Google Cloud

White paper
Janeiro de 2022

Entrega de software com segurança

Índice

Capítulo

01 Entendendo a cadeia de abastecimento de software

Capítulo

02 Frameworks e padrões para todo o setor

Índice

Capítulo

03 Serviços gerenciados para cada estágio

Capítulo

04 Primeiros passos

Capítulo

05 Leitura adicional

Capítulo

01 Entendendo a cadeia de abastecimento de software

Cenário de segurança atual

Velocidade e tempo de lançamento no mercado são as prioridades das empresas que desenvolvem software e aplicações para atender às necessidades dos clientes. Esses imperativos estratégicos têm sido a força por trás do enorme crescimento dos contêineres como plataforma selecionada. No ano passado, como muitas dessas empresas aproveitaram os benefícios dos contêineres, como maior rapidez no lançamento no mercado, maior disponibilidade, mais segurança, melhor escalabilidade e custos reduzidos, elas passaram a considerar também a abordagem sem servidor.

As soluções de software reduziram o tempo de entrega de um novo recurso ou até mesmo de um novo produto, mas muitas das práticas de segurança atuais não conseguem acompanhar o aumento na velocidade, levando a um destes três problemas:

Os desenvolvedores são prejudicados pelos processos existentes, resultando em atrasos

As equipes de segurança e operações fazem concessões que deixam a empresa exposta a ameaças

As equipes de desenvolvimento contornam os processos de segurança para cumprir prazos, ficando vulneráveis

Nos últimos anos, houve uma série de violações de segurança classificadas como ataques à “cadeia de abastecimento de software”.

O **Log4Shell** foi uma vulnerabilidade perigosa no software Apache Log4j identificada em dezembro de 2021. Sinalizada com a pontuação máxima de 10 no CVSS, essa vulnerabilidade foi particularmente devastadora devido à popularidade do Log4j, um framework de registro em log baseado em Java. Dois fatores contribuíram para a gravidade: primeiro, era muito fácil explorar e permitir a execução remota completa do código e, segundo, ele ficava, com frequência, debaixo de muitas camadas na árvore de dependência e, portanto, passava despercebido.

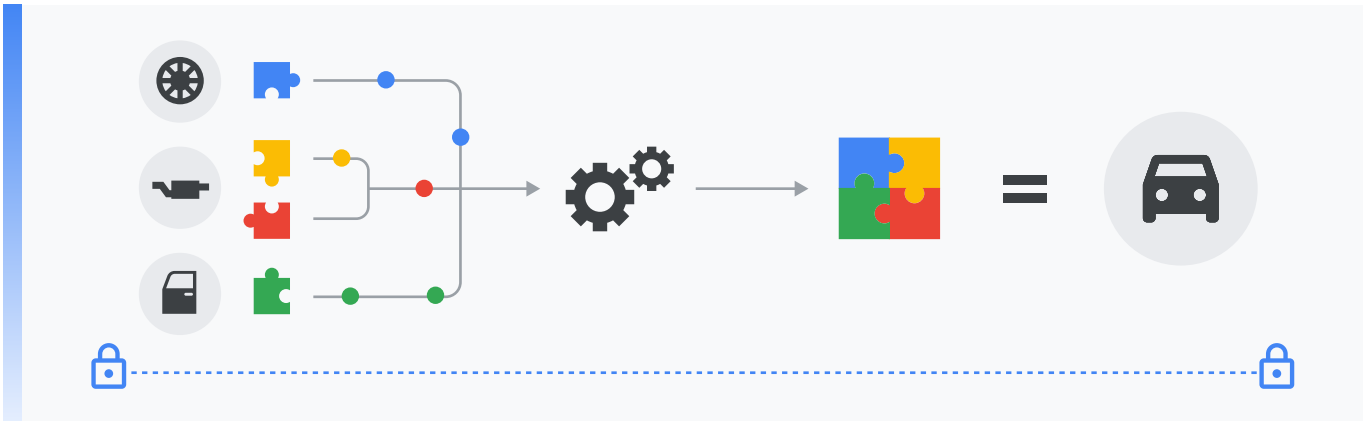
A **Solarwinds,** uma empresa de software de gerenciamento de TI, foi atacada por invasores nacionais que injetaram um código malicioso nos builds oficiais de software de código aberto em uso na empresa. Essa atualização mal-intencionada foi enviada para 18 mil clientes, inclusive os Departamentos de Comércio e do Tesouro dos EUA.

A **Kaseya**, outro provedor de software de gerenciamento de TI, foi atacada por uma vulnerabilidade de dia zero que comprometeu o servidor VSA da Kaseya e enviou um script malicioso com ransomware que criptografava todos os arquivos nos sistemas afetados.

A necessidade urgente de responder a esses e outros incidentes semelhantes levou a Casa Branca, nos EUA, a sancionar um decreto, em maio de 2021, exigindo que as empresas que fazem negócios com o governo federal mantivessem determinados padrões de segurança de software.

Cadeia de abastecimento de software

Em vários aspectos, o termo “cadeia de abastecimento de software” é muito apropriado: os processos usados na criação de uma cadeia desse tipo são bastante semelhantes aos da fabricação de carros.

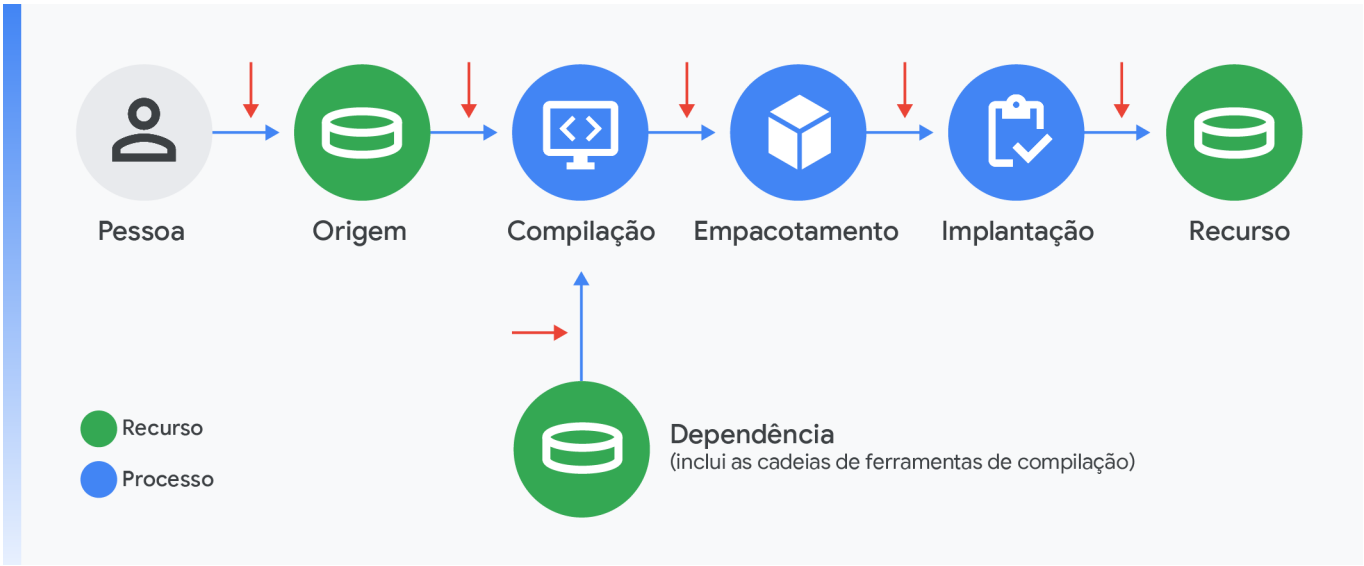


Um fabricante de automóveis compra diversas peças prontas, fabrica componentes próprios e, depois, une todos eles usando um processo altamente automatizado. O fabricante garante a segurança das operações certificando-se que cada componente de terceiros venha de uma fonte confiável. Os componentes próprios são amplamente testados para assegurar que não têm problemas de segurança. Por fim, é feita a montagem seguindo um processo confiável que resulta em carros prontos.

A cadeia de abastecimento de software é parecida em muitos aspectos. Um fabricante de software recebe componentes de terceiros, que realizam funções específicas e geralmente são de código aberto, e desenvolve um software próprio, que é a propriedade intelectual principal. Depois, o código passa por um processo de compilação que combina esses componentes em artefatos, posteriormente implantados em produção.

O elo frágil na cadeia

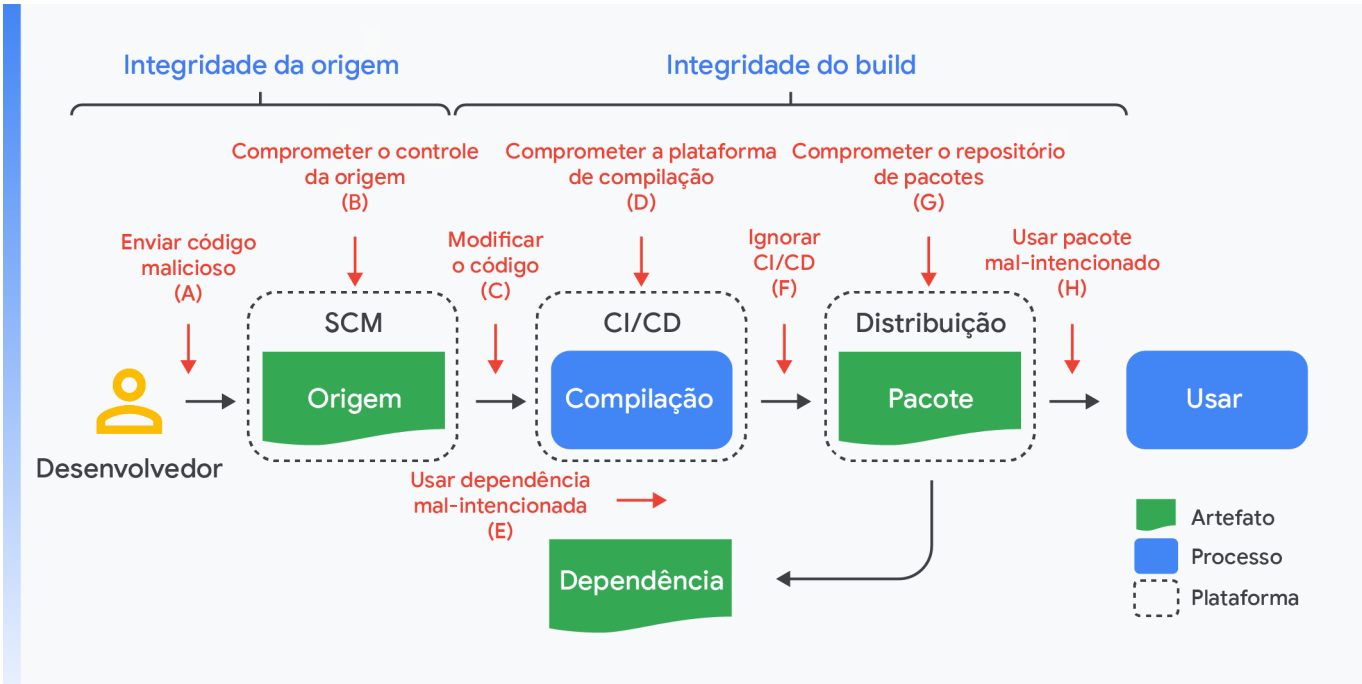
Basta um elo não seguro para violar a cadeia de abastecimento de software



Assim como nos ataques de alto nível do ano passado, cada etapa do processo pode levar a uma fraqueza que os hackers podem explorar.
Por exemplo, um pacote npm tem, em média, 12 dependências diretas e aproximadamente 300 dependências indiretas. Além disso, sabemos que quase 40% de todos os pacotes npm publicados dependem de código com vulnerabilidades conhecidas..

Talvez essas vulnerabilidades não tornem o código inseguro. Por exemplo, a vulnerabilidade pode estar em uma parte da biblioteca que nunca é usada. No entanto, elas precisam ser verificadas.

A escala do problema é gigantesca. Se apenas uma dessas vulnerabilidades ficar sem correção, os agentes de ameaça podem ter uma oportunidade para acessar sua cadeia de abastecimento de software.



Veja alguns exemplos de ataques que aproveitam as vulnerabilidades em cada um dos estágios mostrados no diagrama acima.

	Ameaça	Exemplo conhecido
A	Enviar código malicioso para o repositório de origem	Hypocrite Commits do Linux: um pesquisador tentou inserir vulnerabilidades no kernel do Linux de maneira intencional por meio de patches na lista de e-mails.
B	Comprometer a plataforma de controle da origem	PHP: o hacker comprometeu o servidor Git auto-hospedado do PHP e injetou dois commits maliciosos..
C	Compilar usando o processo oficial, mas a partir de um código que não corresponde ao controle de origem	Webmin: o hacker modificou a infraestrutura do build para usar arquivos de origem que não correspondiam ao controle de origem.
D	Comprometer a plataforma de compilação	SolarWinds: o hacker comprometeu a plataforma de compilação e instalou uma implantação para injetar um comportamento malicioso durante cada compilação.
E	Usar uma dependência mal-intencionada (isto é, A-H, recursivamente)	event-stream: o hacker inseriu uma dependência inócua e, depois, atualizou a dependência para adicionar um comportamento malicioso. A atualização não correspondia ao código enviado ao GitHub (isto é, ataque F).
F	Fazer upload de um artefato que não tenha sido compilado pelo sistema de CI/CD	CodeCov: o hacker usou credenciais provenientes de um vazamento de dados para fazer upload de um artefato malicioso para um bucket do Google Cloud Storage, de onde os usuários fazem o download diretamente.
G	Comprometer o repositório de pacotes	Ataques a espelhamentos de pacote: um pesquisador executou imagens espelhadas de vários repositórios de pacotes conhecidos, que poderiam ter sido usados para enviar pacotes maliciosos.
H	Induzir o consumidor a usar um pacote mal-intencionado	Typosquatting do Browserify: um hacker fez upload de um pacote malicioso com um nome semelhante ao do original.

Fortalecendo a cadeia: liderança técnica do Google Cloud em código aberto

No Google, desenvolvemos aplicações em escala global há décadas. Ao longo do tempo, abrimos o código de muitos dos nossos projetos internos para ajudar a acelerar o trabalho dos desenvolvedores. Ao mesmo tempo, definimos vários processos internos para proteger a experiência de software.

Veja aqui alguns dos esforços em que estamos envolvidos para tornar as cadeias de abastecimento de software mais seguras em qualquer lugar.

Maior investimento – anunciamos em agosto de 2020 que investiremos US$ 10 bilhões nos próximos cinco anos para fortalecer a segurança cibernética, que inclui expandir os programas de confiança zero, ajudar a proteger a cadeia de abastecimento de software e melhorar a segurança do código aberto.

Supply-chain Levels for Software Artifacts (SLSA) – SLSA é um framework de ponta a ponta para manter a integridade da cadeia de abastecimento. É um equivalente em código aberto de muitos dos processos que implementamos internamente no Google ao longo do tempo. O SLSA fornece uma proveniência auditável do que foi compilado e como.

DevOps Research and Assessment (DORA) – nossa equipe DORA conduziu um programa de pesquisa de sete anos, validando diversos recursos técnicos, culturais, de processos e medição que promovem um melhor desempenho da entrega de software e organizacional.

Open Source Security Foundation – em 2019, fomos cofundadores do Open Source Security Foundation, um fórum entre setores sobre a segurança das cadeias de abastecimento.

Allstar – é um app do GitHub instalado em empresas ou repositórios para definir e impor políticas de segurança. Permite a aplicação contínua das práticas recomendadas de segurança nos projetos do GitHub.

Open Source Scorecards – o Scorecards usa métricas de avaliação, como políticas de segurança bem definidas, processo de revisão do código e cobertura de testes contínuos, com ferramentas de teste de fuzzing e análise estática do código para estabelecer a pontuação de risco dos projetos de código aberto.

Acreditamos que dois elementos são necessários para resolver o problema da segurança da cadeia de abastecimento de software:

1. Frameworks e padrões para todo o setor.

2. Serviços gerenciados que implementem esses padrões usando princípios de privilégio mínimo em uma arquitetura de confiança zero. A arquitetura de confiança zero é aquela em que nenhuma pessoa, dispositivo ou rede tem confiança inerente, ou seja, toda confiança, que permite acesso às informações, precisa ser conquistada.

Vamos analisar cada um deles:

Capítulo

02 Frameworks e padrões para todo o setor

Para entender os princípios envolvidos na proteção da cadeia de abastecimento de software, vamos começar pelo SLSA.

No estado atual, o SLSA é um conjunto de diretrizes de segurança adotadas de maneira incremental que estão sendo definidas em consenso no setor. Em sua forma final, o que diferencia o SLSA de uma lista de práticas recomendadas é a imposição: ele será compatível com a criação automática de metadados auditáveis que podem ser inseridos em mecanismos de políticas para conceder a “certificação de SLSA” a um pacote específico ou uma plataforma de compilação.

O SLSA foi desenvolvido para ser incremental e relevante, além de oferecer benefícios de segurança em todas as etapas. Quando um artefato é qualificado com o nível mais alto, os consumidores podem confiar que ele não foi adulterado e pode ser rastreado com segurança até a origem. Isso é difícil, senão impossível, na maioria dos softwares hoje em dia.

O SLSA tem quatro níveis, sendo que o SLSA 4 é o estado final ideal. Os níveis mais baixos representam marcos incrementais com garantias de integridade incrementais correspondentes. Atualmente, os requisitos são definidos da seguinte maneira:

SLSA 1 exige que o processo de compilação seja totalmente automatizado/tenha script completo e gere procedência. A procedência consiste em metadados sobre como um artefato foi criado, incluindo o processo de compilação, a origem de alto nível e as dependências. Conhecer a procedência permite que os consumidores de software tomem decisões de segurança baseadas no risco. Embora a procedência em SLSA 1 não proteja contra adulterações, ela oferece um nível básico de identificação da origem do código e pode ajudar no gerenciamento de vulnerabilidades.

SLSA 2 exige o uso de controle de versão e um serviço de compilação hospedado que gere uma procedência autenticada. Esses requisitos adicionais permitem que o consumidor tenha maior confiança quanto à origem do software. Neste nível, a procedência impede a adulteração, já que o serviço de compilação é confiável. SLSA 2 também oferece um caminho fácil para fazer upgrade para SLSA 3.

SLSA 3, além dos requisitos anteriores, exige que as plataformas de origem e compilação atendam a padrões específicos para garantir a capacidade de auditar a origem e a integridade da procedência, respectivamente. SLSA 3 oferece proteções muito melhores contra adulteração do que os níveis anteriores, evitando classes específicas de ameaças, como a contaminação entre compilações.

SLSA 4 é o nível mais alto, que exige a revisão por duas pessoas de todas as alterações e um processo de compilação hermético e reproduzível. A revisão por duas pessoas é uma prática recomendada do setor para detectar erros e impedir comportamentos mal-intencionados. Os builds herméticos garantem que a lista de dependências da procedência esteja completa. Os builds reproduzíveis, embora não sejam estritamente obrigatórios, oferecem muitos benefícios em relação à auditoria e confiabilidade. Em geral, o SLSA 4 proporciona ao consumidor um alto grau de confiança de que o software não foi adulterado.

Mais detalhes sobre os níveis propostos estão disponíveis no repositório GitHub, incluindo os requisitos correspondentes de origem e compilação/procedência.

A cadeia de abastecimento de software pode ser dividida em cinco fases distintas: programação, compilação, empacotamento, implantação e execução. Vamos abordar cada uma delas de acordo com nossa abordagem de segurança.

Capítulo

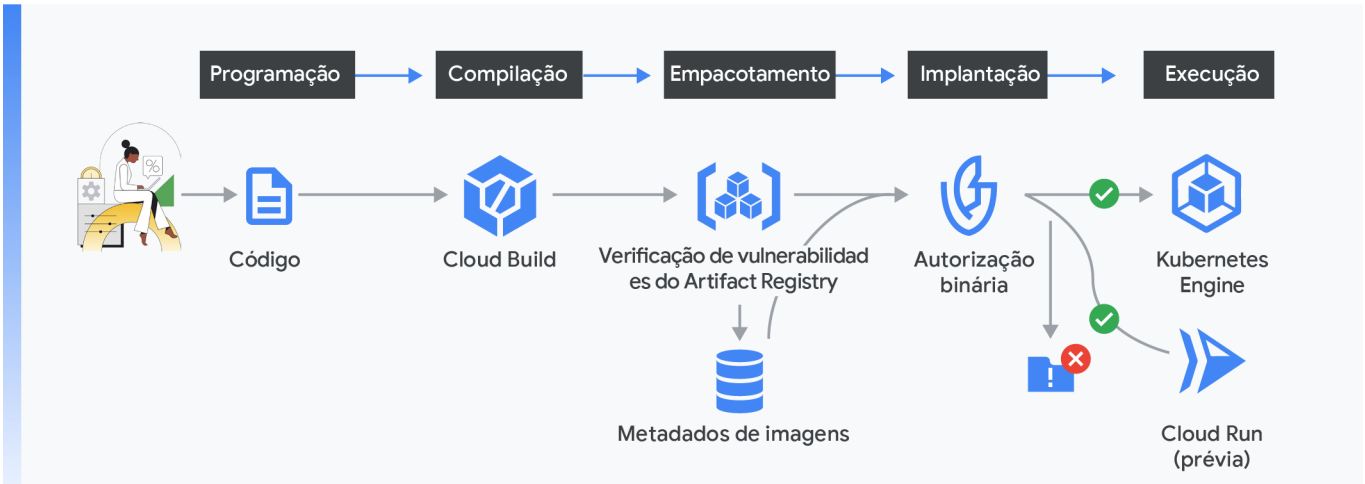
03 Serviços gerenciados para cada estágio

No Google Cloud, fornecemos ferramentas totalmente gerenciadas, desde a programação e compilação até a implantação e execução, com os padrões e as práticas recomendadas acima implementados por padrão.

Proteger sua cadeia de abastecimento de software exige a definição, verificação e manutenção de uma cadeia de confiança que estabeleça a procedência do código e garanta que o conteúdo certo seja executado em produção. No Google, conseguimos isso por meio de atestados gerados e verificados ao longo do processo de desenvolvimento e implantação de software. Dessa forma, alcançamos um determinado nível de segurança do ambiente por meio da revisão do código, procedência verificada do código e imposição de políticas. Juntos, esses processos ajudam a reduzir os riscos da cadeia de abastecimento de software, além de melhorar a produtividade dos desenvolvedores.

Na base, temos serviços de infraestrutura seguros comuns, como gerenciamento de identidade e acesso e registros em log de auditoria. Depois, protegemos a cadeia de abastecimento de software com uma maneira de definir, verificar e impor atestados em todo o ciclo de vida do software.

Vamos analisar os detalhes de como alcançar a segurança do ambiente no seu processo de desenvolvimento usando as políticas e a procedência no Google Cloud.



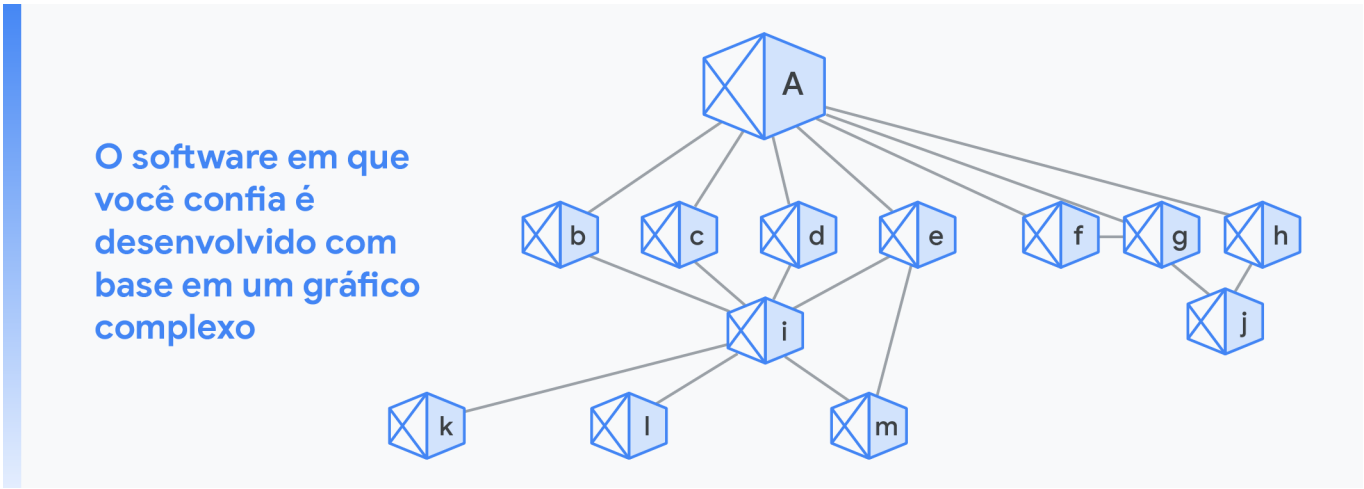
Fase 1: Programação

A proteção da cadeia de abastecimento de software começa quando os desenvolvedores passam a projetar a aplicação e escrever código. Isso abrange tanto softwares próprios como os componentes de código aberto, cada um com seus desafios.

Software de código aberto e dependências

O código aberto permite que os desenvolvedores acelerem a criação das aplicações para que as empresas sejam mais ágeis e produtivas. No entanto, softwares de código aberto não são perfeitos e, embora o setor dependa deles, muitas vezes temos poucas informações sobre suas dependências e os diferentes níveis de risco associados. Para a maioria das empresas, o risco ocorre principalmente devido a vulnerabilidades ou licenças.

O software de código aberto, os pacotes, as imagens base e outros artefatos de que você depende estabelecem a base da "cadeia de confiança".



Por exemplo, imagine que sua empresa está criando o software "a". O diagrama mostra a cadeia de confiança, isto é, o número de dependências implícitas no seu projeto. No diagrama, de "b" até "h" são dependências diretas e de "i" até "m" são dependências indiretas.

Agora, considere que há uma vulnerabilidade na parte inferior da árvore de dependências. O problema pode se espalhar por vários componentes rapidamente. Além disso, as dependências mudam com muita frequência: em um dia típico, 40 mil pacotes npm apresentam uma mudança nas dependências.

O Open Source Insights é uma ferramenta criada pelo Google Cloud que fornece um gráfico de dependência transitiva, para que você veja suas dependências e as dependências decorrentes na representação em árvore. O Open Source Insights é atualizado continuamente com orientações de segurança, informações de licenças e outros dados de segurança em várias linguagens, tudo em um só lugar. Quando usado em conjunto com o Open Source Scorecards, que fornece a pontuação de risco de projetos de código aberto, o Open Source Insights ajuda os desenvolvedores a fazer escolhas melhores entre os milhões de pacotes de código aberto disponíveis.

Para resolver essa questão, é fundamental se concentrar nas dependências como código. À medida que as dependências avançam rumo ao fim da cadeia de abastecimento, é mais difícil inspecioná-las. Para proteger as dependências, recomendamos começar pela origem:

- Use ferramentas como o Open Source Insights e OSS Scorecards para entender melhor as dependências.
- Verifique e revise todos os códigos, pacotes e imagens base usando um processo automatizado que seja uma parte essencial do seu fluxo de trabalho.
- Controle como as pessoas acessam essas dependências. É fundamental ter um controle rígido dos repositórios de código próprio e de código aberto, com restrições em relação a requisitos completos de revisão do código e auditoria.

Vamos explicar mais detalhes sobre os processos de compilação e implantação mais adiante neste artigo, mas também é importante verificar a procedência do build, usar um ambiente de compilação seguro e garantir que as imagens sejam assinadas e validadas posteriormente, no momento da implantação.

Há também uma série de práticas de programação seguras que os desenvolvedores podem aplicar:

- Automatizar testes
- Usar linguagens de software com segurança da memória
- Impor revisões de código
- Garantir a autenticidade dos commits
- Identificar código malicioso nos estágios iniciais
- Evitar a exposição de informações confidenciais
- Exigir registros em log e o resultado da compilação
- Usar gerenciamento de licenças

Fase 2: Compilação

A próxima etapa para proteger a cadeia de abastecimento de software envolve a criação de um ambiente de compilação seguro em larga escala. Basicamente, o processo de compilação começa com a importação do código-fonte, possivelmente em uma das muitas linguagens de um repositório, e a execução de builds para atender às especificações descritas nos arquivos de configuração.

Provedores de nuvem como o Google oferecem acesso a um ambiente de compilação gerenciado e atualizado, que permite criar imagens dos builds em qualquer escala necessária.

Durante o processo de compilação, há várias questões a serem consideradas:

- Os secrets ficam protegidos durante e depois do processo de compilação?
- Quem tem acesso aos ambientes de compilação?
- E quanto aos vetores de ataque ou riscos de exfiltração relativamente novos?

Para desenvolver um ambiente de compilação seguro, comece com os secrets. Eles são essenciais e relativamente fáceis de proteger. Inicialmente, garanta que os secrets nunca sejam texto não criptografado e, se possível, não façam parte do build. Você precisa garantir que eles estejam criptografados e que seus builds sejam parametrizados para se referir a secrets externos para serem usados conforme necessário. Isso também simplifica a rotação periódica de secrets e minimiza o impacto de qualquer vazamento de dados.

A etapa seguinte consiste em configurar as permissões do build. Há vários usuários e contas de serviço envolvidas no processo de compilação. Por exemplo, alguns usuários talvez precisem gerenciar secrets, enquanto outros talvez precisem gerenciar o processo de compilação adicionando ou modificando etapas. Outros ainda podem precisar visualizar registros em log.

Ao fazer isso, é importante seguir estas práticas recomendadas:

- A mais relevante delas é o princípio de privilégio mínimo. Implemente permissões de alta granularidade para conceder aos usuários e contas de serviço as permissões exatas de que precisam para realizar as tarefas com eficiência.
- É necessário saber como os usuários e as contas de serviço interagem e ter uma compreensão clara da cadeia de responsabilidade, como a configuração de um build, sua execução e os efeitos subjacentes do build.

Em seguida, ao escalar esse processo, estabeleça restrições para o build, na medida do possível, e use a automação para aumentar a escala por meio da configuração como código e da parametrização. Dessa forma, você pode auditar todas as alterações do processo de compilação de maneira eficaz. Além disso, verifique se você atende às necessidades de conformidade legal por meio do controle de aprovação de builds e implantações confidenciais, solicitações de pull para alterações na infraestrutura e revisões regulares dos logs de auditoria realizadas por pessoas.

Por fim, verifique se a **rede** atende ao que você precisa. Na maioria dos casos, é melhor hospedar seu próprio código-fonte em redes privadas protegidas por firewall. O Google Cloud oferece acesso a recursos como pools particulares do Cloud Build, um ambiente de compilação seguro e sem servidor no seu perímetro de rede privada, e recursos como os controles de serviço da VPC para impedir a exfiltração da propriedade intelectual.

Autorização binária

Embora o IAM seja essencial e, ao mesmo tempo, um ponto de partida lógico, ele não é infalível. O vazamento de credenciais representa um sério risco à segurança. Para reduzir sua dependência do IAM, é possível mudar para um sistema baseado em atestados menos propenso a erros. O Google usa um sistema chamado de autorização binária, que só permite a implantação de cargas de trabalho confiáveis.

O serviço de autorização binária estabelece, verifica e mantém uma cadeia de confiança por meio de atestados e verificações de política ao longo de todo o processo. Basicamente, a autorização binária gera assinaturas criptográficas, isto é, atestados, à medida que o código e outros artefatos migram para a produção. Posteriormente, antes da implantação, os atestados são verificados com base nas políticas.

Ao usar o Google Cloud Build, um conjunto de atestados é capturado e adicionado à cadeia de confiança geral. Por exemplo, são gerados atestados para quais tarefas foram executadas, quais ferramentas e processos de compilação foram usados e assim por diante. O Cloud Build ajuda a atingir o SLSA de nível 1, capturando a origem da configuração do build, que pode ser usada para confirmar que o script do build foi gerado. Builds com script são mais seguros que os manuais e são obrigatórios para o SLSA nível 1. Além disso, é possível pesquisar a procedência e outros atestados do build usando o resumo de imagens do contêiner, que cria uma assinatura única para qualquer imagem e também é exigido para o SLSA nível 1.

Fase 3: Empacotamento

Quando a compilação for concluída, você terá uma imagem de contêiner quase pronta para produção. É essencial ter um local seguro para armazenar as imagens, que impeça a adulteração das imagens existentes e o upload de imagens não autorizadas. Seu gerenciador de pacotes provavelmente precisará ter as imagens dos seus builds e dos builds de código aberto, bem como os pacotes de linguagens usados pelas aplicações.

O Artifact Registry do Google Cloud oferece esse repositório. O Artifact Registry é um local centralizado para sua empresa gerenciar imagens de contêiner, bem como pacotes de linguagem, como Maven e npm. Ele é totalmente integrado às ferramentas e aos tempos de execução do Google Cloud e é compatível com protocolos de artefatos nativos. Isso facilita a integração com suas ferramentas de CI/CD durante a configuração de pipelines automatizados.

Assim como na etapa de compilação, é necessário garantir que as permissões de acesso ao Artifact Registry sejam bem pensadas e sigam os princípios de privilégio mínimo. Além de restringir o acesso não autorizado, o repositório de pacotes pode agregar muito mais valor. O Artifact Registry, por exemplo, inclui a verificação de vulnerabilidades para conferir as imagens e garantir que elas sejam seguras para a implantação. Esse serviço verifica as imagens em um banco de dados de vulnerabilidades atualizado constantemente para avaliar novas ameaças e pode enviar alertas quando uma vulnerabilidade for encontrada.

Esta etapa gera mais metadados, incluindo um atestado que confirma se os resultados da vulnerabilidade de um artefato estão dentro de determinados limites de segurança. Depois, as informações são armazenadas no nosso serviço de análise, que estrutura e organiza os metadados do artefato, e ficam acessíveis para a autorização binária. É possível usar esse recurso para impedir automaticamente a implantação de imagens perigosas no Google Kubernetes Engine (GKE).

Fases 4 e 5: Implantação e execução

As duas últimas fases da cadeia de abastecimento de segurança de software são a implantação e a execução. Embora sejam duas etapas separadas, faz sentido pensar nelas juntas como uma forma de garantir que somente builds autorizados cheguem à produção.

No Google, desenvolvemos práticas recomendadas para determinar que tipos de build precisam ser autorizados. A primeira prática é garantir a integridade da cadeia de abastecimento para que ela produza apenas artefatos confiáveis. Depois, é preciso incluir o gerenciamento de vulnerabilidades no ciclo de vida de entrega de software. Por fim, combinamos essas duas práticas para impor fluxos de trabalho de verificação de vulnerabilidades e integridade baseados em políticas.

Ao chegar a este estágio, você já passou pelas fases de programação, compilação e empacotamento. A autenticidade dos atestados capturados ao longo da cadeia de abastecimento pode ser verificada pela autorização binária. No modo de imposição, as imagens são implantadas apenas quando os atestados atendem às políticas da sua empresa e, no modo de auditoria, as violações de política são registradas em log e acionam alertas. Também é possível usar a autorização binária para restringir a execução de builds a menos que eles tenham sido criados usando o processo aprovado do Cloud Build. A autorização binária garante que apenas códigos revisados e autorizados sejam implantados.

É essencial implantar suas imagens em um ambiente de tempo de execução confiável. Nossa plataforma gerenciada do Kubernetes, o GKE, usa uma abordagem de priorização da segurança em contêineres.

O GKE resolve muitas das preocupações importantes em relação à segurança dos clusters. Os upgrades automáticos de clusters permitem manter o Kubernetes com patches e atualizado automaticamente usando canais de lançamento.

Inicialização segura, nós protegidos e verificação de integridade garantem que o kernel e os componentes

do cluster do nó não tenham sido modificados, estejam executando o que você pretende e que os nós maliciosos não possam entrar no cluster. Por fim, a computação confidencial permite executar clusters com nós em que a memória é criptografada. Assim, os dados permanecem confidenciais mesmo durante o processamento. Combine isso com a criptografia de dados em repouso e em trânsito na rede, e o GKE fornece um ambiente muito seguro, particular e confidencial para executar suas cargas de trabalho conteinerizadas.

Além disso, o GKE melhora a segurança das aplicações por meio do gerenciamento de certificados dos balanceadores de carga, identidade de cargas de trabalho e recursos de rede avançados com uma maneira eficiente de configurar e proteger a entrada no cluster. O GKE ainda oferece ambientes de sandbox para executar aplicações não confiáveis, protegendo o restante das cargas de trabalho.
Com o GKE Autopilot, as práticas recomendadas e os recursos de segurança do GKE são implementados automaticamente, o que reduz ainda mais a superfície de ataque e minimiza a configuração incorreta que pode levar a problemas de segurança.

Logicamente, a necessidade de verificação vai além da implantação. A autorização binária também oferece validação contínua, permitindo a conformidade contínua com as políticas definidas, mesmo após a implantação. Se uma aplicação em execução não estiver em conformidade com uma política existente ou recém-adicionada, um alerta será criado e registrado em log. Dessa forma, você sabe que o que está sendo executado em produção é exatamente o que você pretendia.

Gerenciamento de vulnerabilidades

Além de assegurar a integridade, outro aspecto da segurança da cadeia de abastecimento consiste em garantir que as vulnerabilidades sejam encontradas rapidamente e corrigidas. Os hackers evoluíram para inserir vulnerabilidades ativamente em projetos upstream. O gerenciamento de vulnerabilidades e a detecção de defeitos precisam ser incorporados em todos os estágios do ciclo de vida da entrega de software.

Quando o código estiver pronto para implantação, use um pipeline de CI/CD e empregue as diversas ferramentas disponíveis para fazer uma verificação abrangente do código-fonte e dos artefatos gerados. As ferramentas incluem analisadores estáticos, teste de fuzzing e vários tipos de verificador de vulnerabilidades.
Depois de implantar a carga de trabalho em produção e enquanto ela é executada para atender aos usuários, é necessário monitorar ameaças emergentes e ter planos para tomar medidas imediatas.

Conclusão

Em resumo, a segurança da cadeia de abastecimento de software envolve práticas recomendadas, como SLSA, e o uso de serviços gerenciados confiáveis que ajudam a implementá-las.

Isso é essencial para:

- Começar a criar o código e as dependências e garantir que é possível confiar nelas.
- Proteger seu sistema de compilação e usar atestados para verificar se todas as etapas de compilação necessárias foram seguidas.
- Garantir que todos os pacotes e artefatos são confiáveis e que não podem ser adulterados.
- Controlar quem pode implantar o quê e manter uma trilha de auditoria. Use a autorização binária para validar os atestados de cada artefato que você quer implantar.
- Executar suas aplicações em um ambiente confiável e garantir que ninguém faça adulterações enquanto estiverem em execução. Monitore todas as vulnerabilidades recém-descobertas para proteger a implantação.

No Google, incorporamos práticas recomendadas para cada etapa dessa jornada no nosso portfólio de produtos. Assim, você tem uma base confiável para desenvolver suas aplicações.

Capítulo

04 Primeiros passos

Preparado para começar a proteger sua cadeia de abastecimento de software? Precisamos deixar claro que nenhuma ação protegerá toda a cadeia de abastecimento, não importa por onde você começar. Além disso, nenhuma ação é mais importante do que outra quando o assunto é a segurança. Dito isso, veja aqui quatro recomendações para começar a proteger sua cadeia de abastecimento.

1. Aplique patches de software

Se você implantar código no ambiente de produção com vulnerabilidades conhecidas, facilitará o trabalho dos hackers. A partir desse ponto, não importa o nível de proteção da sua cadeia de abastecimento de software, porque o invasor já obteve acesso. Portanto, aplicar patches é fundamental.

2. Assuma o controle do que é executado no seu ambiente

Depois de dominar a aplicação de patches, você poderá controlar a própria cadeia de abastecimento de software. Inicialmente, você precisa garantir que o que está em execução realmente veio das suas ferramentas de compilação ou repositórios confiáveis. Esse nível de controle ajuda a evitar ataques propositais e erros involuntários, como no caso de um desenvolvedor que implantou algo que não sabia que era inseguro. Assim, você tem uma base sólida para adicionar ferramentas como testes de cliques e autorização binária.

3. Garanta que os pacotes de fornecedores terceiros são seguros

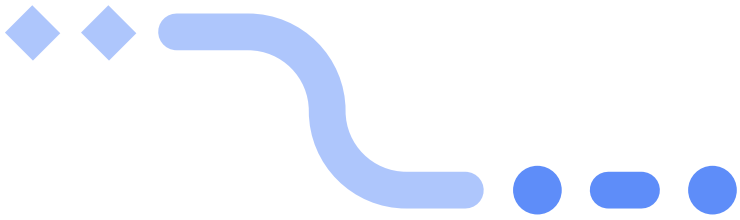
Um problema emergente em relação à segurança da cadeia de abastecimento é a frequência com que o software dos fornecedores é comprometido, abrindo um canal para ransomware ou acesso não autorizado nas implantações de clientes-alvo. Os pacotes de fornecedores terceiros que são executados no ambiente (por exemplo, produtos de gerenciamento de sistemas, produtos de gerenciamento de rede ou produtos de segurança) geralmente têm altos níveis de privilégio. Recomendamos solicitar a esses fornecedores que avancem além das instruções-padrão de segurança para fornecer um determinado grau de garantia sobre os pacotes que você está usando. Pergunte ao fornecedor qual é o nível de conformidade com SLSA ou se ele está no escopo dos requisitos do decreto recente dos EUA.

4. Mantenha uma cópia particular do código-fonte

Se você estiver usando um software de código aberto, não use uma versão extraída diretamente da internet para a criação. Em vez disso, mantenha uma cópia privada com correções para ter um ponto inicial íntegro de cada build e conseguir saber, com total confiança, a origem do código-fonte.

Chapter

05 Leitura adicional



Práticas recomendadas de DevOps

1. Seis anos do relatório State of DevOps, um conjunto de artigos com informações detalhadas sobre os recursos que preveem o desempenho da entrega de software, além de uma verificação rápida para ajudar você a descobrir como está se saindo e como melhorar.
2. Relatório Accelerate State of DevOps de 2021 do Google Cloud
3. Whitepaper do Google Cloud: Re-architecting to cloud native: an evolutionary approach to increasing developer productivity at scale

Proteção da cadeia de abastecimento de software

1. Blog do Google Cloud: What is zero trust identity security?
2. Blog sobre segurança do Google: Introducing SLSA, an end-to-end framework for supply chain integrity
3. Whitepaper do Google Cloud: Shifting left on security: Securing software supply chains

Preparado para as próximas etapas?

Para saber mais sobre como o Google Cloud pode ajudar a proteger sua cadeia de abastecimento de software e sua empresa, entre em contato.

Fale com um especialista