constantes sempre impedem uma padronização do código, dificultando uma posterior manutenção.

c)O esforço gasto para a execução das etapas de revisão é pequeno, podendo chegar no máximo a 5% do desenvolvimento de um software.

d)É necessário alocar funcionários com alto conhecimento técnico para a realização da revisão de códigos desenvolvidos por programadores inexperientes.

e)A análise dinâmica de código é um método que visa revisar um código fonte apenas no final de cada fase do projeto para buscar por vulnerabilidades e potenciais defeitos, garantindo que os desenvolvedores programem na próxima etapa de forma correta e segura.

Q9) [FCC TCE PR 2011] Sobre o processo de revisão de código é correto afirmar:

a)Desenvolver software, adotando uma prática de revisão de fato, eleva o número de defeitos detectados nas fases iniciais do ciclo de vida, o que auxilia o cumprimento de custo e prazo acordados com o cliente, assim como a aderência aos requisitos definidos e consequente satisfação com o produto entregue.

b)Revisões constantes sempre impedem uma padronização do código, dificultando uma posterior manutenção.

c)O esforço gasto para a execução das etapas de revisão é pequeno, podendo chegar no máximo a 5% do desenvolvimento de um software.

d)É necessário alocar funcionários com alto conhecimento técnico para a realização da revisão de códigos desenvolvidos por programadores inexperientes.

e)A análise dinâmica de código é um método que visa revisar um código fonte apenas no final de cada fase do projeto para buscar por vulnerabilidades e potenciais defeitos, garantindo que os desenvolvedores programem na próxima etapa de forma correta e segura.

Q10) [CESPE CGE CE 2019] Considerando que a ferramenta SonarQube permite analisar código gerado na linguagem Java e em outras linguagens, assinale a opção que indica erro que, na análise do código de um sistema de recursos humanos, pode ser identificado como configuração default dessa ferramenta.

a) método de conexão do banco de dados com tratamento de erro/exceção ou falha da conexão

b) método de retorno de inteiro com tratamento de retorno de nulo

c) método de retorno de nome da base de dados com tratamento de exceção caso não se estabeleça a conexão ou não se encontrem dados

d) método projetado para verificar uma conexão de banco de dados e tratar exceção, caso ocorra

e) método projetado para verificar a igualdade e que sempre retorna falso

Q11) [CESPE TCE SC 2016] Com relação à programação segura e à ferramenta Apache Maven, julgue o item seguinte.

Utilizar validação de entrada e codificação de saída, assegurar a abordagem de metacaracteres e evitar consultas parametrizadas fortemente tipificadas são ações compatíveis com as práticas de programação segura relacionadas a bases de dados.

Q10) [CESPE CGE CE 2019] Considerando que a ferramenta SonarQube permite analisar código gerado na linguagem Java e em outras linguagens, assinale a opção que indica erro que, na análise do código de um sistema de recursos humanos, pode ser identificado como configuração default dessa ferramenta.

a) método de conexão do banco de dados com tratamento de erro/exceção ou falha da conexão

b) método de retorno de inteiro com tratamento de retorno de nulo

c) método de retorno de nome da base de dados com tratamento de exceção caso não se estabeleça a conexão ou não se encontrem dados

d) método projetado para verificar uma conexão de banco de dados e tratar exceção, caso ocorra

e) método projetado para verificar a igualdade e que sempre retorna falso

Q11) [CESPE TCE SC 2016] Com relação à programação segura e à ferramenta Apache Maven, julgue o item seguinte.

Utilizar validação de entrada e codificação de saída, assegurar a abordagem de metacaracteres e evitar consultas parametrizadas fortemente tipificadas são ações compatíveis com as práticas de programação segura relacionadas a bases de dados. ERRADO.

Q12) [IADES PC DF 2016] A Runtime Application Self-Protection (RASP) é uma tecnologia acoplada a uma aplicação e que pode detectar e prevenir ataques, em tempo real, pela inspeção da pilha de execução de uma aplicação. Além disso, o Web Application Firewall (WAF) é um equipamento que inspeciona o tráfego HTTP em busca de problemas relacionados à requisição e resposta de aplicações web. A respeito das técnicas de defesa utilizadas em aplicações e do respectivo impacto no ciclo de desenvolvimento seguro de software, assinale a alternativa correta.

a)Medidas de proteção, como o RASP e WAF, são necessárias em razão da incapacidade dos desenvolvedores de produzir código livre de bugs de segurança, e são suficientes para reduzir a superfície de ataque das aplicações.

b) Vulnerabilidade ou falhas em aplicações são bugs que impactam a segurança do software. Desse modo, produzir aplicações livres de defeitos relacionados aos requisitos funcionais e não funcionais pode garantir que o software seja seguro.

c) A utilização de metodologia de validação de entrada e saída de dados com foco em segurança nas aplicações, se bem implementada, pode auxiliar na mitigação de ataques de injeção de comando como o Cross-Site Scripting (XSS) persistente, injeção SQL e injeção LDAP.

d)Equipamentos como o WAF e APIs de validação de entrada e saída estão sujeitos a ataques com strings ofuscadas e são medidas ineficientes de proteção das aplicações.

e)A proteção de dados de privacidade individuais é de baixa importância quando se fala de um ciclo de desenvolvimento de software seguro, pois essa abordagem está preocupada apenas com a mitigação de vulnerabilidades técnicas no código fonte de uma aplicação.

Q12) [IADES PC DF 2016] A Runtime Application Self-Protection (RASP) é uma tecnologia acoplada a uma aplicação e que pode detectar e prevenir ataques, em tempo real, pela inspeção da pilha de execução de uma aplicação. Além disso, o Web Application Firewall (WAF) é um equipamento que inspeciona o tráfego HTTP em busca de problemas relacionados à requisição e resposta de aplicações web. A respeito das técnicas de defesa utilizadas em aplicações e do respectivo impacto no ciclo de desenvolvimento seguro de software, assinale a alternativa correta.

a)Medidas de proteção, como o RASP e WAF, são necessárias em razão da incapacidade dos desenvolvedores de produzir código livre de bugs de segurança, e são suficientes para reduzir a superfície de ataque das aplicações.

b) Vulnerabilidade ou falhas em aplicações são bugs que impactam a segurança do software. Desse modo, produzir aplicações livres de defeitos relacionados aos requisitos funcionais e não funcionais pode garantir que o software seja seguro.

c) A utilização de metodologia de validação de entrada e saída de dados com foco em segurança nas aplicações, se bem implementada, pode auxiliar na mitigação de ataques de injeção de comando como o Cross-Site Scripting (XSS) persistente, injeção SQL e injeção LDAP.

d)Equipamentos como o WAF e APIs de validação de entrada e saída estão sujeitos a ataques com strings ofuscadas e são medidas ineficientes de proteção das aplicações.

e)A proteção de dados de privacidade individuais é de baixa importância quando se fala de um ciclo de desenvolvimento de software seguro, pois essa abordagem está preocupada apenas com a mitigação de vulnerabilidades técnicas no código fonte de uma aplicação.

Q13) [CESPE STJ 2018] Julgue o item a seguir, acerca de eMAG, sistemas de controle de versão e SonarQube.

Uma issue gerada pelo SonarQube com severidade CRITICA requer a imediata correção do código.

Q14) [CESPE TCE SC 2016] A respeito da análise estática de código-fonte em Clean Code e SonarQube, julgue o item subsecutivo.

Um dos modos de análise de código-fonte constante no SonarQube é o publish, que analisa completamente o código e o envia para o servidor que irá processá-lo e salvar os resultados no banco de dados.

Q15) [CESPE TCU 2016] Com referência às ferramentas de desenvolvimento de aplicações, inclusive para ambiente web e dispositivos móveis, julgue o item a seguir.

Uma característica positiva da ferramenta SonarQube, quando utilizada para realizar a análise estática de código-fonte, é a conveniência de instalação e utilização em dispositivos móveis.

Q13) [CESPE STJ 2018] Julgue o item a seguir, acerca de eMAG, sistemas de controle de versão e SonarQube.

Uma issue gerada pelo SonarQube com severidade CRITICA requer a imediata correção do código. ERRADO.

Q14) [CESPE TCE SC 2016] A respeito da análise estática de código-fonte em Clean Code e SonarQube, julgue o item subsecutivo.

Um dos modos de análise de código-fonte constante no SonarQube é o publish, que analisa completamente o código e o envia para o servidor que irá processá-lo e salvar os resultados no banco de dados. ERRADO.

Q15) [CESPE TCU 2016] Com referência às ferramentas de desenvolvimento de aplicações, inclusive para ambiente web e dispositivos móveis, julgue o item a seguir.

Uma característica positiva da ferramenta SonarQube, quando utilizada para realizar a análise estática de código-fonte, é a conveniência de instalação e utilização em dispositivos móveis. ERRADO.

Q16) [CESPE TCE SC 2016] Com relação à programação segura e à ferramenta Apache Maven, julgue o item seguinte.

Utilizar validação de entrada e codificação de saída, assegurar a abordagem de metacaracteres e evitar consultas parametrizadas fortemente tipificadas são ações compatíveis com as práticas de programação segura relacionadas a bases de dados.

Q17) [CESPE TCE PA 2016] Acerca de análise estática de código-fonte, uma das práticas que verifica a qualidade do código e pode ser realizada antes da execução do software, julgue o próximo item.

A ferramenta SonarQube permite analisar a qualidade dos códigos-fontes que envolvem linguagens de computador e de dispositivos móveis e abrange categorias como padrões de codificação, testes e identificação de erros.

Q16) [CESPE TCE SC 2016] Com relação à programação segura e à ferramenta Apache Maven, julgue o item seguinte.

Utilizar validação de entrada e codificação de saída, assegurar a abordagem de metacaracteres e evitar consultas parametrizadas fortemente tipificadas são ações compatíveis com as práticas de programação segura relacionadas a bases de dados. ERRADO.

Q17) [CESPE TCE PA 2016] Acerca de análise estática de código-fonte, uma das práticas que verifica a qualidade do código e pode ser realizada antes da execução do software, julgue o próximo item.

A ferramenta SonarQube permite analisar a qualidade dos códigos-fontes que envolvem linguagens de computador e de dispositivos móveis e abrange categorias como padrões de codificação, testes e identificação de erros. CERTO.