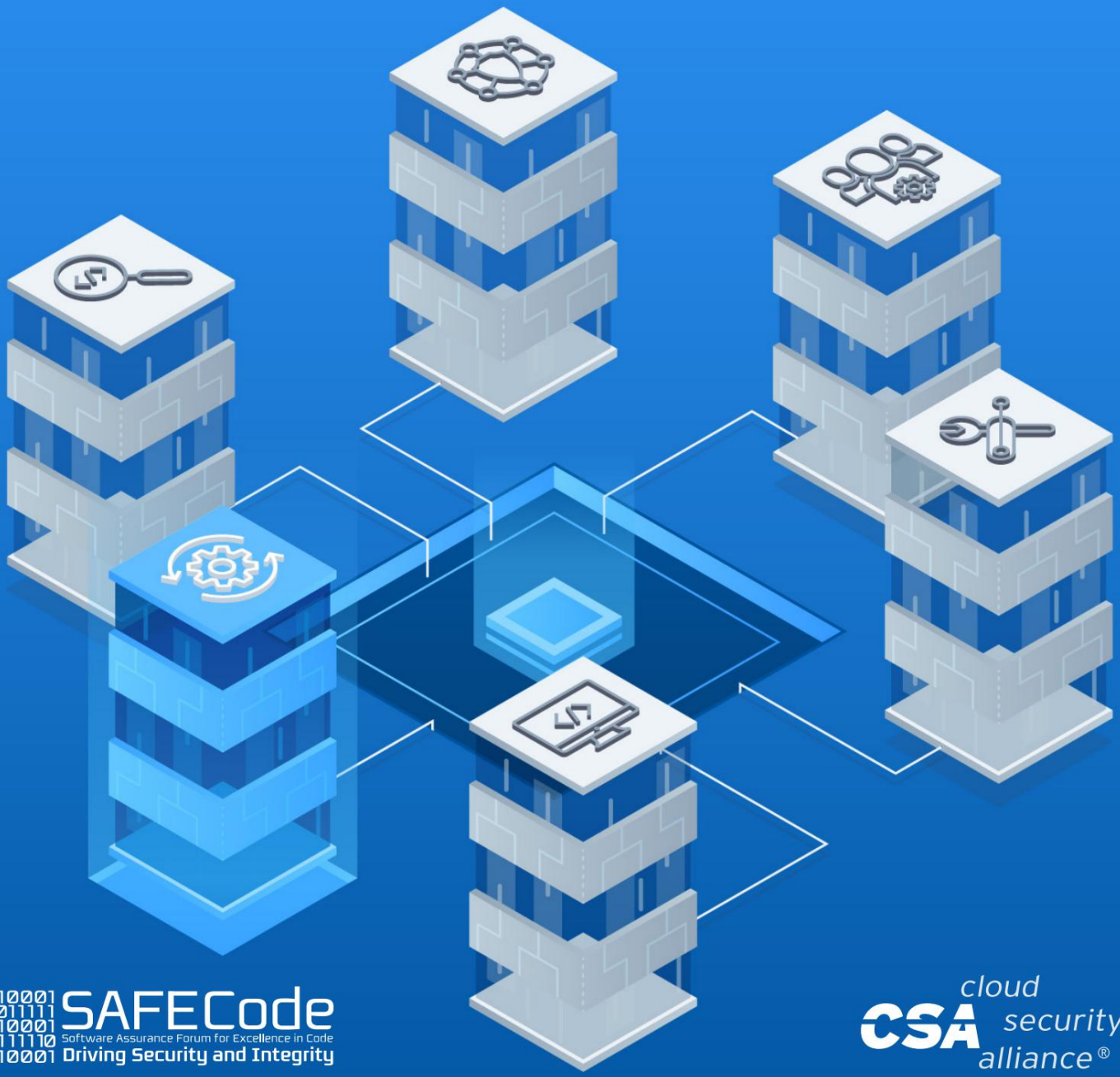
Machine Translated by Google

Os seis pilares do DevSecOps:

Automação



SAFECode
Software Assurance Forum for Excellence in Code
Driving Security and Integrity

CSA cloud security alliance®

Machine Translated by Google

O local permanente e oficial do Grupo de Trabalho DevSecOps é https://
cloudsecurityalliance.org/research/working-groups/devsecops/

© 2020 Cloud Security Alliance – Todos os direitos reservados. Você pode baixar, armazenar, exibir em seu computador, visualizar, imprimir e vincular à Cloud Security Alliance em https://cloudsecurityalliance.org sujeito ao seguinte: (a) o rascunho pode ser usado exclusivamente para seu uso pessoal, informativo, uso não-comercial; (b) a minuta não poderá ser modificada ou alterada de forma alguma; (c) o projeto não poderá ser redistribuído; e (d) a marca registrada, direitos autorais ou outros avisos não podem ser removidos. Você pode citar partes do rascunho conforme permitido pelas disposições de uso justo da Lei de Direitos Autorais dos Estados Unidos, desde que você atribua as partes à Cloud Security Alliance.

Machine Translated by Google

Agradecimentos

Autores principais:

Souheil Moghnie
Theodoro Niedzialkowski
Sam Sehgal

Contribuinte:

Michael Rosa

Analistas CSA:

Sean Heide

Agradecimentos especiais:

Ankur Gargi
Raj Handa
Manuel Ifland
João Martinho
Kamran Sadique
Charanjeet Singh
Altaz Valani

© Copyright 2020, Cloud Security Alliance. Todos os direitos reservados.

Machine Translated by Google

Índice

© Copyright 2020, Cloud Security Alliance. Todos os direitos reservados.

Machine Translated by Google

Prefácio

A Cloud Security Alliance e a SAFECode estão profundamente comprometidas em melhorar os resultados de segurança de software. O artigo *Six Pillars of DevSecOps,* publicado em agosto de 2019, fornece um conjunto de métodos de alto nível e soluções implementadas com sucesso usadas por seus autores para construir software com velocidade e com o mínimo de bugs relacionados à segurança. Esses seis pilares são:

- Pilar 1: Responsabilidade Coletiva (Publicado em 20/02/2020)
- Pilar 2: Treinamento e Integração de Processos
- Pilar 3: Implementação Pragmática
- Pilar 4: Unindo Conformidade e Desenvolvimento
- Pilar 5: Automação
- Pilar 6: Medir, Monitorar, Reportar e Agir

As soluções de sucesso que sustentam cada um destes pilares são temas de um conjunto muito mais detalhado de publicações conjuntas da Cloud Security Alliance e da SAFECode. Este artigo é a segunda dessas publicações subsequentes.

© Copyright 2020, Cloud Security Alliance. Todos os direitos reservados.

Machine Translated by Google

Introdução

0.1 Plano de fundo

O pilar DevSecOps Automation defende o uso da automação de segurança para alcançar segurança reflexiva, especificamente, a implementação de uma estrutura para automação de segurança e execução programática e monitoramento de controles de segurança para identificar, proteger, detectar, responder e se recuperar de ameaças cibernéticas.

A automação é um componente crítico do DevSecOps porque permite a eficiência dos processos, permitindo que desenvolvedores, infraestrutura e equipes de segurança da informação se concentrem na entrega de valor, em vez de repetir esforços e erros manuais com resultados complexos.

Este documento se concentra em uma abordagem de automação de segurança baseada em riscos que encadeia ações de segurança automatizadas durante todo o ciclo contínuo de implantação de desenvolvimento de software. Exemplos de atividades que podem ser automatizadas incluem verificação de vulnerabilidades de aplicativos, hosts e contêineres.

As equipes de DevOps que utilizam as melhores práticas, incluindo integração contínua, entrega contínua e infraestrutura como código, são ágeis, com a capacidade de atender aos requisitos do usuário de forma dinâmica, lançar recursos de forma incremental e entregar em um ritmo mais rápido.

Para integrar os controles de segurança da informação neste processo, os recursos de segurança automatizados são cruciais para fornecer feedback oportuno e significativo.

A complexidade da infraestrutura em nuvem hoje significa que pequenas alterações no código podem levar a um impacto downstream desproporcional; portanto, as verificações de segurança precisam ser integradas em todo o ciclo de vida de desenvolvimento e implantação de software, através do design, implementação, teste e lançamento, e monitoradas na produção.

Em alinhamento com o pilar da Abordagem Pragmática de segurança reflexiva, a adoção de uma automação provisória e modesta dos recursos de segurança já permitirá feedback rápido e poderá potencialmente eliminar classes inteiras de risco, por exemplo, verificação de contêineres para garantir a proteção do sistema operacional ou análise de composição de software para vulnerabilidades comuns conhecidas. e Exposições (CVEs).

0.2 Objetivo

Este documento fornece uma estrutura para permitir que a automação integre de forma transparente a segurança no ciclo de vida de desenvolvimento de software por meio de:

- Permitir um fluxo rápido e recíproco de informações relacionadas à segurança para que as equipes de DevOps criem e validem código seguro desde o projeto de maneira contínua, em vez de adiar a segurança para um impedimento de estágio único na entrega.
- Permitir uma abordagem equilibrada ao desenvolvimento de software e às necessidades de segurança, de modo que a eficiência da entrega seja melhorada através da integração automatizada.

© Copyright 2020, Cloud Security Alliance. Todos os direitos reservados.

Machine Translated by Google

0,3 Público

O público-alvo deste documento inclui aqueles envolvidos nas funções de gestão e operacionais de risco, segurança da informação e tecnologia da informação. Isso inclui o C-suite (CISO, CIO, CTO, CRO, COO, CEO) e especialmente os indivíduos envolvidos nas seguintes áreas funcionais: DevSecOps, automação, DevOps, garantia de qualidade, segurança da informação, governança, gestão de riscos, gerenciamento de mudanças e conformidade.

1. Escopo

Este documento descreve o pilar de automação do DevSecOps: a necessidade de automação de segurança, técnicas de automação de testes de segurança e mecanismos para alcançá-la.

Este documento fornece especificamente uma estrutura holística para facilitar a automação da segurança dentro do DevSecOps e as melhores práticas na automação de controles de segurança e fornece esclarecimentos sobre equívocos comuns sobre testes de segurança no contexto do DevSecOps.

2. Referências Normativas

Os documentos a seguir são mencionados no texto de forma que parte ou todo o seu conteúdo constitua requisitos deste documento. Para referências datadas, aplica-se apenas a edição citada. Para referências não datadas, aplica-se a edição mais recente do documento referenciado (incluindo quaisquer alterações).

- ISO/IEC 27000:2018, Tecnologia da informação - Técnicas de segurança - Segurança da informação sistemas de gestão - Visão geral e vocabulário
- Gerenciamento de segurança da informação CSA por meio de segurança reflexiva
- CSA Os Seis Pilares do DevSecOps

3. Termos e Definições

Para os fins deste documento, aplicam-se os termos e definições fornecidos na ISO 27000, CSA Information Security Management through Reflexive Security e os seguintes.

3.1 Análise de Composição de Software (SCA)

Testes de segurança que analisam o código-fonte do aplicativo ou código compilado para componentes de software com vulnerabilidades conhecidas

Nota 1 de entrada: componentes de software na análise de composição de software podem incluir código aberto, bibliotecas e código comum.

Nota 2 de entrada: vulnerabilidades conhecidas podem ser descobertas por meio de bancos de dados de vulnerabilidades, como o CVE.

© Copyright 2020, Cloud Security Alliance. Todos os direitos reservados.

Machine Translated by Google

3.2 Teste de segurança de aplicativos estáticos (SAST)

Testes de segurança que analisam o código-fonte do aplicativo em busca de vulnerabilidades e lacunas de software em relação às práticas recomendadas

Nota 1 de entrada: A análise estática pode ser realizada em vários ambientes, incluindo o IDE do desenvolvedor, código-fonte e binários.

Nota 2 de entrada: Também chamado de “teste de caixa branca”

3.3 Teste Dinâmico de Segurança de Aplicativos (DAST)

Teste de segurança que analisa um aplicativo em execução, exercitando a funcionalidade do aplicativo e detectando vulnerabilidades com base no comportamento e na resposta do aplicativo

Nota 1 de entrada: Também chamado de “teste de caixa preta”

3.4 Teste Interativo de Segurança de Aplicativos (IAST)

Componente de software implantado com um aplicativo que avalia o comportamento do aplicativo e detecta a presença de vulnerabilidades em um aplicativo que está sendo exercitado em cenários de teste realistas

3.5 Proteção de segurança de aplicativos em tempo de execução (RASP)

Tecnologia de segurança implantada no aplicativo alvo em produção para detectar, alertar e bloquear ataques

Nota 1 de entrada: Semelhante a um WAF, mas instrumentado dentro do aplicativo

3.6 Firewall de Aplicativo Web (WAF)

Firewall de aplicativo que monitora, alerta e bloqueia ataques inspecionando o tráfego HTTP

3.7 Gerenciamento de Postura de Segurança na Nuvem (CSPM)

Tecnologia de segurança que pode descobrir, avaliar e resolver configurações incorretas da infraestrutura em nuvem vulneráveis a ataques

3.8 Monitoramento e conformidade de segurança em nuvem

Tecnologia de segurança que monitora servidores virtuais e avalia dados, aplicativos e infraestrutura em busca de riscos de segurança

3.9 Modelagem de Ameaças

Metodologia para identificar e compreender ameaças que impactam um recurso ou conjunto de recursos

© Copyright 2020, Cloud Security Alliance. Todos os direitos reservados.

Machine Translated by Google

Nota de entrada: Metodologias comuns de modelagem de ameaças incluem STRIDE (Spoofing, Adulteração, Repúdio, Divulgação de Informações, Negação de Serviço, Elevação de Privilégio) e OCTAVE (Avaliação de Ameaças, Ativos e Vulnerabilidades Operacionalmente Críticas).

3.10 Revisão Manual do Código de Segurança

Processo humano de leitura do código-fonte para identificar problemas de segurança

3.11 Pipeline de entrega de software

Conjunto de processos automatizados usados para entregar software desde a concepção até a implantação

3.12 Pipeline de entrega de software CSA DevSecOps (CDDP)

Pipeline de entrega de software habilitado para segurança alinhado aos princípios DevSecOps

4. Pipeline de entrega de software CSA DevSecOps

4.1 Geral

Cada software passa pelos estágios de concepção, design, desenvolvimento, construção e teste antes da implantação em ambientes de produção. No pipeline moderno de entrega de software, as ferramentas são altamente automatizadas para proteger os pontos de verificação do estágio. Em cada etapa do pipeline, são realizadas verificações automatizadas para evitar que problemas de qualidade passem para as próximas etapas.

Vários processos estão envolvidos em um pipeline de entrega de software, incluindo desenvolvimento de software, gerenciamento de mudanças, gerenciamento de configuração e gerenciamento de serviços, potencialmente com os conceitos de integração contínua, entrega contínua e monitoramento contínuo aplicados. As ferramentas usadas pelo desenvolvimento e pelas operações incluem controle de origem, construção, integração contínua, conteinerização, gerenciamento de configuração, orquestração e monitoramento.

A integração dos testes de segurança no pipeline de desenvolvimento de software exige que as atividades de segurança sejam totalmente automatizadas e interligadas com as ferramentas e processos nativos já empregados pelas equipes no pipeline de entrega. As atividades exigidas pelos pontos de verificação, mas cujo resultado requer intervenção manual, denominadas testes assíncronos neste documento, devem ser especialmente consideradas para que os benefícios dos pontos de verificação de segurança automatizados não sejam perdidos para elas.

© Copyright 2020, Cloud Security Alliance. Todos os direitos reservados.

Machine Translated by Google

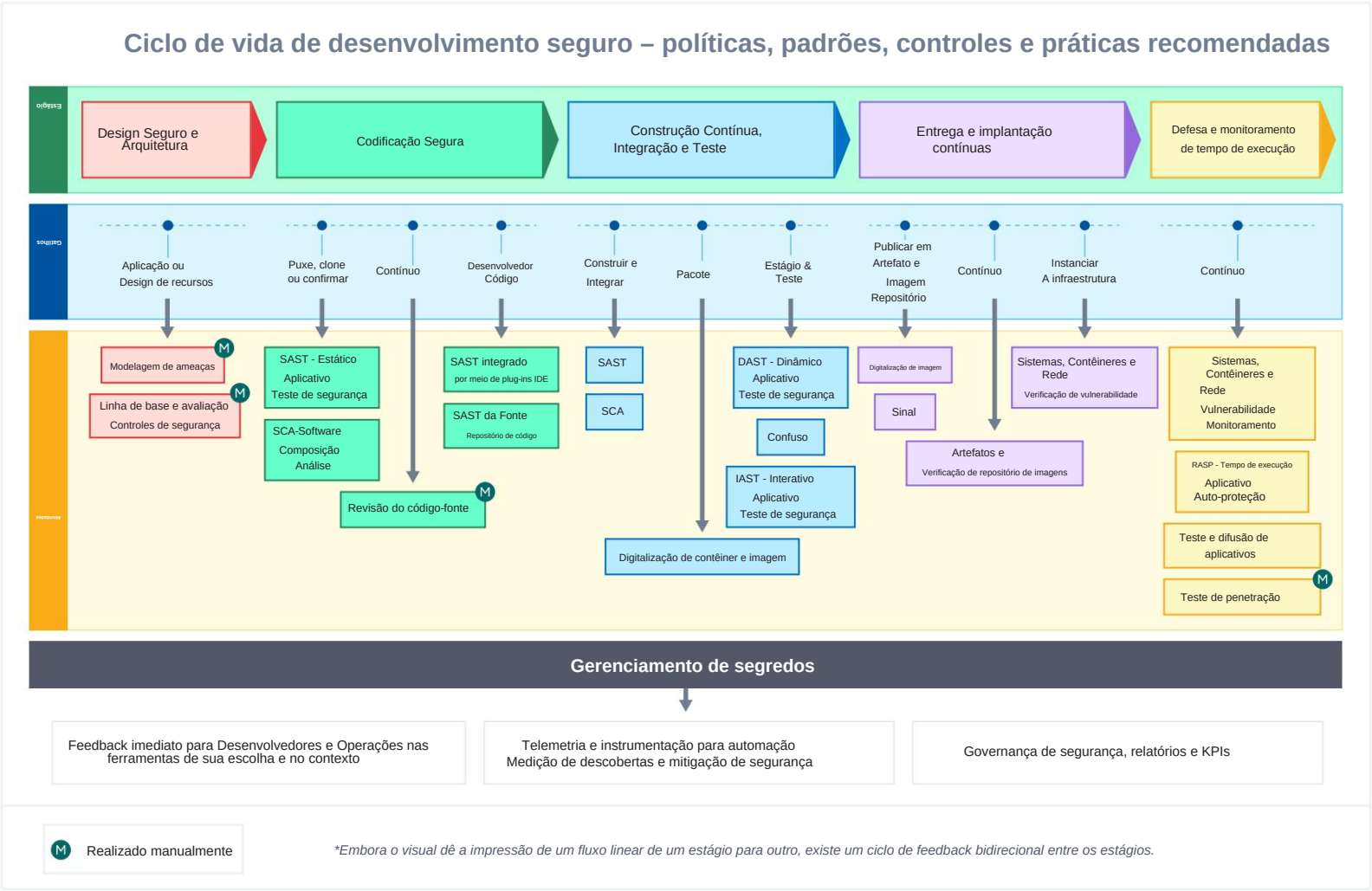


Figura 1: O pipeline de entrega CSA DevSecOps

4.2 Estrutura

4.2.1 Geral

Considera-se que um pipeline de entrega de software habilitado para segurança tem três níveis de granularidade:

- Estágios
- Gatilhos e pontos de verificação
- Atividades

© Copyright 2020, Cloud Security Alliance. Todos os direitos reservados.

Machine Translated by Google

4.2.2 Etapas

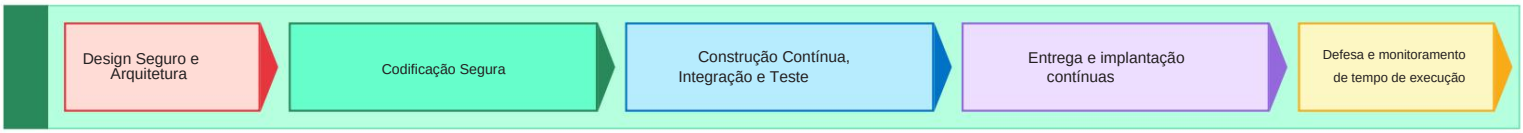


Figura 1-1: Etapas do pipeline de entrega

Os estágios representam um nível de maturidade incremental do produto entregável. Durante a concepção e projeto de arquitetura, o nível de maturidade é baixo. Quando o software está pronto para entrega e implantação contínuas, ele está claramente mais maduro.

Cada entrega de software deve passar por estágios desde o design, codificação até o teste antes de poder ser finalmente implantado na produção. Isso independe da metodologia de desenvolvimento de software (Waterfall, Agile).

Há uma oportunidade em todas as fases de incorporar e automatizar controles de segurança apropriados.

Este documento identifica as seguintes etapas distintas:

1. Design e arquitetura seguros
2. Codificação segura (IDE do desenvolvedor e repositório de código)
3. Construção, integração e teste contínuos
4. Entrega e implantação contínuas
5. Monitoramento contínuo e defesa em tempo de execução

4.2.3 Gatilhos

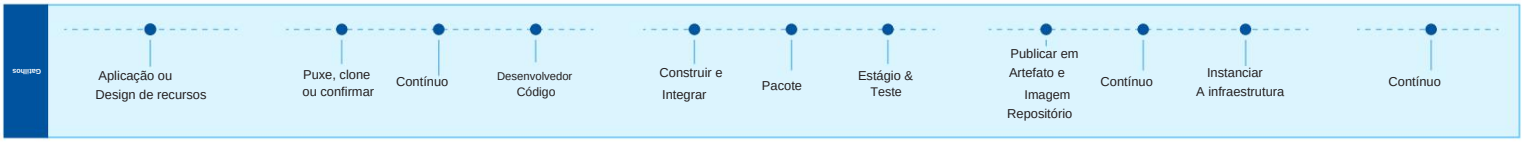


Figura 1-2: Gatilhos no pipeline de entrega

Os gatilhos e os pontos de verificação (veja a Figura 1-2) representam transições dentro dos estágios. Quando uma condição de acionamento é atendida, uma ou mais atividades de segurança são ativadas e os resultados dessas atividades determinam se os requisitos desejados do ponto de verificação são atendidos.

Se o resultado das atividades de segurança atender aos requisitos desejados, a entrega será transferida para o próximo ponto de verificação (ou para o próximo estágio, se o ponto de verificação for o último). Caso contrário, a entrega permanece e não é permitido avançar para o próximo ponto de verificação.

EXEMPLO 1: A arquitetura de um novo recurso pode desencadear o início de um exercício de modelagem de ameaças, que pode ser usado para identificar controles de segurança gerais ou específicos do setor aplicáveis a esse recurso.

© Copyright 2020, Cloud Security Alliance. Todos os direitos reservados.

Machine Translated by Google

EXEMPLO 2: Uma solicitação de confirmação de código ou pull/merge por um desenvolvedor pode acionar automaticamente uma varredura de código-fonte e um processo de revisão de código.

EXEMPLO 3: Enviar uma imagem de contêiner para um registro de contêiner pode acionar uma verificação de vulnerabilidade antes que a imagem fique disponível no registro.

Existem dois tipos de gatilhos:

- Gatilhos baseados em alterações: esses gatilhos são acionados por eventos de mudança, como um envio de código.
- Gatilhos recorrentes: esses gatilhos são acionados por eventos de tempo, como um cronômetro diário.

Gatilhos baseados em alterações são usados para proteger transições de pontos de verificação. Eles geralmente são usados para verificações de segurança de curta duração que podem ser executadas de maneira baseada em eventos.

Os gatilhos recorrentes geralmente são usados para verificações de segurança que são executadas por períodos mais longos, como aquelas que envolvem processamento em lote ou integração entre componentes.

EXEMPLO 4: Um gatilho recorrente é preferido para varredura em lote de vários repositórios de código-fonte em busca de segredos perdidos.

4.2.4 Atividades

Figura 1-3: Atividades de segurança no pipeline de entrega

Atividades são processos individuais que tomam a entrega como entrada e produzem um resultado positivo ou negativo. Se uma entrega atender aos critérios da atividade, o resultado é considerado positivo. Caso contrário, o resultado é negativo.

Especificamente, cada atividade representa a implementação de um controle de segurança – uma tarefa de teste ou verificação realizada para detectar vulnerabilidades de segurança em código, biblioteca, aplicativos ou no ambiente em que um aplicativo será executado.

As atividades são independentes de estágios e gatilhos: algumas atividades podem ser executadas em todos os estágios ou vinculadas a um único estágio. Vários gatilhos podem depender do mesmo resultado de atividade. Descrição detalhada e diretrizes de cada atividade de segurança estão disponíveis nos documentos referenciados.

© Copyright 2020, Cloud Security Alliance. Todos os direitos reservados.

Machine Translated by Google

4.3 Configuração e Manutenção do Pipeline

Para oferecer suporte ao DevSecOps, o pipeline de entrega deve ser integrado às ferramentas e processos nativos já usados pelas equipes de DevOps. A segurança precisa fazer parte do processo normal de lançamento para minimizar o atrito e dar suporte a ciclos de lançamento rápidos e eficientes.

De uma perspectiva pragmática de segurança, os recursos de teste de segurança de pipeline devem ser introduzidos de forma incremental, com recursos de alto valor e baixo impacto adicionados primeiro. As verificações de segurança das ferramentas de qualidade de código ou de gerenciamento de biblioteca1 já utilizadas podem gerar ganhos rápidos com impacto operacional mínimo ou recursos adicionais (identificação de um fornecedor, aquisição de uma nova tecnologia e instrumentação).

Ao introduzir um novo recurso de teste, por exemplo, Static Application Security Testing (SAST), a equipe de DevOps, por exemplo – deve se concentrar primeiro em um tipo de descoberta específico e ajustar a ferramenta para evitar falsos positivos e garantir a cobertura máxima do código (eliminando falsos negativos). A equipe agora pode realizar análises de causa raiz em vulnerabilidades perdidas e falsos positivos para obter a eficiência ideal dos testes automatizados antes de adicionar um novo tipo ou recursos de descoberta.

Um grande número de falsos positivos pode sobrecarregar as equipes de DevOps e prejudicar a credibilidade das ferramentas de segurança.

Uma boa estratégia é “mudar a segurança para a esquerda enquanto acelera para a direita”. Os controles e atividades de segurança automatizados não substituem as políticas, padrões e procedimentos SDLC existentes de uma organização. Em vez disso, a automação aumenta os recursos existentes para proteger processos e ativos. As organizações devem começar automatizando controles e atividades que estão sendo executadas manualmente.

5. Pipelines com prioridade de risco

5.1 Geral

Nem todos os aplicativos têm os mesmos perfis de risco. Para priorizar efetivamente os recursos e encontrar o equilíbrio com a capacidade de entrega, é benéfico implementar um mecanismo para fornecer mais escrutínio às entregas de aplicativos de alto risco, ao mesmo tempo que acelera as entregas de aplicativos com um histórico estável.

Uma abordagem baseada em risco deve ser usada para determinar o rigor geral dos testes de segurança envolvidos na implantação de uma compilação. Geralmente, há três tipos de fatores de risco a serem considerados, descritos abaixo.

1 https://www.csoonline.com/article/3398485/28-devsecops-tools-for-baking-security-in-to-the-development-process.html

© Copyright 2020, Cloud Security Alliance. Todos os direitos reservados.

Machine Translated by Google

5.2 Fatores de Risco

5.2.1 Risco de Aplicação

O risco inerente atribuído à aplicação, independentemente da exposição técnica, incluindo a consideração dos dados que processa e do seu contexto de utilização

EXEMPLO: Um aplicativo que lida com dados bancários ou de saúde pode ser considerado de “alto risco”, enquanto um aplicativo que lida com PII e cartões de crédito pode ser considerado de “médio risco”. Aqueles que lidam com conteúdo estático ou informações públicas podem ser considerados de “baixo risco”.

5.2.2 Risco de Mudança Proposta

O risco introduzido pela própria mudança, considerando o impacto potencial da mudança na segurança, os dados que a mudança afeta e a natureza da mudança

EXEMPLO: Mudanças que afetam os principais componentes de segurança, como autenticação ou gerenciamento de acesso, mudanças que introduzem novos recursos ou tecnologias e mudanças que envolvem rearquitetura têm maior probabilidade de impactar a postura de segurança do aplicativo e, portanto, devem ser testadas de forma mais completa.

5.2.3 Histórico de risco de confiabilidade do pipeline e suas entregas

O risco atribuído ao histórico de confiabilidade e integridade do pipeline de entrega e da aplicação em questão.

EXEMPLO: Um pipeline de entrega estável com um histórico de entregas seguras é compreensivelmente menos arriscado do que um pipeline de entrega instável com um histórico de entregas vulneráveis. Um aplicativo que teve versões estáveis e seguras é menos arriscado do que outro com uma série de falhas de segurança.

5.3 Priorização da configuração do pipeline

As configurações do pipeline de entrega podem ser priorizadas de acordo com uma série de fatores de risco. Cada pipeline de entrega e aplicação é único; na implementação é necessário considerar se o mecanismo de diferenciação de risco é apropriado e suficiente para um caso de uso.

A existência de tratamento diferenciado para pipelines de entrega proporciona um incentivo gradual para aplicações desenvolvidas e implantadas com segurança, levando a prazos de entrega mais rápidos.

© Copyright 2020, Cloud Security Alliance. Todos os direitos reservados.

Machine Translated by Google

Esta cláusula utiliza a metáfora do semáforo com três faixas de processamento de cores diferentes: verde, amarela e vermelha, cada uma oferecendo um nível diferente de escrutínio de segurança:

- Via verde: testes de curta duração, otimizados para entrega contínua
- Via amarela: pode envolver DAST, pequenas atividades fora de banda com tempos de processamento curtos, entrega contínua possível
- Via vermelha: pode envolver IAST, grandes atividades fora de banda com tempos de processamento mais longos, entrega contínua difícil.

EXEMPLO 1: A entrega de aplicativos com classificação baixa em todos os três fatores de risco pode ser considerada de “risco geral baixo” e pode continuar ao longo da “via verde”.

EXEMPLO 2: Uma alteração no código de tratamento de dados pessoais que não afete o compartilhamento de dados pessoais em um aplicativo de baixo risco pode ser considerada de “risco médio” e pode continuar na “via amarela”.

EXEMPLO 3: A entrega de aplicações com registro de vulnerabilidades significativas deveria estar sujeita a maior rigor de segurança utilizando a “via vermelha”.

EXEMPLO 4: Se uma mudança proposta introduzir um novo componente padrão para o manuseio seguro de dados ou exigir que um desenvolvedor seja treinado nas melhores práticas de segurança relevantes, a “via vermelha” seria aplicada.

A análise da causa raiz deve ser realizada quando uma nova vulnerabilidade é detectada para determinar se a causa está relacionada a pessoas, processos ou tecnologia, e o que pode ser reprojetado ou alterado para evitar recorrências.

A tabela abaixo fornece um exemplo de pontuação de alterações que combina esses três fatores para determinar a configuração do pipeline de destino. Os valores numéricos de Alto (3), Médio (2) e Baixo (1) são usados com uma pontuação total de 5 e acima considerada na “faixa amarela” e 7 e acima considerada na “faixa vermelha”. faixa."

	Risco de mudança de risco de aplicativo		Risco de confiabilidade	Pipeline de entrega Configuração
Mudança nº 1	Médio (2)	Baixo (1)	Baixo (1)	Verde (4)
Mudança nº 2	Baixo (1)	Alto (3)	Médio (2)	Amarelo (6)
Mudança #3	Alto (3)	Alto (3)	Alto (3)	Vermelho (9)

Tabela 1: Impacto da construção e pipeline de lançamento resultante

© Copyright 2020, Cloud Security Alliance. Todos os direitos reservados.

Machine Translated by Google

6. Estrutura de atividades do pipeline de entrega

6.1 Geral

Esta seção apresenta uma estrutura (veja a Figura 2) que representa cinco classes de atividades de segurança usadas no pipeline de entrega, bem como diretrizes para atividades dentro dessas classes separadas.

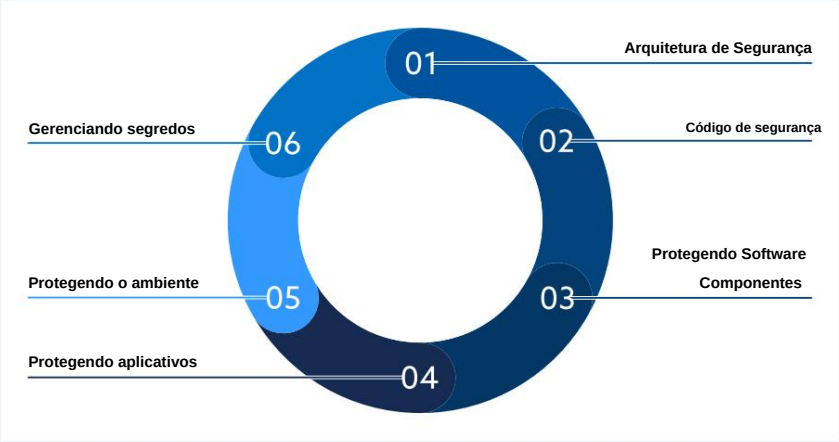


Figura 2: Atividades de segurança nos pipelines de entrega habilitados para segurança

6.2 Protegendo o Projeto

A segurança só pode ser alcançada quando tiver sido concebida. Aplicar medidas de segurança posteriormente é uma receita para o desastre. Falhas de projeto são conhecidas por serem responsáveis por uma grande parte das vulnerabilidades [1].

Embora alguns considerem os processos manuais incompatíveis com as práticas de DevSecOps, não é o caso. Existem atividades básicas de projeto de segurança que exigem inteligência humana, como modelagem de ameaças, que deve ser realizada para garantir o estabelecimento de uma arquitetura segura.

Tradicionalmente, a modelagem de ameaças só pode ser realizada manualmente, mas hoje existem ferramentas que ajudam a automatizar esse processo. Além disso, os modelos de ameaças normalmente não exigem atualizações frequentes, a menos que novos componentes sejam adicionados ou componentes sensíveis à segurança sejam alterados no aplicativo, portanto, as atualizações manuais ainda permanecem um mecanismo viável.

6.3 Código de Segurança

Vulnerabilidades podem surgir de codificação deficiente, independentemente de quão bem pensada e robusta seja a arquitetura de segurança de um aplicativo. Se o código escrito pelos desenvolvedores contiver defeitos de segurança, é apenas uma questão de tempo até que o aplicativo seja violado.

As ferramentas SAST podem ajudar a verificar os códigos-fonte dos resultados e relatar defeitos de segurança (e genéricos), evitando que se manifestem como vulnerabilidades de segurança.

© Copyright 2020, Cloud Security Alliance. Todos os direitos reservados.

Machine Translated by Google

O objetivo do DevSecOps ao automatizar testes de código estático é estreitar o ciclo de feedback para os desenvolvedores, fornecendo alertas de segurança de código o mais cedo possível, no contexto e da forma mais prescritiva possível para os desenvolvedores.

Existem inúmeras oportunidades no pipeline de entrega para realizar varreduras de segurança, incluindo:

- Na máquina local: por meio do IDE ou suítes de testes integradas
- Integração contínua com leitura de código: leitura de código em envio de código, fusões e lançamentos no sistema de integração contínua
- Pré-implantação de entrega contínua: verificação antes da implantação

6.4 Protegendo Componentes de Software

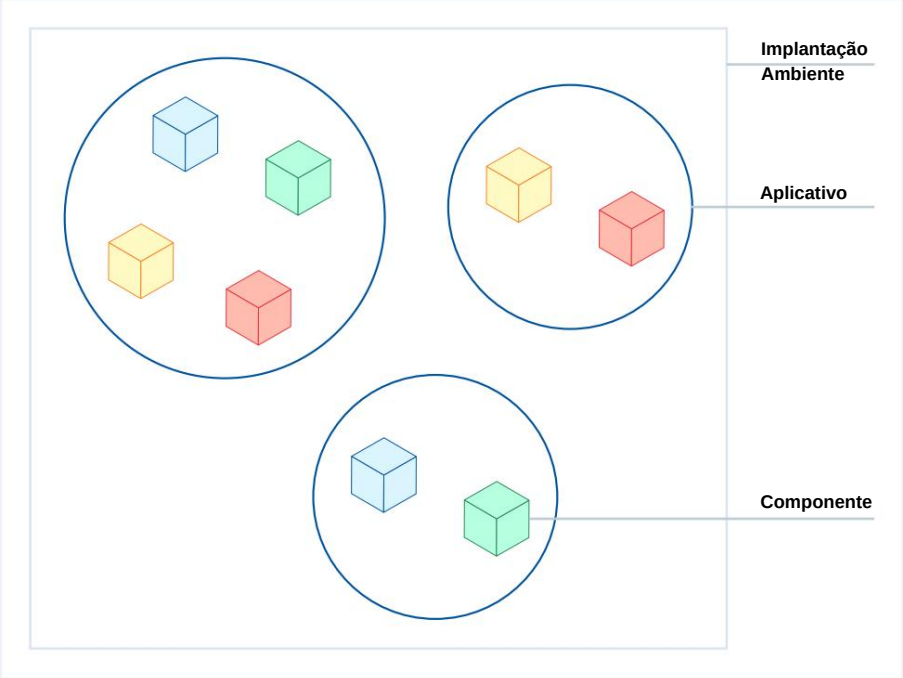


Figura 3: Relacionamento entre Aplicativo, Componentes de Aplicativo e Ambiente

Um aplicativo só pode ser tão seguro quanto seus componentes. As vulnerabilidades dos seus componentes, sejam eles produzidos internamente, por terceiros ou a partir de projetos de código aberto, provavelmente comprometerão a aplicação mais ampla.

É crucial gerir estes componentes de software de forma sistemática, com medidas como:

- detecção imediata e relatório de vulnerabilidades em software de terceiros, como software aberto fonte e contêineres
- detectar o uso de componentes não autorizados

© Copyright 2020, Cloud Security Alliance. Todos os direitos reservados.

Machine Translated by Google

O gerenciamento de vulnerabilidades desses componentes deve ser contínuo, de forma que seja feito não apenas durante o processo de desenvolvimento, mas também posteriormente. É fundamental garantir que os componentes de terceiros utilizados permaneçam seguros durante a vida útil do aplicativo.

Após a liberação de qualquer vulnerabilidade pública de componentes da aplicação, um processo deverá ser acionado indicando a necessidade de atualização para uma versão não vulnerável da componentLibrary.

Este processo pode ser automatizado usando ferramentas de varredura de código-fonte por meio do processo de integração contínua ou varreduras em tempo de execução.

6.5 Protegendo Aplicativos

Um aplicativo deve ser testado em termos de segurança em um ambiente realista antes de ser confiável. Mesmo com uma arquitetura sólida, código seguro e componentes livres de vulnerabilidades, a integração deles não leva necessariamente a uma aplicação à prova de balas.

O teste dinâmico de aplicativos pode ser usado para testar aplicativos nos ambientes de preparação, teste e produção para garantia. As ferramentas DAST podem descobrir uma variedade de problemas de segurança, desde injeção de falhas, detecção de falhas até vazamentos de recursos e segredos. (Veja a Figura 1 para o(s) posicionamento(s) apropriado(s) do DAST no pipeline).

A difusão também é um método eficaz para identificar vulnerabilidades de segurança. Vários tipos de difusão automatizada estão disponíveis para serem executados no pipeline de entrega para garantir a resiliência do aplicativo contra entradas errôneas e maliciosas.

6.6 Protegendo Ambientes

Em geral

Um ambiente hostil torna um aplicativo seguro inseguro. As abordagens de Infraestrutura como Código (IaC), aplicações em contêineres e implantação contínua apresentam novas possibilidades para proteger infraestruturas e ambientes, incluindo:

Um ambiente hostil torna um aplicativo seguro inseguro. As abordagens de Infraestrutura como Código (IaC), aplicações em contêineres e implantação contínua apresentam novas possibilidades para proteger infraestruturas e ambientes, incluindo:

1. Verificação de código e artefatos IaC
2. Defesa em tempo de execução dentro dos ambientes

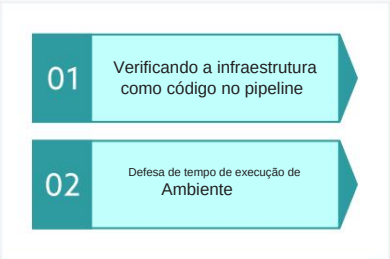


Figura 4: Abordagem Multifacetada para Protegendo a infraestrutura

© Copyright 2020, Cloud Security Alliance. Todos os direitos reservados.

Machine Translated by Google

Segurança IaC

O código IaC é usado para gerenciamento declarativo de recursos de infraestrutura em nuvem, e o estado de configuração IaC pode ser um reflexo preciso da configuração real desses recursos. A análise de segurança do código e artefatos IaC pode ser realizada com dois mecanismos:

1. Verificação estática de modelos IaC e estados de configuração
2. Verificação automatizada de recursos implantados declarativamente

Segurança de tempo de execução

Os aplicativos implantados estão sujeitos à segurança do seu ambiente e, portanto, devem ser monitorados. Geralmente existem duas abordagens para isso:

1. O monitoramento do tempo de execução de aplicativos e ambientes implantados são medidas importantes para garantir que tanto o ambiente quanto o aplicativo estejam protegidos (veja a Figura 4).
2. A defesa em tempo de execução garante que as vulnerabilidades e ataques descobertos em tempo de execução sejam mitigados e relatado o mais rápido possível.

6.7 Gerenciando Segredos

O gerenciamento de segredos dentro da aplicação e do pipeline de entrega é fundamental e impacta todo o pipeline de entrega. O uso de algoritmos criptográficos adequados não fornece necessariamente segurança suficiente contra adversários. Por exemplo, um desenvolvedor pode selecionar o melhor algoritmo de criptografia, mas se a chave for armazenada em um local aberto, a segurança ficará comprometida.

Os processos, incluindo gerenciamento de chaves e rotação de chaves, devem ser tratados adequadamente. Os cofres de chaves na nuvem e as infraestruturas de gestão de chaves podem ser considerados parte de uma solução automatizada.

7. Melhores práticas de automação

7.1 Geral

Muitas atividades de segurança usadas no pipeline de entrega habilitado para segurança podem ser usadas de forma independente em qualquer processo DevSecOps. A maioria das organizações maduras dispõe de disposições para testes fora de banda (também conhecidos como assíncronos). Esta seção destaca as melhores práticas e considerações aplicáveis além do DevSecOps.

7.2 Mitigando Vulnerabilidades

As vulnerabilidades são mais fáceis de resolver quando acabam de ser criadas. Com isto em mente, a estratégia é mover a detecção de vulnerabilidades recém-criadas para o processo de desenvolvimento, antes da implantação. Por exemplo, podem ser consideradas ferramentas que verificam problemas de segurança no IDE ou antes de confirmar as alterações no código.

© Copyright 2020, Cloud Security Alliance. Todos os direitos reservados.

Machine Translated by Google

Para códigos vulneráveis detectados em fases posteriores, é importante que sejam atenuados antes da implantação em produção. Com base no risco avaliado de vulnerabilidades, as de baixa gravidade poderão ser abordadas posteriormente; aqueles de alta gravidade devem ser tratados o mais rápido possível.

7.3 Teste Assíncrono (Teste Fora de Banda)

Geralmente existem duas categorias de testes assíncronos:

- Certas atividades cruciais de segurança não podem ser totalmente automatizadas, pois exigem inteligência. Isso inclui modelagem de ameaças, testes de penetração e revisão de código por pares.
- Testes automatizados pesados que levam muito tempo para serem executados, como o SAST.

Essas atividades podem ser acionadas e executadas “fora da banda”, em vez de alinhadas com a implantação automatizada, para não interromper o pipeline de implantação do software.

Os problemas identificados fora da banda devem ser rastreados pela equipe de entrega para garantir visibilidade, priorização e possível reversão, dependendo dos riscos identificados. Se problemas críticos decorrentes de testes fora de banda não forem resolvidos a tempo, eles deverão contar como bloqueadores do processo de entrega contínua nos pontos de verificação apropriados.

7.4 Ciclos de Feedback Contínuo

Quanto mais tempo um defeito de segurança existir no aplicativo, maior será a probabilidade de novas alterações e implantações aumentarem a complexidade e o custo da correção do defeito.

As equipes de DevSecOps devem ser alertadas sobre problemas de segurança o mais cedo possível para que os defeitos sejam evitados.

Além disso, o feedback oportuno permite que os indivíduos diagnostiquem e corrijam rapidamente as alterações que causaram a vulnerabilidade, permitindo-lhes realizar análises de causa raiz enquanto o contexto que as acompanha ainda está fresco em mente.

7.5 Quebrando Construções

A organização precisa reconhecer que a segurança é um requisito de negócios para que as atividades de segurança declarem “construções quebradas” em caso de falha no cumprimento dos requisitos de segurança, ao mesmo tempo que interrompe um pipeline de implantação automatizado.

Quebrar compilações para fins de segurança reforça a qualidade, fornece feedback rápido às equipes de DevSecOps e evita a exposição de dados e possíveis explorações fora do apetite de risco da organização, por exemplo, liberando código com alta severidade, vulnerabilidade publicamente conhecida com explorações ativas.

© Copyright 2020, Cloud Security Alliance. Todos os direitos reservados.

Machine Translated by Google

As atividades que podem quebrar uma compilação incluem:

> • Defeito de alta gravidade identificado além do apetite de risco externo da organização • Vulnerabilidade específica ou classe de vulnerabilidade identificada que entra em conflito com a política da organização
> • O número de descobertas altas ou críticas ultrapassa um determinado limite
> • As atividades de segurança necessárias não são executadas de forma eficaz, por exemplo, as verificações automatizadas estão falhando por causa de erros de configuração

8. Conclusão

Para se beneficiar do DevSecOps e da abordagem de segurança reflexiva, é preciso confiar fortemente na automação.

O pilar Automação pode ser alcançado usando o pipeline de entrega habilitado para segurança e pela aplicação de controles de segurança dentro dele. A abordagem priorizada pelo risco permite ainda a otimização para entrega contínua e um mecanismo para lidar com verificações de segurança não automatizadas.

© Copyright 2020, Cloud Security Alliance. Todos os direitos reservados.

Machine Translated by Google

Referências

1. 3 riscos de segurança que a análise da arquitetura pode resolver. Sinopse, 2016. Disponível em: https://www.synopsys.com/blogs/software-security/security-risks-that-architecture-análise-can-resolve/

2. Modelagem Tática de Ameaças. SAFECode, 2017. Disponível em: https://safecode.org/wp-content/uploads/2017/05/SAFECode_TM_Whitepaper.pdf

3. Gerenciamento de riscos de segurança inerentes ao uso de componentes de terceiros. Código SEGURO, 2017. Disponível em: https://safecode.org/wp-content/uploads/2017/05/SAFECode_TPC_Whitepaper.pdf

4. Concentre-se na difusão. SAFECode, 2020. Disponível em: https://safecode.org/fuzzing/

© Copyright 2020, Cloud Security Alliance. Todos os direitos reservados.