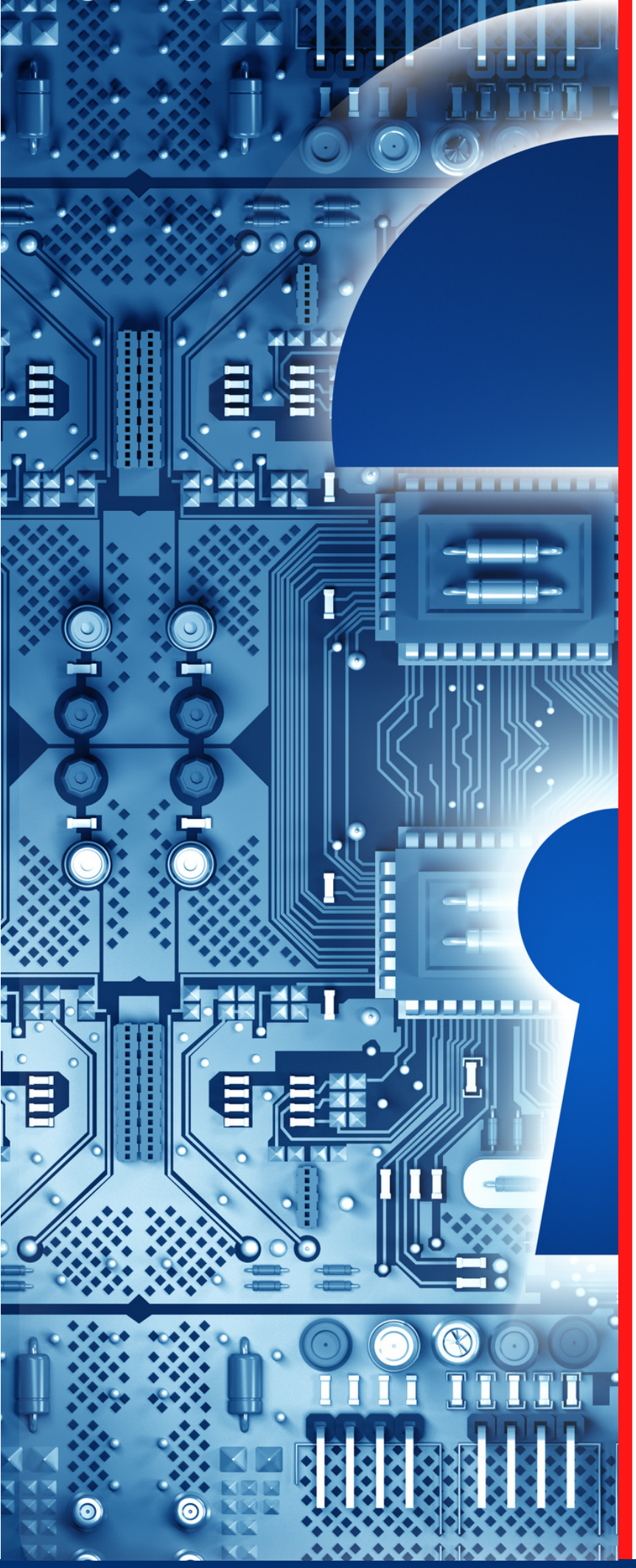


OSIC

ORIENTAÇÃO DE SEGURANÇA DA INFORMAÇÃO E CIBERNÉTICA

10/2023

***Application Programming Interface* (APIs)**

GOVERNO FEDERAL

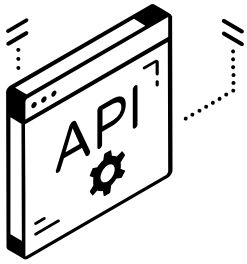
BRASIL

UNIÃO E RECONSTRUÇÃO

Espaço cibernético inclusivo, seguro, estável, acessível e pacífico.

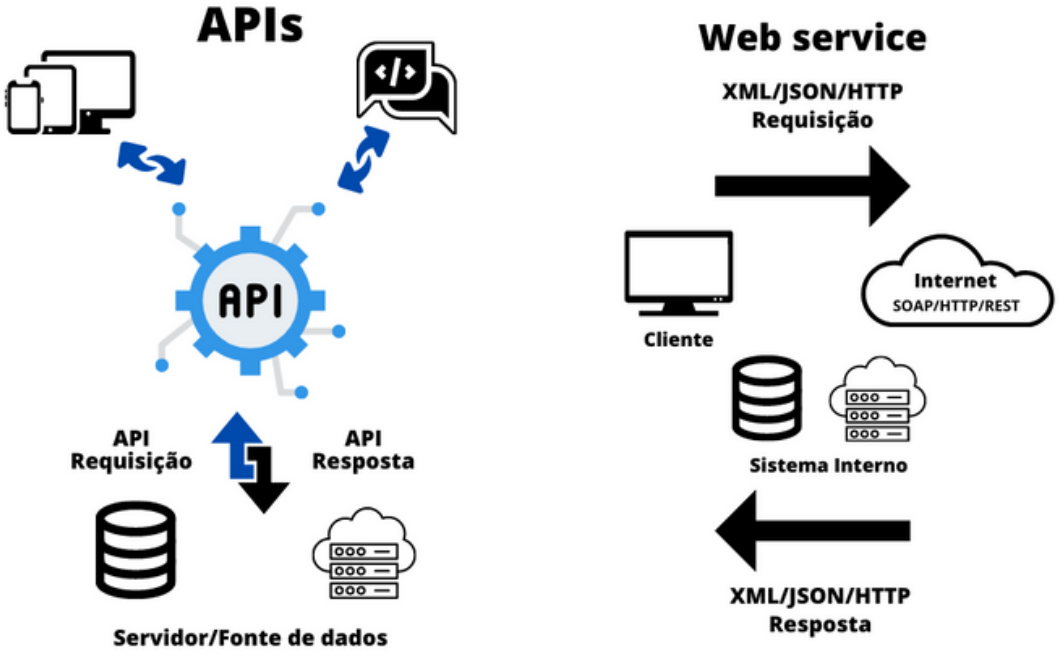
TLP:CLEAR

Introdução



Uma interface de programação de aplicativos, ou simplesmente API (*Application Programming Interface*), tem por objetivo disponibilizar recursos de uma aplicação para serem usados por outra, abstraindo os detalhes da implementação e, muitas vezes, restringindo o acesso a esses recursos com regras específicas.

Um *Web Service* (Serviço Web), que se caracteriza como uma API, é uma interface projetada para se comunicar via rede. Tipicamente, o protocolo HTTP é o mais utilizado, mas também pode ocorrer o uso de SOAP, REST e XML-RPC como meio de comunicação. A figura abaixo permite uma visão geral de um API e de um *Web Service*.



Inicialmente, a maioria das organizações usava as APIs em rede privada segura ou as acessava por meio de canais de comunicação seguros. Todavia, a transformação digital promoveu as APIs para uso remoto, particularmente para atender parceiros, fornecedores ou clientes.

Principais tipos de APIs

Existem seis tipos principais de APIs:

1. API aberta (*Open* API): pode ser utilizada por qualquer pessoa ou organização interessada; não há restrições de acesso, pois é gratuita e com código fonte aberto ao público.

2 API pública: apesar de similar a API aberta, nem sempre poderá ser de uso gratuito, pois provedores podem cobrar pelos serviços por meio de assinaturas mensais ou de tarifas. Seu uso costuma ser regulado por termos e condições de uso estabelecidas pelo provedor. Também envolve com frequência o uso de chaves ou processos similares de autenticação. A API pública tende a ser mais robusta que a API aberta.

3 API privada: também conhecida como API interna, tem acessibilidade restrita. É desenvolvida para uso interno de uma organização e costuma ser acessível apenas na rede interna da própria organização.

4 API de parceiros: assim como a API privada, também tem acessibilidade restrita. No entanto, ao contrário da API privada, não é direcionada ao público interno da organização que fornece a API, pois geralmente visa suprir necessidades de seus parceiros comerciais e estratégicos. Seu uso costuma ser altamente regulado e sujeito aos termos e condições do provedor. Normalmente, conta com alto nível de segurança para garantir a privacidade do provedor e do usuário.

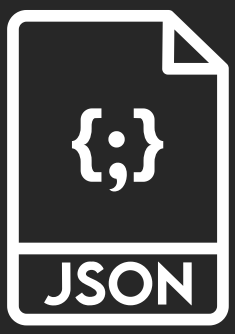
5 API composta: esse tipo combina diferentes APIs em uma única API de chamada (API *Call*), com o objetivo de criar uma sequência de operações relacionadas ou interdependentes. Esse tipo de API pode ser benéfica para lidar com comportamentos de APIs complexos ou que, em um determinado momento, estejam fortemente associados. Geralmente, é utilizada para agilizar o processo de execução e melhorar o desempenho dos listerners nas interfaces *web*.

6 API unificada: semelhante à API *Call*, agrupam várias APIs, no entanto também agrupam recursos de *backend* em uma única interface. Portanto, são agregadores, frequentemente agrupando funcionalidades comuns ou APIs de um setor específico. As APIs unificadas levantam algumas preocupações logísticas e riscos de segurança, devendo ser oferecidas camadas de abstração adicionais e criptografia em nível de campo.

Formato de troca de dados (XML e JSON)

Os principais formatos de troca de dados usados no desenvolvimento de aplicativos da *web* são:

XML: é o formato de troca de dados usado no desenvolvimento de aplicativos da *web*. XML significa "Linguagem de Marcação Extensível" (e*Xtensible Markup Language*). É uma extensão da *Standard Generalized Markup Language* (SGML). Ele contém um conjunto de regras para codificar um documento para que seja legível por humanos e máquinas. Foi consolidado pelo *World Wide Web Consortium* (W3C), com o objetivo de criar um tipo de formato que poderia ser lido por *software* e que tivesse flexibilidade e simplicidade, visando entre outras coisas a possibilidade de criação de *tags* e a concentração na estrutura da informação e não em sua aparência. Por meio do uso misto de atributos e elementos, o XML suporta objetos junto com namespaces compatíveis e comentários; e



JSON: é um acrônimo para "*JavaScript Object Notation*", é um formato de padrão aberto que utiliza texto legível a humanos para transmitir objetos de dados consistindo de pares atributo-valor. O formato está formalizado em RFC (*Request for Comments*) e também em norma ISO, possuindo uma sintaxe mais leve que o XML, suportando objetos mas não comentários ou namespaces. Em termos de simplicidade e velocidade de análise, o JSON tem vantagem na maioria dos casos. Sua sintaxe é geralmente menos detalhada porque o XML requer tags de início e fim para cada ramificação da árvore de dados, enquanto o JSON representa dados em *arrays* com pares nome/valor (vide exemplo abaixo).

XML

```
<lista_pessoas>
  <pessoas>
    <pessoa>
      <nome>Ana Maria</nome>
      <sexo>F</sexo>
      <idade>40</idade>
    </pessoa>
    <pessoa>
      <nome>Bruno Soares</nome>
      <sexo>M</sexo>
      <idade>31</idade>
    </pessoa>
    <pessoa>
      <nome>Carlos Magno</nome>
      <sexo>M</sexo>
      <idade>29</idade>
    </pessoa>
  </pessoas>
</lista_pessoas>
```

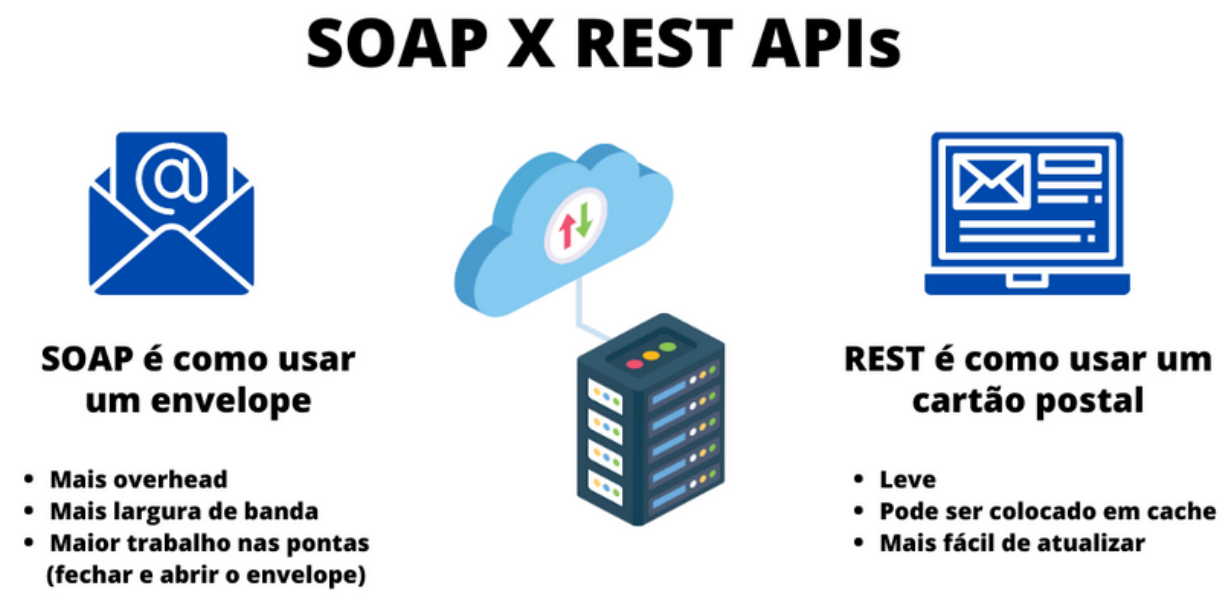
JSON

```
{ "lista_pessoas" :
  {
   "pessoas" : [
    {
     "nome" : "Ana Maria",
     "sexo" : "F",
     "idade" : 40,
    },
    {
     "nome" : "Bruno Soares",
     "sexo" : "M",
     "idade" : 31,
    },
    {
     "nome" : "Carlos Magno",
     "sexo" : "M",
     "idade" : 29,
    },
          ]
  }
}
```

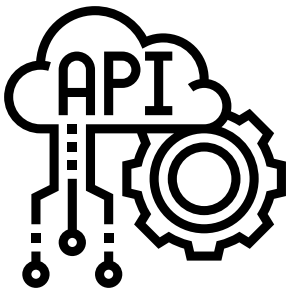
Protocolos e Arquiteturas de APIs

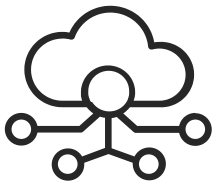
Atualmente, existem três categorias de protocolos ou arquiteturas de APIs:

- *Representational State Transfer* (REST);
- *Simple Object Access Protocol* (SOAP); e
- *Remote Procedural Call* (RPC).



O REST não é um protocolo ou padrão, mas sim um conjunto de princípios de arquitetura. É considerada a abordagem mais utilizada para a construção de APIs e possui uma arquitetura Cliente-Servidor. O cliente e o servidor comunicam-se via HTTP (protocolo de transferência de hipertexto), usando identificadores de recursos exclusivos ou *Uniform Resource Indentifiers* (URIs), operações básicas de persistência "criar, ler, atualizar, excluir" (ou CRUD, no acrônimo em inglês) e convenções JSON. Cliente e servidor devem ser independentes um do outro, ou seja, as alterações realizadas em um não podem afetar no outro. Além disso, o cliente deve armazenar em cache as respostas, o que contribui com uma experiência de usuário mais rápida e eficiente. O REST é direcionado a dados e necessita de baixa largura de banda.





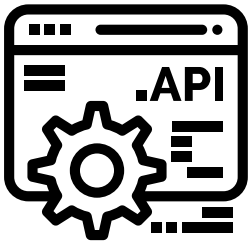
O protocolo SOAP, por sua vez, possui um padrão altamente estruturado, rigidamente controlado e claramente definido, o que o torna bastante diferente da flexibilidade fornecida pela API REST. Devido às suas regras rígidas, a API SOAP é mais utilizada quando uma organização requer uma segurança rígida para troca de dados mais complexa. Assim, os desenvolvedores frequentemente usam SOAP para APIs internas ou de parceiros.

Já o protocolo RPC permite a execução remota de uma função num contexto externo ao seu, ou seja, permite que o cliente execute o código num servidor. O protocolo RPC pode utilizar tanto o XML quanto o JSON, e passa a ser classificado como:

- XML-RPC: utiliza um formato XML específico para transferir dados, diferentemente do SOAP, que usa um formato XML proprietário. Além de ser mais antigo que o SOAP, o XML-RPC utiliza baixa largura de banda e possui formato mais simples que SOAP; e
- JSON-RPC: ao contrário do XML-RPC, ele não está restrito ao HTTP como protocolo de transferência.

Superfície de Ataque

As APIs evoluíram para um mecanismo importante para organizações e serviços, inserindo-se na cadeia de valor das instituições. As APIs tornaram-se um grande negócio – 83% das organizações consideram a integração de APIs uma parte crítica de sua estratégia de negócios, de acordo com o relatório "*The State of API Integration* 2020" (ver: https://cdn2.hubspot.net/hubfs/440197/2020-04%20-%20SOAI%20Report.pdf).



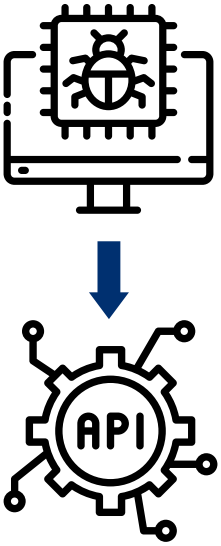
De janeiro de 2019 a janeiro de 2020, ocorreu um aumento de mais de 100% no número de APIs disponíveis, passando de 17,4 milhões para 34,9 milhões. Em janeiro de 2021, o total ultrapassou 46 milhões de coleções (pacotes de APIs) (ver: https://knowledgeburrow.com/how-many-apis-are-there-in-2020/). De acordo com a pesquisa "*Evolution of API Security – A Practical Guide to Addressing* API *Threats in 2023*" da *Wallarm*, aproximadamente 1,7 bilhões de APIs ativas deverão estar disponíveis em 2030 (ver: https://lab.wallarm.com/evolution-of-api-security-in-2023-a-practical-guide/).

As APIs que conectam aplicativos e dados corporativos à *internet* estão sujeitas às mesmas vulnerabilidades que os aplicativos *web* comuns e precisam ser abordadas com pelo menos o mesmo rigor. Na verdade, o acesso externo direto a atualizações de transações e de dados em massa que as APIs permitem sujeitam-nas a ameaças adicionais que os aplicativos web raramente encontram.

Segundo artigo publicado no *site* da *Nordic* APIs, uma pesquisa de 2020 da *SlashData* apontou que:

- 90% dos desenvolvedores estão utilizando APIs, sendo que 70% utilizam APIs de terceiros e 20% utilizam APIs desenvolvidas internamente; e
- 83% de todo o tráfego da *Internet* está relacionado a serviços baseados em API.

(ver: https://nordicapis.com/apis-have-taken-over-software-development/)

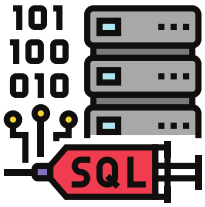


Como resultado desse cenário, a superfície de ataque relacionada às APIs aumentou significativamente. Os ataques que exploram vulnerabilidades em APIs estão mais frequentes, o que gera a expectativa para acreditar que essas vulnerabilidades estarão entre os vetores mais comuns de violações de segurança em futuro próximo. Uma análise da Forbes, partindo de uma previsão do Gartner e usando várias fontes públicas, verificou que API tornou-se um do vetores mais usados em ataques envolvendo dados de aplicativos corporativos, com um aumento significativo do número de explorações (*exploits*) de APIs (ver: https://www.forbes.com/sites/forbestechcouncil/2022/07/25/how-to-address-growing-api-security-vulnerabilities-in-2022/?sh=512bae165a9e), em especial as ameaças do tipo injeção (ver também: https://owasp.org/www-project-api-security/).

Embora existam padrões como o OpenAPI e o AsyncAPI, o uso de APIs ainda é bastante diversificado e cada API tem sua própria peculiaridade. Isso torna difícil para os provedores de serviços de aplicativos financeiros ou CRMs, por exemplo, fazer conexões confiáveis com outros provedores para atender aos requisitos de integração de seus clientes.

Principais Ameaças Cibernéticas

As ameaças que utilizam técnicas de injeção incluem dezenas de variações, desde as conhecidas SQL *injection* e injeções de comandos do sistema operacional até injeções de modelos do lado do servidor ou *Server Side Template Injection* (SSTI), como a vulnerabilidade log4shell, que permitiu aos invasores lançar ataques utilizando a falha do Log4J.



O formato de troca de XML para serviços *web* está sujeito a ataques XML *External Entity Injection* (XXE), que são possíveis quando uma aplicação resolve entidades externas arbitrárias definidas em um documento XML. Infelizmente, os analisadores XML comuns não são adequados para essa finalidade em sua configuração padrão, pois o principal problema, nesse caso, é que a expansão da entidade externa geralmente é habilitada por padrão. Somente com fortalecimento da proteção, desabilitando completamente tanto a possibilidade de expansão de entidades externas como a validação de arquivos *Document Type Declaration* (DTD) externos, os analisadores XML tornam-se seguros.

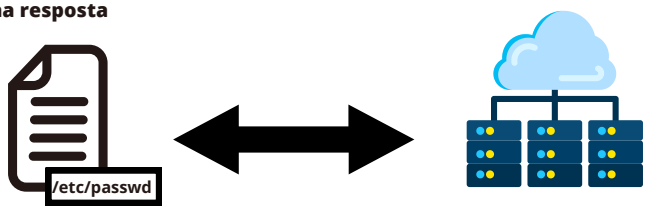
Exemplo de um ataque XXE

1 **O invasor envia um documento XML com uma requisição de envio de dados de arquivos**

```
POST / HTTP1./1
Host:example.com
...

<?xml version="1.0" standalone="no" ?>
<!DOCTYPE replace [
                    <!ENTITY entity SYSTEM
"file:///etc/passwd">
]>
<userinfo>
<firstName>John<firstName>
<lastName>&entity;</lastName>
</userinfo>
```

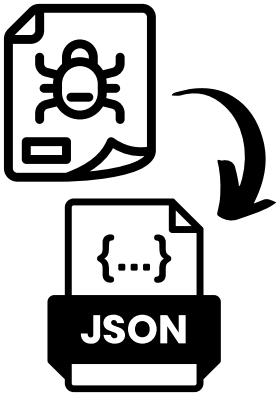
2 **O servidor Web analisa o documento XML, o que o causa a leitura do arquivo /etc/passwd, anexando o resultado na resposta**




```
<userinfo>
<firstName>John<firstName>
<lastName>root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
...
</lastName>
</userInfo>
```

3 **O invasor recebe a resposta do servidor com o conteúdo do arquivo /etc/passwd**

No caso da análise JSON, esta é quase sempre segura quando são utilizadas técnicas mais atuais em vez de JSONP (*JavaScript Object Notation* com *Padding*). A utilização do JSONP pode levar a ataques do tipo *Cross-Site Request Forgery* (CSRF), pois, como o elemento HTML <script> não respeita a política de mesma origem (*Same Origin Policy* – SOP) nas implementações do navegador da *web*, uma página maliciosa pode solicitar e obter dados JSON pertencentes a outro *site*. Isso permitirá que os dados codificados em JSON sejam avaliados no contexto da página maliciosa, possivelmente divulgando senhas ou outros dados confidenciais se o usuário estiver conectado ao outro *site*.



Diferenças entre a segurança cibernética tradicional e a segurança cibernética de API

As principais características da segurança web tradicional são:

ABORDAGEM DE CASTELO E FOSSO

A rede tradicional tem um perímetro claro que controla os pontos de acesso para atribuir permissões aos solicitantes e, em seguida, assumir que aqueles que entraram na rede são benignos.

USO DE PROTOCOLOS ESTÁTICOS

De maneira geral, as solicitações recebidas aderem a protocolos principalmente estáticos, permitindo que os administradores configurem um *firewall* de aplicativo da *Web* (WAF) para impor esses protocolos.

USO DE NAVEGADOR WEB PELOS USUÁRIOS

o WAF pode verificar o ambiente do navegador dos clientes e, se a verificação falhar, assume que o cliente é um *bot* que usa um navegador do tipo *headless browser* ou um emulador.

EXAME DAS SOLICITAÇÕES PARA DETECÇÃO DE ATAQUES

uma rede tradicional pode empregar um WAF para bloquear tentativas de *cross-site scripting* (XSS). Se for identificado um aumento anormal de tráfego de um único IP, o WAF pode determinar que é uma tentativa de ataque de negação de serviço distribuído (DDoS).

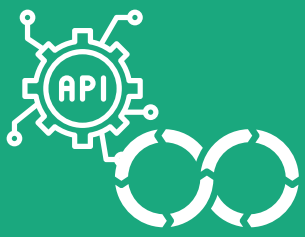
As principais características de segurança de uma API que a distinguem da segurança *web* tradicional são:

UM CASTELO COM MUITAS ABERTURAS E SEM FOSSO

Antigamente, as redes tradicionais precisavam proteger apenas portas comuns como 80 (HTTP) e 443 (HTTPS). Atualmente, os aplicativos *web* têm vários *endpoints* de APIs que usam diferentes protocolos. Como as APIs geralmente se expandem com o tempo, até mesmo uma API pode tornar a segurança um empreendimento difícil.

FORMATOS DE SOLICITAÇÃO DE ENTRADA QUE MUDAM COM FREQUÊNCIA

As APIs evoluem rapidamente em um ambiente *DevOps*, e a maioria dos WAFs não consegue acomodar esse nível de elasticidade. Sempre que uma API é alterada, as ferramentas de segurança tradicionais precisam de ajustes e reconfigurações manuais, um processo sujeito a erros que consome recursos e tempo.



OS USUÁRIOS GERALMENTE NÃO USAM UM NAVEGADOR DA *WEB*

A maioria das APIs de serviço ou microsserviços é acessada por aplicativos nativos e móveis ou outros serviços e componentes de software. Como esses clientes não usam um navegador, as ferramentas de segurança da *Web* não podem usar a verificação do navegador. As soluções que dependem da verificação do navegador para detectar *bots* maliciosos geralmente não são capazes de excluir o tráfego automatizado dos *endpoints* da API.

Os riscos de segurança de APIs mais comuns

O aumento de ameaças de segurança relacionadas a APIs nos últimos anos levou o *Open Web Application Security Project* (OWASP) a lançar o projeto API *Security* Top 10 (https://owasp.org/www-project-api-security/), que ajuda a aumentar a conscientização sobre os problemas mais sérios de segurança de APIs que afetam as organizações.

Dentre os riscos apresentados pelo OWASP, os seguintes riscos devem ser abordados durante o desenvolvimento ou sempre que uma API for atualizada:

AUTORIZAÇÃO QUEBRADA AO NÍVEL DE OBJETO (*BROKEN OBJECT-LEVEL AUTHORIZATION* – BOLA)

Também conhecido como *insecure direct object reference* – IDOR, é uma das vulnerabilidades de APIs mais graves e comuns. Ocorre quando uma solicitação pode acessar ou modificar dados aos quais o solicitante não deveria ter acesso, como acessar a conta de outro usuário adulterando um identificador na solicitação;

AUTORIZAÇÃO QUEBRADA AO NÍVEL DE FUNÇÃO (*BROKEN FUNCTION-LEVEL AUTHORIZATION*):

Ocorre quando o princípio do privilégio mínimo (PoLP) não é implementado, geralmente como resultado de políticas de controle de acesso excessivamente complexas. Isso faz com que um invasor seja capaz de executar comandos privilegiados ou acessar *endpoints* destinados a contas privilegiadas;

AUTENTICAÇÃO QUEBRADA DO USUÁRIO (*BROKEN USER AUTHENTICATION*):

Assim como no BOLA, se o processo de autenticação puder ser comprometido, um invasor pode se passar por outro usuário uma única vez ou até mesmo permanentemente;

EXPOSIÇÃO EXCESSIVA DE DADOS:

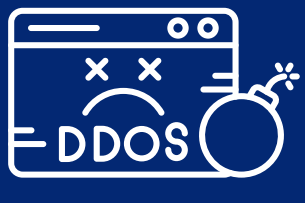
As respostas da API a uma solicitação geralmente retornam mais dados do que são relevantes ou necessários. Mesmo que os dados não sejam exibidos para o usuário, eles podem ser facilmente examinados e podem levar a uma possível exposição de informações sensíveis;

GESTÃO INADEQUADA DE ATIVOS:

O desenvolvimento e a implantação de APIs geralmente são rápidos e a documentação completa geralmente é omitida na liberação de APIs novas ou atualizadas. Isso acaba por criar *endpoints* expostos ou fantasmas, bem como dificulta a compreensão de como as APIs mais antigas funcionam e precisam ser implementadas;

FALTA DE RECURSOS E LIMITAÇÃO DE TAXA

Os *endpoints* da API geralmente são abertos à *internet* e, se não houver restrições quanto ao número ou tamanho das solicitações, estarão abertos a ataques do tipo DoS ou DDoS e a ataques de força bruta;



FALHAS DE INJEÇÃO:

Se os dados solicitados não forem analisados e validados corretamente, um invasor pode lançar um comando ou ataque de injeção para acessá-los ou executar comandos mal-intencionados sem autorização; e

ATRIBUIÇÃO EM MASSA:

As estruturas de desenvolvimento de *software* muitas vezes comprovam a funcionalidade de inserir todos os dados recebidos de um formulário *on-line* em um banco de dados com apenas uma linha de código, conhecida como atribuição em massa, removendo, portanto a necessidade de escrever linhas repetitivas de código de mapeamento de formulário. Se isso for feito sem especificar quais dados serão aceitáveis, abre-se uma variedade de vetores de ataque.

Métodos de Teste da Segurança de APIs

Podem-se utilizar os seguintes métodos para testar manualmente suas APIs em busca de vulnerabilidades de segurança:

TESTE DE ADULTERAÇÃO DE PARÂMETROS

Na maioria dos casos, os parâmetros enviados por meio de solicitações de API podem ser facilmente adulterados. Por exemplo, ao manipular parâmetros, os invasores podem alterar o valor de uma compra e receber produtos gratuitamente ou levar uma API a fornecer dados confidenciais não autorizados para a conta do usuário. A adulteração de parâmetros geralmente é realizada usando campos de formulário ocultos; pode-se testar a presença de campos ocultos usando o inspetor de elementos do navegador. Se for encontrado esse campo, deve-se experimentar diferentes valores a fim de verificar como a API reage.

TESTE DE INJEÇÃO DE COMANDO

Para testar se uma API é vulnerável a ataques de injeção de comando, deve-se tentar injetar comandos do sistema operacional instalado no servidor responsável pela execução da API por meio da API em questão. É recomendável usar comandos de menor impacto no sistema operacional os quais possam ser facilmente observados, como, por exemplo, um comando de reinicialização ou de criação de diretórios.

TESTE *FUZZING* DE ENTRADA DE API

A técnica *fuzzing* ajuda a detectar *bugs* de *software*, como vazamentos de memória, corrupções de banco de dados, brechas de segurança, resultados de pesquisa ruins, erros de codificação, falhas do sistema, DoS (*Denial of Service*), XSS (*Cross-site Scripting*), entre outros. No caso de APIs, a maioria dos testes *fuzzing* implica em fornecer dados aleatórios para a API até que se descubra um problema funcional ou de segurança. A intenção, portanto, é encontrar evidências de que a API retornou um erro, processou entradas incorretamente ou travou.

TESTE PARA MÉTODOS HTTP NÃO TRATADOS

Os aplicativos *web* que se comunicam usando API podem usar vários métodos HTTP. Esses métodos HTTP são usados para armazenar, excluir ou recuperar dados. Se o servidor não suportar o método HTTP, o usuário geralmente receberá um erro. No entanto, nem sempre será esse o caso. Se o método HTTP não for suportado no lado do servidor, isso criará uma vulnerabilidade de segurança. Podem-se testar métodos HTTP não suportados fazendo uma solicitação HEAD para um terminal de API que requer autenticação, em especial os métodos HTTP comuns - *post, get, put, patch, delete*, etc.

Além de testes manuais, proteger API de produção, especialmente aquelas que têm um processo regular de desenvolvimento e de lançamento, requer o uso de ferramentas automatizadas. Existem no mercado diversas ferramentas de código aberto que podem ajudar a projetar cenários de teste relacionados à segurança, executá-los em terminais de APIs e corrigir problemas que forem descobertos. Essas ferramentas também podem descobrir vulnerabilidades de lógica de negócios.

Melhores Práticas para Segurança de APIs

As seguintes práticas podem auxiliar no processo melhoria da segurança das APIs de uma organização.

1 Manter o controle das APIs em um registro

Ninguém pode garantir o que não conhece, logo é essencial manter o controle de todas as APIs em um registro. Esse registro de APIs deve armazenar as características como nome, finalidade, carga útil, uso, acesso, data de ativação, data de desativação e proprietário. Isso evitará a existência de APIs de sombra ou silos que foram esquecidos, nunca documentados ou desenvolvidos fora de um projeto principal.

O registro dos detalhes das APIs auxilia:

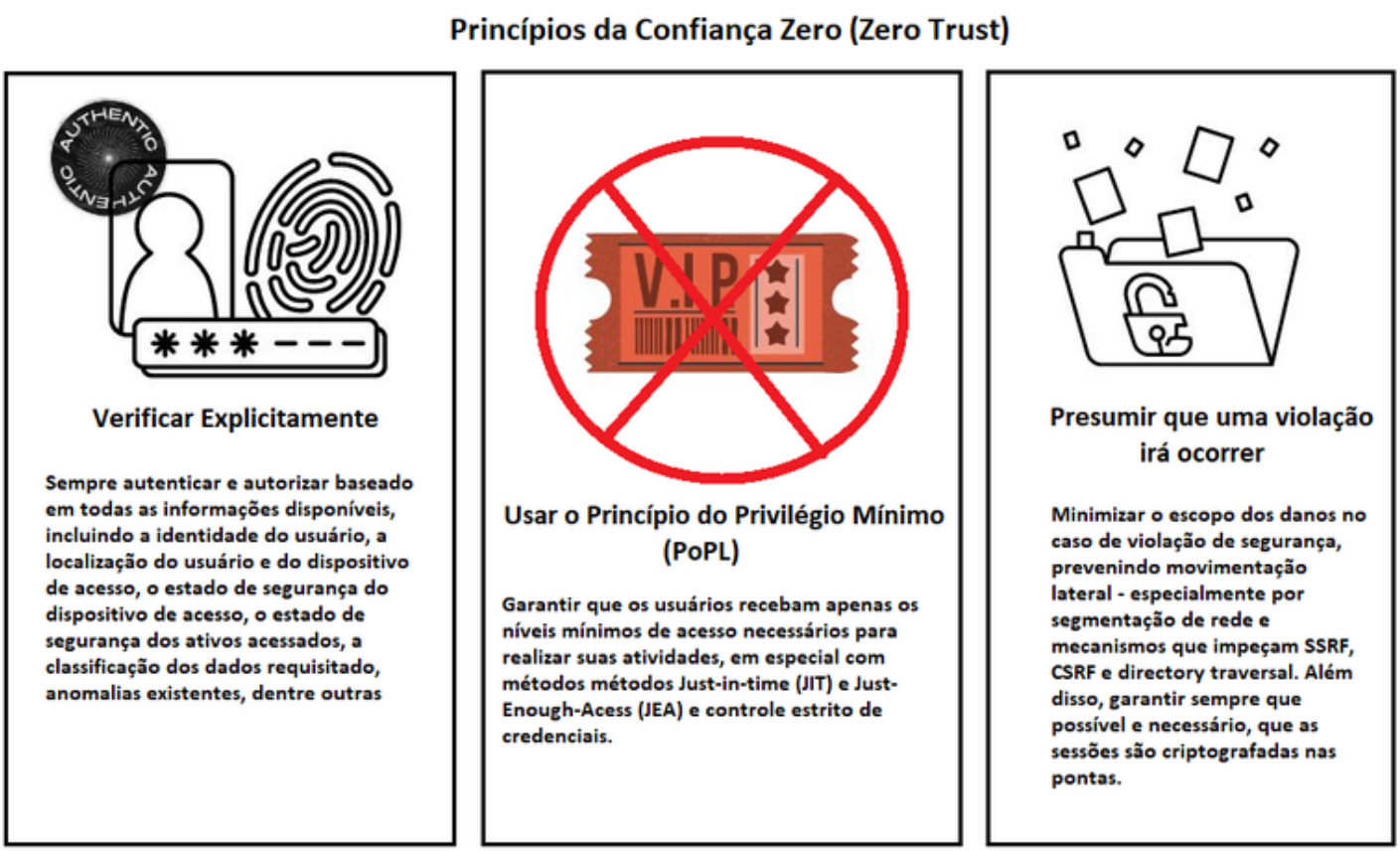
- no atendimento dos requisitos de conformidade e auditoria;
- na análise de riscos das APIs; e
- na análise forense, no caso de um incidente de segurança.

Uma boa documentação é particularmente importante para desenvolvedores terceirizados que desejam incorporar essas APIs em seus próprios projetos. O registro da API deve incluir *links* para a documentação, a qual contém todos os requisitos técnicos da API, incluindo funções, classes, tipos de retorno, argumentos e processos de integração.

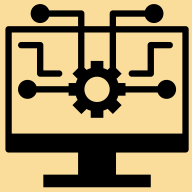
2 Adotar uma filosofia de confiança zero (*Zero-Trust Model* – ZTM)

No modelo de segurança de perímetro, o que está "dentro" é confiável e o que está "fora" não é confiável. As redes não são mais tão simples, pois as ameaças internas tornaram-se predominantes e os usuários legítimos passaram a se conectar de fora do perímetro. Isso é especialmente verdadeiro para API públicas, com usuários de todo o mundo acessando componentes internos de *software* e dados sensíveis.

Uma filosofia de confiança zero (ZTM) muda o foco da segurança tradicional para os usuários, os ativos e recursos específicos. O ZTM ajuda a garantir que as APIs sempre autentiquem usuários e aplicativos (dentro ou fora do perímetro), que sejam fornecidos os privilégios mínimos de que as APIs realmente precisem para desempenhar suas funções e que permitam o devido monitoramento de comportamento anômalo.



3 Identificar os riscos das APIs



A única maneira de proteger efetivamente uma API é entender quais partes de seu ciclo de vida são inseguras. Isso pode ser complexo, especialmente se uma organização opera um grande número de APIs. É importante considerar todo o ciclo de vida. Toda API deve ser tratada como um artefato de *software*, que passa por todos os estágios de um produto de *software*, desde o planejamento até o desenvolvimento, teste, preparação e produção.

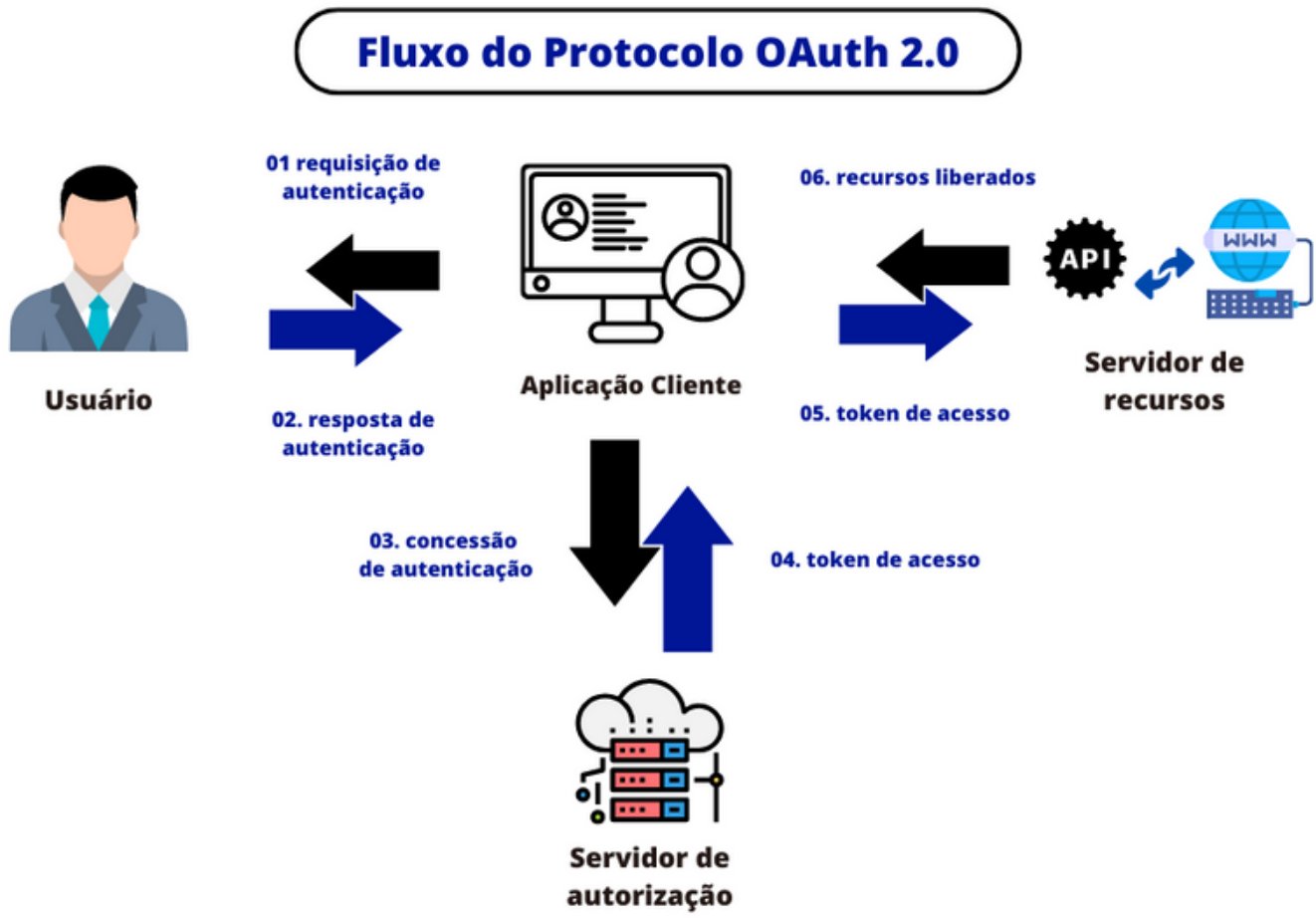
Logo, uma das mais importantes práticas de segurança é realizar uma avaliação de risco para todas as APIs da organização (existentes ou em uso). Assim sendo, devem ser estabelecidas medidas para garantir que as APIs atendam às políticas de segurança e não sejam vulneráveis a riscos conhecidos. A lista de vulnerabilidades "Top 10" do OWASP (https://owasp.org/www-project-api-security/) é um bom recurso para manter o controle sobre ataques existentes e *software* mal-intencionado.

4 Controlar o acesso aos recursos da API

Para controlar o acesso aos recursos da API, deve-se identificar todos os usuários e dispositivos relacionados. Isso normalmente requer que os aplicativos do lado do cliente incluam um *token* na chamada de API, para que o serviço possa validar o cliente.
Uma forma de controlar o acesso a API é a utilização de padrões, como *OAuth* 2.0, *OpenID Connect* ou *tokens* da *Web* JSON para:

- autenticar o tráfego da API; e
- definir regras de controle de acesso que determinam quais usuários, grupos e funções podem acessar recursos específicos da API.

Além disso, sempre deve ser utilizado o princípio do privilégio mínimo (PoPL), de forma a impor mecanismos que apenas as permissões necessárias sejam atribuídas aos usuários.
Todas as APIs devem ser mantidas, sempre que possível, atrás de um *firewall*, que pode ser um *firewall* de aplicativo da *Web* (WAF) ou *gateway* de API, acessado por meio de um protocolo seguro, como HTTPS, para fornecer proteção de linha de base, como a verificação de ameaças baseadas em assinatura e os ataques baseados em injeção.



5 Implementar limitação de taxa e verificação de geovelocidade nas APIs

Para manter a disponibilidade, os serviços compartilhados precisam se proteger contra o uso excessivo, pois mesmo os sistemas altamente escalonáveis devem ter limites de consumo em algum nível.

Por esse motivo, devem ser aplicadas uma limitação de taxa e verificações de velocidade geográfica nas APIs, como medidas de defesa. Isso auxilia na redução da superfície de ataque, evitando que as APIs sejam sobrecarregadas com requisições, reduzindo, dessa forma, a possibilidade de ataques de negação de serviço (DoS).

As verificações de velocidade geográfica fornecem autenticação baseada em contexto, analisando a taxa de acesso com base na velocidade de tráfego exigida entre as tentativas de *login* anteriores e as atuais.

Todas essas verificações devem ser aplicadas pelo código de middleware que faz parte do aplicativo da API. O *middleware* lida com as solicitações antes de encaminhá-las para atendimento.

6 Criptografar todas as requisições e respostas via API

Todos os dados gerenciados por uma API, especialmente informações de identificação pessoal (*Personal Identification Information* - PII) ou outros dados confidenciais protegidos por normas e regulamentos de conformidade, devem ser criptografados.

Deve-se implementar criptografia para os dados em repouso, a fim de garantir que invasores que comprometam o servidor de APIs não possam utilizá-los.

Todo tráfego de rede deve ser criptografado usando o *Transport Layer Security* (TLS), principalmente em solicitações e respostas de API, pois provavelmente conterão credenciais e dados confidenciais. Também deve ser exigido assinaturas para garantir que apenas usuários autorizados possam descriptografar e modificar os dados fornecidos por sua API.

Por fim, habilitar o HTTP *Strict Transport Security* (HSTS) sempre que possível é melhor do que redirecionar o tráfego HTTP para HTTPS, pois os clientes da API podem não se comportar conforme o esperado.

7 Validar todos os dados obtidos via API

Nunca se deve assumir que os dados obtidos via API foram limpos ou validados corretamente. Toda organização deve implementar suas próprias rotinas de limpeza e de validação de dados no lado do servidor para evitar falhas de injeção padrão e ataques de falsificação de solicitação entre *sites*. O uso de ferramentas de depuração ajuda a examinar o fluxo de dados da API e a rastrear erros e anomalias.

8 Compartilhar apenas a informação necessária numa API

As respostas de uma API geralmente incluem um registro de dados inteiro, em vez de apenas os campos relevantes, contando com o aplicativo cliente para filtrar o que um usuário tem acesso. Isso não é uma boa prática de programação, pois apesar de diminuir os tempos de resposta também fornece aos invasores informações adicionais sobre a API e os recursos que ela acessa.

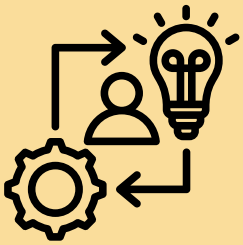
Ao desenvolver uma API, deve-se garantir que as respostas contenham apenas as informações mínimas necessárias para atender uma solicitação.

9 Decidir sobre o uso de SOAP ou REST

Como visto anteriormente, as opções dominantes para acessar serviços web são o SOAP, que é um protocolo de comunicação, e a REST API (ou RESTful API), que é um conjunto de princípios arquitetônicos para transmissão de dados.

SOAP e REST usam formatos e semânticas diferentes, bem como exigem estratégias diferentes para garantir uma segurança robusta.

A segurança SOAP é aplicada no nível da mensagem, usando assinaturas digitais e partes criptografadas dentro da própria mensagem XML. A REST depende muito das regras de controle de acesso associadas ao identificador de recurso universal da API, como *tags* HTTP e o caminho da URL.

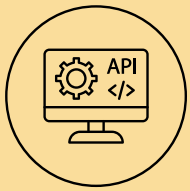


É recomendável utilizar o SOAP se as principais preocupações forem padronização e segurança. Embora ambas as opções suportem *Secure Sockets Layer/Transport Layer Security* (SSL/TLS), o SOAP também suporta *Web Services Security*, verificação de identidade por meio de intermediários, em vez de apenas verificação ponto a ponto fornecida por SSL/TLS, e tratamento de erros integrado. No entanto, o SOAP expõe os componentes da lógica do aplicativo como serviços, em vez de dados, o que pode torna-lo complexo de implementar e pode exigir que vários aplicativos sejam refeitos.

A REST, por sua vez, é compatível com vários tipos de saída de dados – incluindo JSON, valores separados por vírgula (CSV) e HTTP –, enquanto o SOAP só pode lidar com XML e HTTP. Além disso, a REST apenas acessa dados, portanto é uma maneira mais simples de acessar serviços da *web*. Por esses motivos, as organizações geralmente preferem REST para projetos de desenvolvimento *web*. No entanto, a segurança deve ser incorporada para trocas de dados, para implantação da API no ambiente e para interação com os clientes.

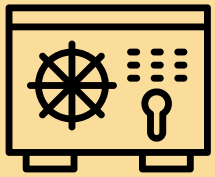
10 Armazenar as chaves de API

As chaves de API identificam e verificam o acesso do aplicativo ou do serviço que chama uma API. Elas também podem bloquear ou limitar as chamadas feitas para uma API e identificar padrões de uso.



As chaves de API são menos seguras do que os *tokens* de autenticação e requerem um gerenciamento cuidadoso. Não se deve incorporar essas chaves diretamente em seu código ou em arquivos na árvore de origem do aplicativo, onde podem ser expostas acidentalmente.

Como melhores práticas, as chaves de API podem ser armazenadas em variáveis de ambiente ou em arquivos fora da árvore de origem do aplicativo, ou, ainda, ser utilizado um serviço de gerenciamento de chaves, como o cofre de chaves, que proteja e gerencie todas as chaves de API de um aplicativo.



Mesmo com essas medidas em vigor, devem ser excluídas todas chaves desnecessárias, a fim de minimizar a superfície de exposição a ataques, além de realizar a troca periódica das chaves, especialmente se houver a suspeita de que ocorreu uma violação.

10 Considerações finais

À medida que as organizações continuam a dividir aplicativos monolíticos em microsserviços e avançam no caminho nativo da nuvem, o uso de APIs continuará a crescer e suas vulnerabilidades serão mais exploradas.

A aplicação de melhores práticas de segurança auxilia na segurança no uso de APIs, além de ajudar a fortalecer a infraestrutura de aplicativos.

Além disso, é imperativo que os responsáveis por aplicações *web* no âmbito da administração pública federal tomem conhecimento dos seguintes guias de autoria da Secretaria de Governo Digital - SGD:

- Guia de Requisitos Mínimos de Segurança e Privacidade para APIs (https://www.gov.br/governodigital/pt-br/seguranca-e-protecao-de-dados/guias/guia_requisitos_minimos_apis.pdf)
- Guia de Segurança de Aplicações *Web* (https://www.gov.br/governodigital/pt-br/seguranca-e-protecao-de-dados/guias/guia_requisitos_minimos_web.pdf)

Vale ressaltar que o Departamento de Segurança da Informação e Cibernética (DSIC) também recomenda aos usuários das diversas organizações, além das recomendações apresentadas nesta OSIC, que:

- promovam, divulguem e incentivem o uso do múltiplo fator de autenticação (MFA); e
- informem imediatamente à Equipe de Prevenção, Tratamento e Resposta a Incidentes Cibernéticos (ETIR) e sua instituição a ocorrência de um incidente cibernético.

Outras Orientações de Segurança da Informação e Cibernética (OSICs) estão disponíveis em https://www.gov.br/gsi/pt-br/composicao/SSIC/dsic/osic e propostas de temas, sugestões ou outras contribuições para serem abordadas em futuras OSICs podem ser encaminhadas ao *e-mail* educa.si@presidencia.gov.br.