API Security

Guia para iniciantes

gocache

Sumário

Você conhece a lista das Top 10 ameaças para aplicações web que a OWASP divulga periodicamente?

Se respondeu que sim, você sabia que o mesmo órgão iniciou em 2019 um projeto similar, mas focado em APIs? Se você não conhece a **OWASP API Top 10**, não se sinta sozinho. Poucas pessoas têm conhecimento da existência deste projeto, principalmente no Brasil. Isso revela duas coisas:

1. **API Security é um assunto emergente e que merece atenção dedicada.**

2. **O assunto ainda é mal difundido.**

Mas, por que API security seria importante, se já existe Web Application Security? O mercado de tecnologia nos últimos anos viu transformações numa velocidade inimaginável a 10 anos atrás. Muito tem se falado sobre abordagens Cloud Native, Transformação Digital e API Economy. Tudo isso tem gerado novos desafios em segurança que a abordagem tradicional de segurança para aplicações web não é capaz de resolver. Os desafios vêm das interações, e as interações são feitas por APIs.

Há alguns anos, o Gartner afirmou que **em 2022, as APIs seriam o maior vetor de incidentes de segurança causando vazamento de dados.** Em 2021, a mesma instituição incluiu em seu modelo de referência para arquiteturas a camada de API Security, sendo que, anteriormente, ela colocava segurança para APIs como parte de soluções de Web Application Firewall (WAF). Tudo isso demonstra a necessidade de difusão do assunto. E você saberá mais sobre o assunto a seguir.

Na primeira parte deste documento, damos um panorama geral sobre web APIs: o que são, quais os principais usos e principais tipos. Depois, abordamos as principais ameaças para APIs segundo a OWASP e explicamos porque APIs precisam de uma abordagem diferente de aplicações web. Por fim, apresentamos algumas recomendações para melhorar sua prática em API Security.

O que é uma Web API?

Para entender o que é uma web API, precisamos dar um passo para trás e entender o que é uma API em si. **Segundo a Mozilla Foundation, APIs (Application Programming Interfaces) são construções que permitem que se interaja com uma aplicação através de software.**

Elas possibilitam conectar diferentes sistemas que podem se utilizar de diferentes linguagens de programação. O que poderia ser uma comunicação impossível ou inviável, encontra um ponto de conexão em comum usando este artefato.

Vejamos em uma contextualização.

Considere que um cliente GoCache queira ter uma integração com a CDN da GoCache para obter dados de tráfego e eventos de segurança, como essa comunicação é possível se ele não tem acesso ao código da CDN GoCache?

Simples, **através da API GoCache**, que disponibiliza mediante a regras de negócio o acesso a funções, dados e controle da CDN.

O que diferencia uma web API é a comunicação via web, usando principalmente o protocolo HTTP. Com a vida do ser humano cada vez mais cercada por aplicações web e mobile, as web APIs estão se tornando peça chave para grandes transformações na forma como se desenvolve e opera software. A seguir, apresentaremos os principais usos de APIs e os principais padrões em que sua comunicação ocorre.

Quais são os principais usos de Web APIs?

Modernização de arquiteturas de aplicações web

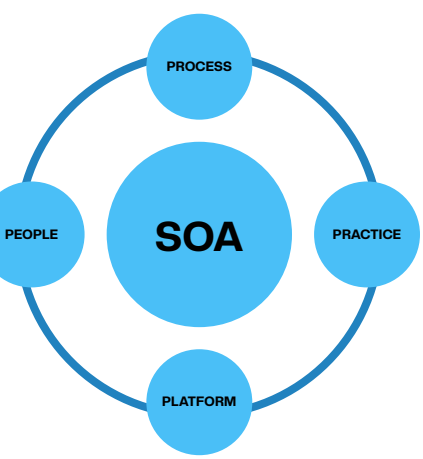
As primeiras aplicações web eram bastante estáticas. As páginas HTML já chegavam do servidor em sua forma final e mesmo que houvesse interatividade via javascript, para atualizar algum conteúdo da página, era necessário que toda sua estrutura fosse enviada novamente pelo servidor. Vamos entrar num acordo que há muito a ser melhorado nessa experiência. E foi justamente o motivo de uma grande evolução introduzida por volta de 2005.

Nessa época foi introduzido o AJAX (Asynchronous Javascript and XML). Essa tecnologia permitia a solicitação assíncrona de dados ao servidor, por meio de javascript e XML, evitando que toda a página tivesse que ser recarregada para atualizar apenas uma pequena parte do conteúdo. Apesar de não serem chamadas por esse nome neste período, pode-se considerar este uso um exemplo de web API. Mais tarde, no fim da década, a introdução dos smartphones trouxe os aplicativos móveis, cuja comunicação com o servidor ocorre de maneira similar ao AJAX.

De lá para cá, novos padrões e frameworks têm surgido, mas podemos agrupar essas tendências em um fenômeno: desacoplamento entre front-end e back-end, no qual os dados são fornecidos assincronamente via API e a página é renderizada ou no navegador, ou em um servidor dedicado para essa função. Desconsiderando outros aspectos, uma das principais vantagens dessa abordagem arquitetural é a agilidade ganha com o menor acoplamento. Quem desenvolve o front-end depende menos de quem trabalha com o back-end para desenvolver e testar novos produtos, e vice-versa. Equipes que definem o design das APIs antes de desenvolver (API First) se beneficiam ainda mais dessa arquitetura agilidade.

Outra característica de arquitetura das primeiras aplicações web é o monolito, ou seja, as aplicações eram majoritariamente constituídas por um bloco homogêneo de código, executado em um único servidor (ou replicado em mais servidores para escala horizontal). Quando se trata de aplicações pequenas, como por exemplo, um blog WordPress, essa abordagem não costuma gerar transtornos, sendo até preferível em grande parte dos casos. Mas quando falamos de aplicações mantidas por dezenas de desenvolvedores ou mais, a situação pode ficar um pouco mais complicada.

Isso começou a ser percebido com mais clareza na época do florescimento das metodologias ágeis e da cultura DevOps. Com dezenas de equipes autônomas trabalhando no mesmo código, é fácil enxergar o potencial que as dependências entre o trabalho delas têm de ser um obstáculo ao desenvolvimento ágil. E como, mesmo o deploy de uma pequena mudança impacta toda a aplicação, a frequência com que novas funcionalidades entram em produção também fica comprometida.



Houveram algumas tentativas de mudar esse cenário, como SOA (Service Oriented Architecture), mas não alcançaram o sucesso. O cenário começou a melhorar com bastante influência da arquitetura de contêineres. Os contêineres fornecem um ambiente muito mais leve, consistente e seguro para quebrar o monolito em dezenas, centenas e até milhares de pequenos serviços, os chamados "microsserviços". Tecnologias de orquestradores, como Kubernetes, tem auxiliado muito o trabalho de lidar com a complexidade criada por esse tipo de ambiente.

A velocidade desenvolvimento de software cresceu de forma assombrosa nos últimos anos, junto com a difusão dessa arquitetura. E onde entra o papel das APIs? São justamente elas as responsáveis pela comunicação entre os microsserviços. As APIs fornecem uma interface com um nível de abstração suficiente para diminuir o acoplamento entre as diferentes funções da aplicação sem comprometer a coesão do conjunto.

Trocas de valor entre empresas

É praticamente inviável para a maioria dos produtos que todos os seus componentes sejam produzidos pela mesma empresa. Você dificilmente verá um fabricante de celular fazendo seus próprios parafusos, por exemplo. Mas uma coisa que não é tão evidente, é que certas indústrias, como a automotiva e mais recentemente, a aeronáutica, descobriram que podem terceirizar o desenvolvimento e produção de componentes mais complexos e fundamentais para a funcionalidade de seu produto, buscando focar em suas competências chave.

Uma fabricante de aeronaves pode terceirizar o desenvolvimento e produção das asas de seu avião, diminuindo o aporte de capital necessário para desenvolver seu novo produto e podendo concentrar seu foco em aspectos mais globais dele. Mas o que isso tem a ver com APIs? O mesmo fenômeno acontece com a indústria de software e as APIs têm papel importante nisso.

Se voltarmos algumas décadas no tempo, veremos que mesmo nas aplicações web mais antigas, muitos componentes já eram fornecidos por terceiros. Bancos de dados, servidores web e sistemas operacionais seriam muito onerosos se cada empresa que criasse uma aplicação web tivesse que construí-los. E entre 1999 e 2000 houve um grande marco de evolução: a Salesforce surgiu como a primeira empresa que nasceu no modelo de Software como um Serviço (SaaS - Software-as-a-Service). Curiosamente, nesse mesmo ponto surgiu o que é considerado hoje como a primeira web API.

Basicamente, ao invés de oferecer um CD-ROM com um software de CRM a ser instalado nos servidores de seus clientes, a Salesforce trouxe a proposta de entregar a funcionalidade de seu

produto via internet. Desse momento em diante, principalmente nos últimos anos, vimos uma explosão de softwares sendo vendidos como serviços. Movimentos como Open Banking e Open Health só acentuam essa tendência.

Hoje, para criar um e-commerce, você não precisa criar seus próprios sistema de pagamentos, helpdesk, mecanismo de recomendação, etc. Tudo isso pode ser fornecido por terceiros, ajudando a focar no que importa, que no caso é garantir uma boa experiência de compra na loja virtual. Como nesse modelo o fornecedor opera seu próprio software, é imperativo que este processe os dados fornecidos por seus clientes. E são justamente as APIs que são a interface de troca de dados entre os dois lados dessa troca de valor.

Quais são os principais protocolos das web APIs?

REST (Representational State Transfer)

O REST não é exatamente um protocolo ou um padrão, mas sim um conjunto de restrições de arquitetura. Essas restrições facilitam o entendimento tanto para quem desenvolve, quanto para quem consome a API, mas sem limitar demais a liberdade do desenvolvedor. Esses ingredientes podem explicar em partes o grande sucesso em sua adoção, já que diferentes fontes citam que mais de 90% das APIs públicas são RESTful, mesmo considerando a emergência e crescimento de novos padrões.

Suas principais características são usar unicamente o protocolo HTTP para comunicação, seguir uma arquitetura cliente-servidor e serem sem estado (Stateless). Por arquitetura cliente-servidor, pode-se entender que temos um "cliente", que requisita um recurso da API, e um "servidor", que responde com a representação do estado atual daquele recurso. Por "stateless", pode-se entender que uma requisição de consulta não altera o estado da requisição seguinte.

Em uma API RESTful que segue boas práticas, cada recurso é representado pela URI da request (ex.: /users), enquanto o verbo HTTP indica que tipo de ação será feita sobre seu estado (ex.: GET para consultar e DELETE para apagar). Podem existir parâmetros na URI, na query string, no corpo da requisição e até nos cabeçalhos. Cabeçalhos, inclusive, sejam de requisição ou de resposta, são muito importantes, para ações como autenticação, autorização, declaração do tipo de conteúdo, código de status, entre outras funções. Esse tipo de API é bem flexível quanto ao formato do corpo da requisição ou da resposta, sendo que o formato mais usado é o JSON, seguido por XML.

SOAP

SOAP (Simple Object Access Protocol)

Antes do consumo e produção das APIs públicas explodir, as APIs SOAP reinavam. Ao contrário do REST, SOAP é um padrão bem definido e restritivo, mas aceita praticamente qualquer protocolo de comunicação além do HTTP. Isso remete à sua origem com propósito corporativo, quando se buscava uma interoperabilidade padronizada, para facilitar a integração de sistemas internos e de outras empresas. Suas limitações o fizeram perder espaço e sua liberdade na adoção de protocolos de comunicação não fez muita diferença, já que o HTTP foi o protocolo preferido para este tipo de API.

Uma das principais limitações do SOAP foi depender unicamente do formato XML, que é mais verboso, e por consequência, mais lento e pesado para processar do que, por exemplo, o JSON. Fora isso, o que deveria ser seu trunfo, que é a definição mais consistente de um padrão, acabou tornando-se seu algoz, pois a curva de aprendizado é maior e nem todas as pessoas que desenvolviam obedeciam o padrão com disciplina. O protocolo é dividido em 3 partes: envelope, cabeçalho e corpo.

O envelope é o elemento-raiz em cada mensagem SOAP, e pode conter declarações de namespaces e também atributos adicionais como o estilo de codificação, que define como os dados são representados no documento XML. O cabeçalho, que é opcional, traz informações adicionais, como por exemplo, se a mensagem deve ser processada por um determinado nó intermediário na rede. Por fim, o corpo, que é obrigatório, contém a informação a ser transportada para o seu destino final. Pode conter um sub-elemento opcional Fault, usado para carregar mensagens de status e erros retornadas pelos “nós” ao processarem a mensagem.

GRAPHQL

GraphQL (Graph Query Language)

O GraphQL é um dos padrões que desafiam o REST. Foi introduzido pelo Facebook, no modelo open-source e desde então tem sido adotado por empresas de grandes a pequenas. Seu principal trunfo é introduzir um modelo mais direto de comunicação, no qual o cliente requisita exatamente o que ele precisa do servidor, evitando a realização de muitas chamadas, e o retorno de dados desnecessários. Como seu objetivo é concentrar em apenas uma chamada dados de diferentes domínios de negócio, tem tido sucesso na comunicação entre front-end e back-end.

Seu funcionamento é similar a linguagens de consulta de banco de dados (SQL), só que aplicado para APIs. O cliente envia quais campos precisa e em que formato, e o servidor responde com estes campos. Além disso, o GraphQL proporciona aos profissionais responsáveis pela manutenção das APIs flexibilidade para adicionar ou preterir campos, sem afetar as consultas existentes. Os desenvolvedores podem criar APIs com o método que quiserem, pois a especificação do GraphQL assegura que elas funcionem de maneira previsível para os clientes.

O esquema do GraphQL é direto e formado com três tipos principais que suportam conjuntos distintos de ações: Mutações são usadas para criar, atualizar e excluir dados; As consultas são utilizadas pelo cliente para solicitar os dados do back-end; As assinaturas são uma maneira de criar e ter um relacionamento em tempo real com o servidor.

RPC

Protocolos baseados em RPC (Remote Procedural Call)

O RPC surgiu em uma época anterior aos SOAP. Diferente do REST em que a chamada é por dados (o estado do recurso, no caso), no caso do RPC, a chamada é por atividades ou funções, as quais podem ser chamadas pela aplicação como se fossem uma função local. XML-RPC e JSON-RPC são mais antigos e hoje são usados em poucos casos de uso (como no WordPress e Ethereum, respectivamente). Porém, este modelo de comunicação tem ressurgido com entusiasmo devido à variante gRPC, criada pelo Google, que encontrou na arquitetura de microsserviços um casamento quase perfeito.

O que diferencia o XML-RPC do JSON-RPC, são os formatos dos arquivos usados para transportar os procedimentos pela rede (XML e JSON, respectivamente). O gRPC tem algumas particularidades. Ao invés de usar XML ou JSON, esse padrão utiliza "Protobuf" (Protocol Buffers) como método de serialização para transferência de dados na rede. O protobuf pode ser até 30% mais enxuto que o JSON. O gRPC também depende do HTTP/2, tirando proveito de suas vantagens, como streaming bidirecional. Dessa forma, o cliente não precisa iniciar uma nova requisição a cada consulta.

A combinação do HTTP/2 com protobuf chega a ser 8 vezes mais rápido do que uma API RESTful se comunicando em HTTP/1. Essas características fazem com que gRPC seja uma boa escolha para arquiteturas de microsserviços, já que uma única chamada pode se desdobrar em uma cadeia de dezenas de comunicações entre serviços, onde as latências individuais se agregam na latência final. Por se tratar principalmente de comunicações dentro da própria empresa, a maior curva de aprendizado acaba sendo superada pela informação ser mais acessível, mesmo quando falta documentação. A ampla disponibilidade de SDKs em diferentes linguagens também ajuda. Porém, a adesão em APIs públicas ainda é baixa.

Webhooks

Uma das restrições que definem o funcionamento de uma API REST, é a arquitetura cliente servidor. Nesta abordagem, o cliente, normalmente o navegador, solicita o estado de um recurso da API, enquanto o servidor responde com o estado ou aplica alguma transformação sobre ele. Porém, isso é limitante em alguns casos de uso.

Por exemplo, vamos supor que um usuário está em uma página de um aplicativo de delivery esperando a atualização de seu pedido. Com isso, seu dispositivo teria que fazer consultas frequentemente para procurar atualizações em seu pedido. Os webhooks foram criados para evitar esse problema ao inverter o processo. Neste caso específico, o servidor do aplicativo poderia usar um webhook para enviar a informação sobre uma atualização do pedido ao cliente, sem necessidade de consulta.

WEB HOOKS

Websockets

O padrão requisição/resposta do protocolo HTTP também limita outros tipos de aplicativos, como os que se comunicam em tempo real. Imagine um jogo online ou aplicativo de mensagens. São necessárias inúmeras trocas de dados, às vezes até ao mesmo tempo. O Websocket é um protocolo que guarda algumas semelhanças com o HTTP, como a conexão TCP, o uso das portas 80 e 443 e de servidores web. Mas as semelhanças não vão muito além disso.

A tecnologia não força que exista uma requisição para haver resposta. O cliente pode enviar mensagens e receber respostas orientadas a eventos, sem necessariamente um estar relacionado ao outro e até ao mesmo tempo. Toda a comunicação acontece a partir de uma sessão de comunicação interativa entre um cliente e um servidor.

WEB SOCKETS

Quais são as principais ameaças para APIs?

Não é segredo que as APIs estão sob risco de ataque. O Gartner já estimou que em 2022 as APIs serão o maior vetor de vazamentos de dados. Porém, a discussão sobre quais são os riscos e como se proteger contra eles ainda é muito tímida nas empresas. O problema só piorará à medida que as APIs se tornarem mais complexas e mais empresas confiarem nelas para funções críticas de negócios. Os riscos de segurança aumentam exponencialmente.

Para saber como se proteger, é fundamental conhecer quais são os principais riscos. Muitos ainda acreditam que Web Application Security é suficiente para lidar com os riscos de segurança de API, o que não é verdade. A OWASP, conhecida por suas publicações periódicas a respeito das 10 principais ameaças para aplicações web, possui uma lista exclusiva para APIs, fato que ainda é pouco difundido. Conheça nas próximas páginas as Top 10 ameaças para a segurança de APIs listadas pela OWASP.

API1:2019

Broken Object Level Authorization (BOLA)

O BOLA, traduzido para o português como Autorização Quebrada a Nível de Objeto, é um tipo específico de Quebra de Controle de Acesso (Top 1 na lista da OWASP para aplicações web). É uma vulnerabilidade idêntica à conhecida como IDOR (Insecure Direct Object Reference ou Referência Direta ao Objeto Insegura), que já esteve no Top 2 em uma das primeiras listas para aplicações web.

Pense no objeto como sendo o estado de um recurso da API, como um dado pessoal de um usuário, um comentário ou uma compra. Quando um objeto tem sua autorização quebrada, significa que alguém que não tem permissão sobre ele possa acessá-lo se referencia-lo diretamente, mesmo estando autenticado.

É como se houvesse um condomínio fechado em que todas as casas estivessem destrancadas. Um visitante malicioso pode conseguir a credencial de um morador para atravessar o portão do condomínio (autenticação). Uma vez dentro do condomínio, ele é capaz de entrar não só na casa do dono dessa credencial, como em qualquer casa, já que estão destrancadas (quebra na autorização).

Agora, imagine uma API em que você forneça seu CPF para obter seus dados financeiros. Se ela estiver vulnerável ao BOLA, para um agente malicioso obter dados de qualquer cliente, basta oferecer o CPF dele, mesmo que tenha se autenticado como outro. Como você pode ver, uma vez identificada, é uma vulnerabilidade fácil de explorar e escalar. E de grande impacto: sua aplicação vai desde vazar dados importantes a criar transações em nome de outras pessoas.

Ela também é bastante comum. Principalmente quando falamos de aplicações complexas, formadas por muitas APIs, desenvolvidas por diferentes times e com deploys frequentes, a probabilidade de algo ser posto em produção com essa falha em algum momento é alta. E por se tratar de uma falha na lógica de negócio, não é fácil detectar sua presença de forma escalável, seja em testes unitários ou a nível de análise de tráfego.

API2:2019

Broken User Authentication

Para que se faça o correto controle de acesso, é fundamental que se identifique quem está solicitando acesso. Não fazer a identificação de usuário em uma API exibe dados pessoais e sensíveis, ou não ter controle contra alguém que viole as credenciais de outros usuários outro usuários, é o que se chama de Quebra de Autenticação de Usuário ou Broken User Authentication.

Por incrível que pareça, é comum haver APIs que não deveriam ser anônimas sem autenticação. Pode acontecer tanto involuntariamente, quando o time de produto e desenvolvimento se esquece deste componente na fase de concepção, como pode acontecer voluntariamente, pelo time considerar equivocadamente que nenhum atacante irá descobrir a API, o que costuma se chamar de "segurança por obscuridade".

Por outro lado, há APIs que possuem processo de autenticação, porém ele não é robusto o suficiente para resistir a um ataque. Agentes maliciosos podem se aproveitar do uso de credenciais roubadas, sequestro de sessão, senhas fracas, tokens de API com baixa entropia, ausência de rate-limiting e de autenticação de 2 fatores, entre outras formas de se violar a autenticação de terceiros. O impacto causado por este tipo de ataque é similar ao da vulnerabilidade anterior.

BROKEN USER
AUTHENTICATI

API3:2019

Excessive Data Exposure

Excessive Data Exposure, Exposição excessiva de dados em português, é quando um aplicativo revela mais informações do que o necessário para o usuário por meio de uma resposta de API. Isso pode incluir dados confidenciais, como números de segurança social, números de cartão de crédito e credenciais de login. Quando essas informações são divulgadas, agentes mal-intencionados podem usá-las para explorar o usuário de alguma forma, como roubo de identidade ou fraude financeira.

Os desenvolvedores de API geralmente cometem o erro de expor todas as propriedades do objeto sem considerar sua sensibilidade individual e confiam no código do lado do cliente para realizar a filtragem de dados. O problema é que agentes maliciosos podem instalar proxies em computadores ou smartphones, tendo acesso aos dados que não são exibidos pelo front-end. Além disso, combinada com outras vulnerabilidades, a exposição excessiva pode ampliar o estrago de uma vulnerabilidade, como por exemplo, exibir dados de cartão de crédito em uma API de dados pessoais com autenticação quebrada.

EXCESSIVE DATA EXPOSURE

gocache

API4:2019

Lack of Resources & Rate Limiting

Lack of Resources & Rate Limiting, em português, significa falta de recursos ou de rate-limiting. Toda API consome recursos que são limitados, como CPU, memória e armazenamento. Quando o consumo de algum desses recursos chega ao limite, a API começa a falhar, trazendo prejuízo à experiência dos usuários. Existem duas formas de exaurir uma API: por meio de consultas onerosas ou por meio do volume de consultas. Por isso é importante o dimensionamento correto do back-end e forçar limites de uso que o mantenham longe do limite suportado.

Quando o uso excessivo da API é intencional, com o objetivo de derrubar ou prejudicar sua experiência, chamamos isso de Ataque de Negação de Serviço ou Denial of Service (DoS ou DDoS quando se usam diversos IPs para atacar, ou seja, distribuído). A falta de limitação da taxa de requisições, o que chamamos de rate-limiting, também abre portas para outros objetivos maliciosos, como ataques por brute-force, roubo de credenciais e enumeração, todos ataques que dependem de tentativa e erro para descobrir informações como senhas e vulnerabilidades.

LACK OF RESOURCE
RATE LIMIT

API5:2019

Broken Function Level Authorization (BFLA)

O BFLA, traduzido para o português como Autorização Quebrada a Nível de Função, é outro tipo de Quebra de Controle de Acesso. É parecido com o BOLA, porém, enquanto o alvo do BOLA são os objetos com que a API interage, o alvo do BFLA são as funções da API. Para explorar um caso desse tipo, o atacante acessa APIs que executam ações às quais seu usuário não tem privilégio, por exemplo, acessar uma API administrativa tendo privilégio de usuário comum.

Também é um caso de BFLA um usuário anônimo executar ações restritas apenas a usuários autenticados. Um usuário acessar recursos aos quais ele não teria privilégio em sua própria conta gera apenas prejuízos localizados na receita da empresa, porém, se um usuário que não é administrador obter privilégios administrativos, ele pode gerar um dano muito maior, ao poder visualizar informações, modificar e até excluir outras contas. Outra forma grave de explorar essa vulnerabilidade é acessar objetos de outras contas que seriam só para visualização, como comentários de redes sociais, e poder modificá-los ao alterar o método da requisição (ex.: substituir GET por DELETE).

BROKEN FUNCTION LEVEL AUTHORIZATIO

gocache

API6:2019
Mass Assignment

A vulnerabilidade de Atribuição em Massa, também chamada de Mass Assignment, ocorre quando um aplicativo da Web usa um conjunto de dados fornecidos pelo usuário e os atribui automaticamente a variáveis no código do aplicativo sem a higienização adequada. Essa falha de segurança permite que um invasor injete dados maliciosos nos campos de entrada de um aplicativo, resultando na execução de ações não intencionais ou na exposição de dados confidenciais.

Os dados maliciosos podem ser parâmetros ocultos que se relacionem diretamente a desde um campo privado do banco de dados, quanto uma variável interna que eleve os privilégios do usuário. Frameworks de desenvolvimento são especialmente vulneráveis, pois incorporam essas relações para acelerar o processo de criação de software, o que pode passar despercebido pelos engenheiros.

MASS ASSIGNMENT

API7:2019
Security Misconfiguration

Configuração Incorreta de Segurança, ou Security Misconfiguration, representa uma série de erros como cabeçalhos HTTP mal configurados, cabeçalhos desnecessários, mensagens de erro verbosas, bancos de dados abertos à internet entre inúmeros outros exemplos. Atacantes exploram essa vulnerabilidade buscando conhecer mais profundamente a API em busca de informações que direcionam melhor seu ataque e brechas que sirvam de porta de entrada.

Esse problema pode ser particularmente delicado em operações grandes e complexas. Por mais que haja validações manuais e automáticas para identificá-lo em diferentes instâncias, esses processos abrangem apenas erros de configuração conhecidos e divulgados. A quantidade de partes móveis também aumenta a probabilidade de alguma área estar sujeita a erros. Por fim, as falhas de configuração podem estar escondidas nas interações entre essas partes móveis, ocultando o problema.

SECURITY
MISCONFIGUR

gocache

API8:2019
Injection

A Injeção é um antigo problema do domínio de segurança para aplicações web que também acontece com frequência em APIs. Ocorre quando uma API recebe entrada de uma fonte não confiável e a usa de forma não segura, executando um comando que não é previsto no funcionamento correto, cujo objetivo é malicioso. Isso pode conduzir, por exemplo, à exposição de informações sensíveis, perda de dados e quebra de autenticação.

Os vetores de ataque típicos de ameaças maliciosas de injeção de API incluem SQL, comandos do Sistema Operacional, LDAP, NoSQL, entre outros. Usando como exemplo a injeção de SQL, um invasor envia uma solicitação contendo um trecho de sentença SQL que pode ser interpretado literalmente, estendendo a sentença que seria enviada ao banco de dados com um trecho com objetivos maliciosos.

INJECTION

API9:2019

Improper Assets Management

Se suas APIs estão mal documentadas, pode ter certeza de que alguém queira atacá-las fará uma documentação mais completa usando engenharia reversa. É disso que se trata a vulnerabilidade Improper Assets Management ou Gestão de Ativos Imprópria. Para se estar à frente dos ataques, é fundamental ter conhecimento de todo o ambiente para se avaliar com precisão quais APIs oferecem mais risco e priorizar ações sobre elas.

Com frequência, empresas se deparam com APIs que nunca foram documentadas, principalmente internas, APIs com documentação desatualizada e APIs que já foram descontinuadas, mas ainda não foram tiradas de produção. Isso pode oferecer grandes riscos. APIs descontinuadas deixam de receber patches de segurança, mesmo estando em produção. APIs sem documentação podem conter falhas de configuração como ausência de rate-limiting e monitoração. APIs com documentação desatualizada podem conter vulnerabilidades escondidas em parâmetros ocultos.

IMPROPER ASSETS MANAGEMENT

API10:2019

Insufficient Logging & Monitoring

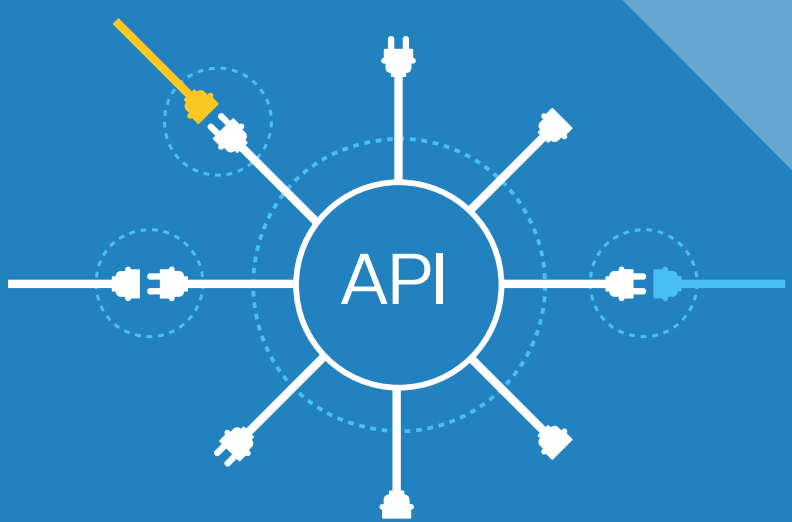
Grande ataques normalmente não acontecem rapidamente. Existe um tempo para reconhecer o terreno, encontrar uma brecha, escalar horizontalmente, comprometer sistemas. Quanto mais o atacante voa abaixo do radar, mais chances ele tem de atingir seus objetivos. Insufficient Logging and Monitoring, ou Registro e Monitoração Insuficiente tratam sobre a falta de sensibilidade deste radar.

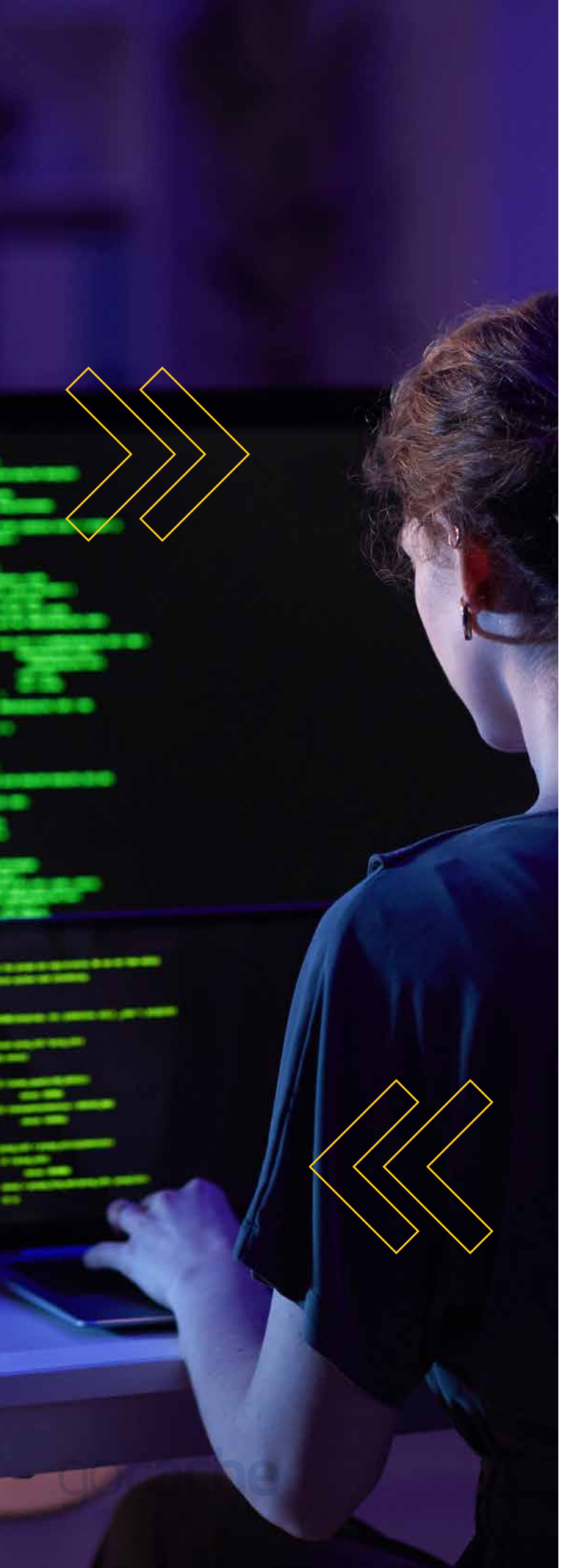
Quanto mais completo e objetivo é o sistema de registro e monitoramento das APIs, maiores as chances de se detectar comportamentos anormais que atacantes exibem em sua jornada. Quanto mais cedo um agente malicioso é identificado e bloqueado, melhor. Por mais que se deseje aplicações seguras por padrão, e soluções de mitigação que trabalhem em tempo de execução, não é possível garantir 100% de efetividade nessas estratégias, por isso registro e monitoramento são indispensáveis.

INSUFFICIENT LOGGING & MONITORING

gocache

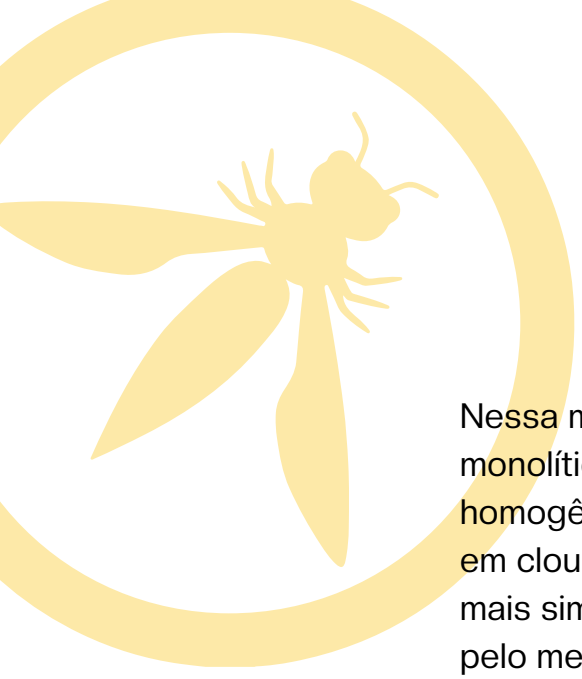
Por que API Security difere de Web Application Security?





Se você já conhecia a OWASP Top 10, mas não sabia que havia uma lista específica da entidade para APIs, provavelmente você deve ter estranhado. Porém, quando você pensa em termos de arquitetura da aplicação, faz todo o sentido haver uma lista dedicada para enfrentar problemas específicos de API.

A OWASP Top 10 para aplicações web surgiu em uma época em que a maioria das aplicações seguia uma arquitetura em que o front-end era renderizado no servidor. Sendo assim, as regras de negócio eram todas processadas no servidor, que entregava os arquivos HTML já com os dados incorporados. Isso limitava muito a superfície de ataque, sendo necessário um pouco de engenhosidade para conseguir manipular dados, por isso a prevalência de SQL Injection e XSS.



Nessa mesma época, as aplicações eram majoritariamente monolíticas, sendo executadas em um grande bloco de código homogêneo, dentro de máquinas em um datacenter ou em VMs em cloud. Os pontos de entrada na aplicação eram poucos, sendo mais simples de controlar. Era simples passar todo o tráfego pelo mesmo nó de uma ferramenta de WAF e normalmente, autenticação e autorização eram realizadas em um ponto único na entrada da aplicação. Problemas de autenticação e autorização já se faziam presentes, mas não com a mesma força de injections.

O que muda com as APIs? Dada a função delas, os dados são expostos de forma bem mais direta do que em uma página renderizada no servidor. Uma simples falha de quem programa já é suficiente para expor informações sensíveis sem muito trabalho de quem está explorando. Talvez você esteja se perguntando por que isso é um problema, dado que o que é mais simples também é mais fácil de revisar a qualidade. O grande problema é que a evolução das APIs vem acompanhada com o crescimento da complexidade.

As APIs diminuem o acoplamento das funções de uma aplicação, permitindo uma distribuição mais eficiente do trabalho de desenvolvimento, o que acelera a entrega de novos produtos a um nível que antes era inviável. Como consequência, geramos uma malha cada vez mais complexa de comunicação entre back-end e front-end e dentro do próprio back-end.

O tráfego externo que antes era forçado para entrar em apenas um nó, hoje pode ter à disposição inúmeros pontos de entradas (às vezes desconhecidos). O controle e conhecimento pleno da arquitetura que uma equipe de segurança, se perdeu como consequência da velocidade com que novos produtos são lançados. As assinaturas de padrões de ataque, normalmente associados às linguagens e frameworks específicos utilizados em um monolito, perderam eficiência em um ambiente que pode envolver a comunicação de microsserviços construídos em diferentes linguagens.

Como exemplo das consequências deste fenômeno, vemos que a OWASP Top 10 para APIs tem uma presença muito forte de falhas de autenticação e autorização quando comparada com a OWASP Top 10 para aplicações web de 2017. Provavelmente, a ascensão da Quebra de Controle de Acesso à primeira posição na lista para aplicações web divulgada em 2021 tenha relação com a importância das APIs nas aplicações web hoje. O processo de autenticação e autorização em aplicações distribuídas (que muitas vezes estão por trás de APIs) pode ser extremamente desafiador em relação ao mesmo processo em um monolito.

Isso não significa que ferramentas de Web Application Security não funcionam mais. WAF (Web Application Firewall), SAST (Static Application Security Testing), DAST (Dynamic Application Security Testing) entre outras soluções para segurança de aplicações web continuam a ter o seu papel, uma vez que os problemas solucionados por elas continuam existindo, principalmente no contexto individual de cada microsserviço. O que muda é que com a complexidade trazida pelas arquiteturas de aplicação viabilizadas pelas APIs, novos comportamentos emergem, sendo necessárias novas abordagens para mitigar os riscos.

O que você pode fazer para proteger APIs?

Agora, vamos para a prática! Trazemos nesta seção, algumas recomendações de como começar a melhorar suas práticas para trazer mais segurança às suas APIs.

A lista não é exaustiva, mas sim, uma coletânea baseada em práticas frequentemente recomendadas e algumas abordagens comuns que devem ser evitadas.

Reconheça a importância de API Security

API security não é aplicar uma camada de Web Application Security sobre APIs. É de surpreender a quantidade de empresas que consideram como abordagem suficiente para proteger APIs, fazer a comunicação via TLS, instalar um WAF e fazer autenticação. Algumas empresas vão um pouco além e adotam uma cultura Shift Left, usando Application Security Testing, como SAST e DAST.

Enquanto isso, a quebra de autorização é considerada a maior ameaça pela OWASP, seja na lista focada em APIs, ou na tradicional lista para aplicações web, divulgada em nova versão recentemente. Muitos especialistas em segurança ofensiva afirmam que virtualmente todas as aplicações apresentam alguma forma dessa vulnerabilidade. Autorização é algo diretamente associado à lógica de negócio, portanto, é difícil de ser detectada sem entendimento do contexto, algo que o modelo baseado em assinaturas de ataques comum não suporta.

Isso não significa necessariamente que a qualidade de desenvolvimento de software tem diminuído. A ascensão da importância da segurança para APIs é consequência da evolução na velocidade de desenvolvimento e entrega de software. O lançamento contínuo de novas funcionalidades, por dezenas e até centenas de equipes que trabalham de forma semi-autônoma, em uma arquitetura heterogênea, criam um cenário complexo em que a probabilidade de algo apresentar um comportamento inesperado se eleva. Pesquise sobre o assunto e difunda-o dentro de sua empresa.

Desenvolva uma cultura de API Security

Provavelmente, se você pesquisar sobre o assunto, descobrirá que API Security não é um jogo individual. Se apenas o time de segurança abraçar, este estará sempre correndo atrás do desconhecido, pois sua capacidade de descobrir e documentar novos elementos na aplicação sempre será menor que a capacidade de todos os times de produto da empresa introduzirem novos elementos.

Se for abraçado apenas pelo time de operações, este não terá o conhecimento especializado em segurança, necessário para saber identificar o risco e priorizá-lo, gerando overhead e falta de efetividade. Se for conduzido pelos times de produto e desenvolvimento, não há garantia de coesão e coordenação, tornando-se uma iniciativa pontual e cheia de pontos cegos.

Envolva toda a cadeia responsável por criar e manter as APIs. Quem concebe o produto sabe enxergar como o produto deve funcionar, e principalmente, como ele não deveria funcionar, além de poder evitar a introdução de falhas na lógica de negócio, a partir do aprendizado com incidentes anteriores. Quem opera, conhece a arquitetura o suficiente para identificar possíveis pontos cegos na estratégia de segurança para APIs. E quem gerencia a segurança, saberá identificar as ameaças e preparar uma resposta ao incidente.

Crie uma estratégia

Não existem soluções mágicas. Principalmente em se tratando da natureza contextual da segurança para APIs, em que as ameaças variam de acordo com a lógica de negócio de cada API. Por isso, é importante que você monte uma estratégia de identificação e resposta a incidentes de segurança em APIs que se molde às características de comportamento e arquitetura das APIs de sua empresa. Não terceirize essa responsabilidade para apenas uma solução off-the-shelf, ainda mais se ela for uma caixa preta.

Isso não significa que você não precisará contar com alguma solução de mercado para lidar com o problema. Mas antes, você precisa analisar muito bem o cenário, para escolher as soluções que melhor se encaixem em seu caso. Mapeie suas deficiências, estipule metas que indiquem que os incidentes estão realmente sendo detectados e mitigados com a menor quantidade possível de falsos positivos e elabore um playbook que cerque alguém que tente atacar suas APIs.

Tenha o inventário completo e atualizado de suas APIs

Há uma máxima que diz que o que não é conhecido não pode ser protegido. E isso é fato quando se trata de APIs. Dada a crescente velocidade de desenvolvimento e implementação de novos produtos digitais, e por consequência, APIs, fica cada vez mais difícil ter controle de tudo o que é lançado ou descontinuado. Ao ter a visão do todo, você pode priorizar com mais facilidade o que oferece mais risco para sua empresa.

Mas veja bem: não faça isso como uma iniciativa pontual! A grande questão que eleva a importância da segurança para APIs é sua alta mutabilidade. Se as APIs fossem um ambiente estático, provavelmente este assunto nem estaria sendo discutido. Um time de produto pode ter introduzido novos endpoints e não ter documentado, o que se chama de shadow APIs ou APIs fantasmas. Neste caso, uma falha de autorização pode ter passado despercebida, e mais ninguém dentro da empresa pode identificá-las, pois ninguém mais tem conhecimento sobre essa API.

Outro time pode ter lançado uma nova versão da API e não ter estipulado um prazo para retirar a antiga do ar, o que se chama de zombie APIs ou APIs zumbi. Essa versão antiga pode seguir em produção sem aplicação de patches de segurança, por ninguém mais lembrar da existência dela. Dada essa característica, é fundamental apoiar o inventário das APIs com um processo contínuo de descobertas de novas APIs, o que se chama de API discovery.

Pratique observabilidade com o contexto adequado

Reforçamos novamente aqui a natureza contextual das vulnerabilidades de APIs. A observalidade sob um escopo mais generalista é importante para detecção de muitos comportamentos inesperados em APIs e aplicações web. Mas não é suficiente para identificar boa parte das grandes ameaças que APIs são vulnerabilidades. Para isso, você necessita de um pouco mais de profundidade e customização.

Por exemplo, se você sabe que em um microsserviço específico do seu ambiente, cada usuário pode acessar apenas um objeto específico, se um token de identificação tentou acessar mais de um objeto, isso pode indicar uma possível tentativa de violação. Ou, se um comentário em uma rede social só pode ser editado pelo autor, dois usuários diferentes acessando o objeto em requisições do tipo PUT ou PATCH também podem levantar suspeitas. A Inteligência Artificial pode ser aliada para identificar esses casos.

Fontes

How REST replaced SOAP on the Web: What it means to you
https://www.infoq.com/articles/rest-soap/

Different types of API Protocols
https://iq.opengenus.org/different-types-of-api-protocols/

API Security Tutorial
https://www.wallarm.com/what/api-security-tutorial

What are the types of APIs and their differences?
https://www.techtarget.com/searchapparchitecture/tip/What-are-the-types-of-APIs-and-their-differences

Types of APIs
https://rapidapi.com/blog/types-of-apis/

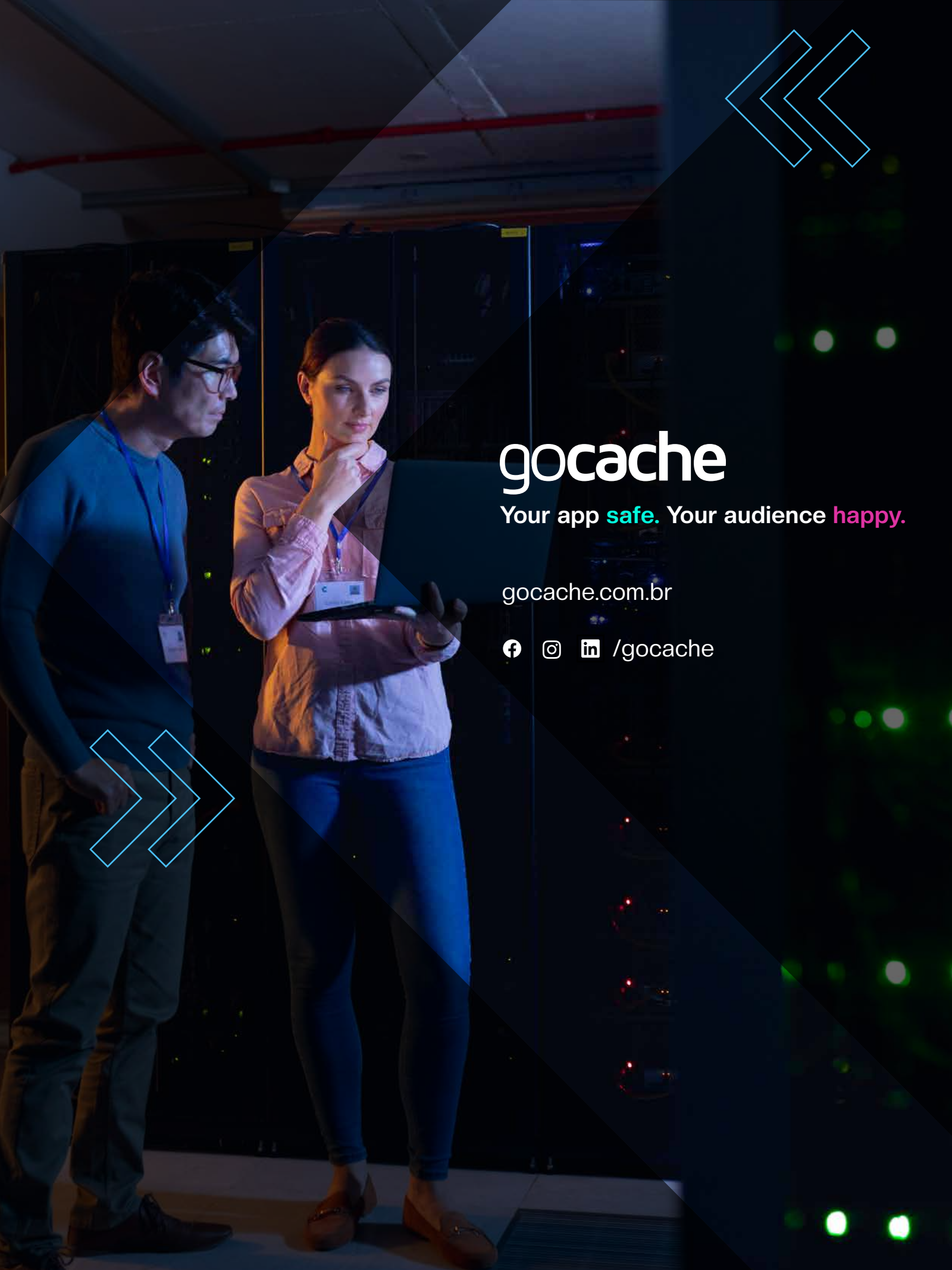
Types of APIs & Popular REST API Protocol
https://stoplight.io/api-types

WebHook vs API: What's the Difference? Which is Better?
https://buttercms.com/blog/webhook-vs-api-whats-the-difference

The Evolution of APIs: From RPC to SOAP and XML
https://konghq.com/ebooks/apis-rpc-soap-xml

The Evolution of APIs: From the Cloud Age and Beyond (PDF)
https://konghq.com/wp-content/uploads/2021/11/The-Evolution-of-APIs-From-the-Cloud-Age-and-Beyond.pdf

Protobuf — Uma alternativa ao JSON e XML
https://medium.com/trainingcenter/protobuf-uma-alternativa-ao-json-e-xml-a35c66edab4d


gocache
Your app safe. Your audience happy.
gocache.com.br
/gocache