OWASP API Security Top 10 2019

Os Dez Mais Críticos Riscos de Segurança de API

TOC

Tabela de conteúdo

Table of Contents

Sobre o OWASP

OWASP é o acrônimo em inglês para "Open Web Application Security Project", é uma comunidade aberta, dedicada a habilitar as organizações a desenvolver, comprar e manter aplicações e APIs que podem ser confiáveis.

No OWASP, você irá encontrar de forma gratuita e aberta:

Ferramentas e padrões de segurança de aplicativos.

Livros completos sobre testes de segurança de aplicativos, desenvolvimento seguro de código e revisão de código seguro.

- Apresentações e vídeos.
- Cheat sheets em diversos tópicos.
- Padrões de controles de segurança e bibliotecas.
- Capítulos locais em todo o mundo.
- Pesquisas inovadoras.
- Extensivas conferências ao redor do mundo.
- Listas de e-mail.

Saiba mais em: https://www.owasp.org.

No OWASP, todas as ferramentas, documentações, vídeos, apresentações e capítulos são gratuitos e abertos para qualquer um interessado em melhorar a segurança de aplicações

Nós advogamos a abordagem da segurança das aplicações como um problema de pessoas, processos e tecnologia, uma vez que as abordagens mais efetivas na segurança de aplicações necessitam de melhorias nestas áreas.

O OWASP é um novo tipo de organização. Nossa independência de pressão comercial permite-nos prover informações e práticas livres de vieses e de efetivo custo benefício sobre segurança de aplicações.

O OWASP não é afiliado a qualquer empresa de tecnologia, embora nosso suporte ao uso de tecnologias comercias. O OWASP produz diversos materiais de forma colaborativa, transparente e aberta.

A Fundação OWASP é uma entidade não comercial, que encoraja projetos de sucesso há muito tempo. Praticamente todos os envolvidos com o OWASP são voluntários, incluindo os que integram a diretoria, líderes de capítulos, líderes de projetos e membros de cada projeto. Nós apoiamos pesquisas inovadoras de segurança com doações e infraestrutura.

Venha conosco!

FW

Prefácio

Elemento fundamental na inovação nas soluções app-driven nos dias de hoje são as APIs (Application Programming Interface). Desde os bancos, lojas, transportes, IoT, veículos autônomos e cidades inteligentes, as APIs são parte crítica de soluções modernas de móvel, SaaS, aplicações web em geral, e podem ser encontradas em interfaces com o cliente, parceiros e aplicações internas.

Por natureza, as APIs expõem a lógica dos aplicativos e dados sensíveis, inclusive, dados pessoais sensíveis, e por esta razão, as APIs vem se tornando cada vez mais alvo de atacantes. Sem APIs seguras, inovações podem se tornar impossíveis.

Embora uma ampla avaliação de Top 10 a respeito da segurança de aplicações faça sentido, em razão de suas particularidades, uma lista de riscos de segurança específica para APIs também é um requisito. A segurança de APIs tem foco em estratégias e soluções para a compreensão e mitigação de vulnerabilidades únicas associadas às APIs.

Se você é familiarizado com o projeto OWASP Top 10, irá perceber similaridades entre os dois documentos: Isto é intencional para facilitar a compreensão e adoção. Se você é um novato nas śeries Top 10 do OWASP, talvez seja melhor você ler atentamente as seções Riscos de Segurança de API e Metodologia e Dados antes.

Você pode contribuir com o projeto OWASP API Security Top 10 com questionamentos, comentários e ideias por meio do repositório do projeto no GitHub:

- https://github.com/OWASP/API-Security/issues
- https://github.com/OWASP/API-Security/blob/master/CONTRIBUTING.md

Você encontra o OWASP API Security Top 10 aqui:

- https://owasp.org/www-project-api-security/
- https://github.com/OWASP/API-Security

Nós gostaríamos de agradecer a todos colaboradores que fizeram este projeto possível com seus esforços e contribuições. Todos que participaram estão listados na seção Agradecimentos. Obrigado!

Introdução

Seja bem-vindo ao OWASP API Security Top 10 - 2019!

Seja bem-vindo à primeira edição do projeto OWASP API Security Top 10. Se você está familiarizado com a série Top 10 do OWASP, irá perceber certas similaridades: elas são intencionais para melhor leitura a adoção. Se não é o seu caso, considere visitar a wiki do projeto antes de mergulhar mais profundamente nos mais críticos riscos de segurança de APIs.

As APIs possuem papel fundamental na arquitetura de aplicações modernas. Inovação e consciência de segurança possuem ritmos diferentes. É importante ter foco nas fraquezas mais comuns na segurança de APIs.

O primeiro objetivo do projeto OWASP API Security Top 10 é educar aqueles que estejam envolvidos no desenvolvimento e manutenção de APIs. Podem ser, por exemplo: desenvolvedores, designers, arquitetos, gerentes e organizações.

Na seção Metodologia e Dados você pode ler mais a respeito de como esta primeira versão foi criada. Em futuras versões, desejamos envolver a indústria da segurança com uma chamada pública para contribuição de dados. Por ora, nós encorajamos todos que possam contribuir com questionamentos, comentários e ideias em nosso repositório no GitHub ou nossa lista de e-mails.

RN Notas da Versão

Esta é a primeira edição do projeto OWASP API Security Top 10, que desejamos que seja atualizado periodicamente a cada três ou quatro anos.

Ao contrário dessa versão, para o futuro, desejamos fazer uma chamada pública por dados e envolver a indústria de segurança neste esforço. Na seção Metodologia e Dados você irá encontrar mais detalhes a respeito de como esta versão foi construída. Para mais detalhes a respeito de riscos de segurança, consulte a seção Riscos de Segurança de API.

É importante realizar que durante os últimos anos a arquitetura das aplicações foram significativamente modificadas. Atualmente, as APIs representam um papel muito importante nessa nova arquitetura de microserviços, single page applications, aplicativos móveis, IoT e etc.

O projeto OWASP Security Top 10 foi um esforço necessário para criar uma consciência a respeito de problemas de segurança de APIs. Este projeto só foi possível com o esforço de diversos voluntários, todos listados na seção Agradecimentos. Obrigado!

RISK

API Security Risk

A Metodologia de Avaliação de Risco do OWASP foi adotada para a análise dos riscos de API.

A tabela abaixo resume a terminologia associada à pontuação de risco.

Agentes de Ameaça	Explorabilidade	Prevalência da Fraqueza	Detecção da Fraqueza	Impacto Técnico	Impacto ao Negócio
Específico da API	Fácil: **3**	Difundida **3**	Fácil **3**	Severo **3**	Específico do negócio
	Médio: **2**	Comum **2**	Média **2**	Moderado **2**	
	Difícil: **1**	Difícil **1**	Difícil **1**	Menor **1**	

Nota: Esta abordagem não leva em consideração um agente de ameaça interno. Também não considera detalhes técnicos associados à sua aplicação em específico. Estes são fatores que podem afetar de maneira significativa a probabilidade de um atacante encontrar e explorar vulnerabilidades específicas. Esta classificação também não avalia impactos ao seu negócio. Sua organização terá que decidir quanto risco de segurança de aplicativos e APIs que a organização está disposta a aceitar, dada sua cultura, indústria e ambiente regulatório. O propósito do OWASP API Security top 10 não é desenvolver uma análise de risco por você.

Referências

OWASP

- OWASP Risk Rating Methodology
- Article on Threat/Risk Modeling

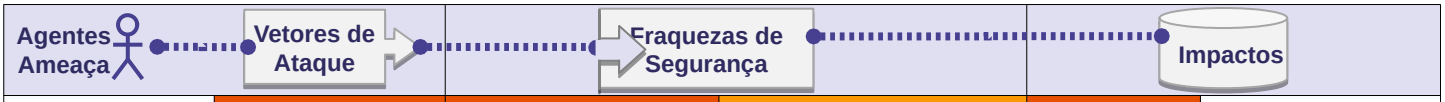
Externas

- ISO 31000: Risk Management Std
- ISO 27001: ISMS
- NIST Cyber Framework (US)
- ASD Strategic Mitigations (AU)
- NIST CVSS 3.0
- Microsoft Threat Modeling Tool

T10 OWASP API Security Top 10 - 2019

API1:2019 - Broken Object Level Authorization	APIs tendem a expor *endpoints* para manipulação de objetos internos criando uma ampla camada de ataque ao controle de nível de acesso. O controle de autorização deve ser verificado em toda função que tenha acesso à fontes de dados que utilizem dados enviados pelo usuário.
API2:2019 - Broken User Authentication	Mecanismos de autorização não raramente são implementados incorretamente, permitindo que atacantes comprometam tokens de autenticação ou explorem falhas na implementação para assumir identidades temporária ou permanentemente. Isto compromete a habilidade dos sistemas em identificar o usuário/cliente comprometendo a segurança da API.
API3:2019 - Excessive Data Exposure	Implementações rápidas e de maneira genérica podem fazer que desenvolvedores exponham todas as propriedades dos objetos sem considerar que determinadas informações podem ser sensíveis ao indivíduo e deixando ao cliente as atividades de filtrar as informações antes de exibi-las ao usuário.
API4:2019 - Lack of Resources & Rate Limiting	Não raramente APIs não impõem restrições ao tamanho ou à quantidade de recursos que podem ser requisitados pelo cliente/usuário. O impacto vai além do desempenho de servidores de API, resultado em *Denial of Service* (DoS), mas também, podem deixar portar abertas para falhas de autenticação por força bruta.
API5:2019 - Broken Function Level Authorization	Políticas complexas no controle de acesso, com diferentes hierarquias, grupos, papéis e uma falta de clareza na separação entre papéis comuns e de administração podem levar a falhas de autenticação. Explorando esta condição, atacantes podem ganhar acesso à recursos de terceiros ou à funções administrativas.
API6:2019 - Mass Assignment	Conectar dados entregues pelo cliente (ex. JSON) diretamente em modelos de dados sem filtrar apropriadamente propriedades em *whitelist* podem eventualmente levar a atribuição em massa. Mesmo buscando adivinhar propriedades, explorando outros *endpoints*, consultando documentação e enviando propriedades adicionais em *payload* os atacantes podem modificar propriedades de objetos os quais não deveria.
API7:2019 - Security Misconfiguration	Configurações de segurança incorretas, de maneira geral, é resultado de configurações padrão ou configurações incompletas. Armazenamento em nuvem abertas, configurações incorretas em cabeçalhos HTTP, métodos HTTP desnecessários, CORS permissivos e divulgação de mensagens de erro com informações sensíveis.
API8:2019 - Injection	Injeção de falhas como SQL, NoSQL, injeção de comandos e etc, ocorrem quando informações não confiáveis são enviadas ao interpretador como parte de um comando ou consulta. A informação maliciosa do atacante pode enganar o interpretador a executar comandos não esperados ou acessar dados sem a autorização adequada.
API9:2019 - Improper Assets Management	APIs tendem a expor mais *endpoints* que uma aplicação web tradicional, de forma que documentação apropriada e atualizada é muito importante. Gestão de inventário de versões de API tem papel importante para mitigar problemas como versões antigas de API e *endpoints* de debug.
API10:2019 - Insufficient Logging & Monitoring	Falta de log e monitoramento, acoplado com falta ou ineficiente integração com respostas a incidentes, permite que atacantes consigam ter persistência e descobrir outros sistemas para explorar, extrair ou destruir informação. Muitos estudos demonstram que a descoberta de uma brecha leva mais de 200 dias para ser identificada, além de muitas vezes ser detectada mais por terceiros que por um processo interno.

API1:2019 Broken Object Level Authorization



Específico da API	Explorabilidade: 3	Prevalência: 3	Detecção: 2	Técnico: 3	Específico do negócio
Atacantes podem explorar *endpoints* de API que estejam vulneráveis a quebrar níveis de autorização com a manipulação de ID de objeto enviado na requisição. Isto pode levar a acesso à informações não autorizadas a dados sensíveis. Este problema é extremamente comum em aplicações baseadas em APIs uma vez que os componentes de servidores usualmente não verificam o estado do cliente, ao invés disso, confia em parâmetros como IDs de objeto que são enviados pelo cliente que decide qual objeto acessar.		Este é o mais comum e impactante ataque contra APIs. Autorização e mecanismos de controle de acesso em aplicações modernas são complexos e amplos. Mesmo se a aplicação implementa uma apropriada infraestrutura de verificação de autorização, desenvolvedores podem esquecer de usar tais verificações antes de acessar objetos com informações sensíveis. Controle de acesso não é tipicamente favorável a testes automatizados, estáticos ou dinâmicos.		Acesso não autorizado pode resultar no vazamento de dados à partes não autorizadas, perda ou manipulação de dados. Acesso não autorizado à objetos também podem levar a tomada de contas.	

A API está vulnerável?

O nível de autorização de objeto é um mecanismo de controle de acesso que usualmente é implementado ao nível de código para validar que um usuário pode apenas acessar objetos aos quais realmente tem permissão.

Todo *endpoint* de API que recebe um ID de objeto, e executa qualquer tipo de ação sobre este objeto, deve implementar verificações de autorização de acesso ao nível deste objeto. A verificação deve validar que o usuário tem acesso para executar aquela ação no objeto requisitado.

Falhas nesse mecanismo geralmente levam ao acesso não autorizado de informações, vazamento de dados, modificação ou destruição de dados.

Cenários de exemplo de ataques

Cenário #1

Uma plataforma de e-commerce para lojas de compras online entrega uma listagem com os gráficos de receita de suas lojas hospedadas. Inspecionando as requisições do navegador, o atacante pode identificar que os endpoints utilizados como fonte de dados para os gráficos utiliza um padrão como /shops/{shopName}/revenue_data.json. Utilizando outro *endpoint* da API, o atacante consegue uma lista de todos os nomes das lojas hospedadas na plataforma. Com um simples *script* o atacante pode agora, manipulando o nome substituindo o parâmetro {shopName} na URL, ganhar acesso aos dados das vendas de milhares de lojas que utilizam a plataforma de *e-commerce*.

API1:2019 Broken Object Level Authorization

Cenário #2

Enquanto monitora o tráfego de rede um *wearable device*, o atacante tem sua atenção despertada ao perceber o verbo HTTP PATCH possui o cabeçalho customizado X-User-Id: 54796. Substituindo o valor do cabeçalho o atacante recebe uma resposta HTTP válida, sendo possível portanto modificar os dados de outros usuários.

Como prevenir

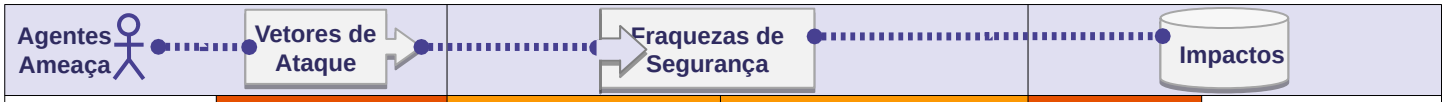
- Implementar mecanismo apropriado de autorização de acesso baseado em políticas e hierarquias.
- Utilizar uma autorização para verificar se o usuário pode acessar e executar ações nos registros em todas as funções que utiliza *input* do usuário para acessar dados.
- Prefira utilizar valores randômicos como GUIs para ids de registros.
- Escreva testes para avaliar seu mecanismo de autorização, não autorize *deployment* de mudanças de código que quebrem estes testes.

Referências

Externas

- CWE-284: Improper Access Control
- CWE-285: Improper Authorization
- CWE-639: Authorization Bypass Through User-Controlled Key

API2:2019 Broken User Authentication



Específico da API	Explorabilidade: 3	Prevalência: 2	Detecção: 2	Técnico: 3	Específico do negócio
Autenticação em APIs é um mecanismo complexo e confuso. Engenheiros e arquitetos de software e segurança podem ter conceitos equivocados a respeito de quais são os limites da autenticação e na forma como devem ser implementados corretamente. Adicionalmente, mecanismos de autenticação são alvo de atacantes pois estão expostos a todos. Estes dois pontos fazem da autenticação um componente vulnerável a muitos tipos de *exploits*.		Existem dois tipos de sub-problemas: 1. Falta de mecanismo de proteção: Os *endpoints* da API que geralmente são responsáveis pela autenticação devem ser tratados diferentemente dos *endpoints* regulares e implementam camadas extras de proteção. 2. Má implementação do mecanismo: O mecanismo é usado ou implementado sem considerar vetores de ataque ou com casos de uso inadequados (ex.: um mecanismo de autenticação desenvolvido para dispositivos IoT pode não ser a melhor opção para aplicativos web).		Atacantes podem tomar controle de outras contas de usuários no sistema, acessar seus dados pessoais e executar ações sensíveis em seu nome, como transações financeiras e enviar mensagens pessoais.	

A API está vulnerável?

Os *endpoints* e fluxos de autorização são ativos que devem ser protegidos. "Esqueci minha senha/Redefinição de senha" devem ser tratados da mesma forma que outros de autenticação.

Uma API está vulnerável se:

- Permite a prática de *credential stuffing* o qual o atacante tem uma lista de nomes de usuário e senhas.
- Permite que atacantes executem força bruta contra uma mesma conta de usuário sem exibir CAPTCHA ou mecanismo de bloqueio da conta.
- Permite o uso de senhas fracas.
- Envia detalhes sensíveis da autenticação como tokens e senhas na URL.
- Não executa a validação de autenticidade de tokens.
- Aceita *tokens* JWT não assinados/fracos ("alg":"none")/não valida data de expiração.
- Utiliza senhas em texto plano, não criptografadas ou com hash fraco de criptografia.
- Usa chaves de criptografias fracas.

Cenários de exemplo de ataques

Cenário #1

Credential stuffing (utilizando listas de usuário e senha conhecidas), é um ataque comum. Se uma aplicação não implementar em sua arquitetura proteções automatizadas ou proteções contra *credential stuffing*, a aplicação pode ser utilizada como base de teste para determinar se credenciais são válidas.

API2:2019 Broken User Authentication

Cenário #2

Um atacante inicia um processo de recuperação de senha enviando uma requisição POST para o *endpoint* /api/system/verification-codes e enviando um usuário no corpo da requisição. Um *token* SMS com 6 dígitos é enviado para o telefone móvel da vítima. Uma vez que a API não implementa um mecanismo de limite, o atacante pode testar todas as combinações possíveis utilizando um *script multi-thread* contra o *endpoint* /api/system/verification-codes/{smsToken} e assim descobrir o *token* correto em alguns minutos.

Como prevenir

- Certifique-se que conhece todos os fluxos possíveis para autenticar-se na API.
- Pergunte aos engenheiros/arquitetos quais fluxos você esqueceu.
- Leia sobre seus mecanismos de autenticação. Tenha certeza que você compreende quando e como foram utilizados. OAuth não é mecanismo de autenticação, e não é chave de API.
- Não reinvente a roda em autenticação, geração de *token*, armazenamento de senhas. Utilize o que é padrão.
- *Endpoints* para recuperação de senhas devem ser tratados assim como aqueles voltados para os processos de login para questões como ataques de força bruta, limitação de confiança e bloqueio de contas.
- Use as informações do projeto OWASP Authentication Cheatsheet.
- Sempre que possível, implemente autenticação multi-fator.
- Implemente mecanismos anti força bruta para mitigar *credential stuffing*, ataque por dicionário em seus *endpoints* de autenticação. A taxa de confiança da proteção desse mecanismo deve ser mais restrito que os demais mecanismos na sua API.
- Implemente bloqueio de conta / mecanismos de CAPTCHA a fim de prevenir o uso de força bruta contra usuários específicos. Implemente verificação de senhas fracas.
- Chaves de API não devem ser utilizadas para autenticação de usuários, mas para aplicativos clientes e autenticação de projetos.

Referências

OWASP

- OWASP Key Management Cheat Sheet
- OWASP Authentication Cheatsheet
- Credential Stuffing

Externas

- CWE-798: Use of Hard-coded Credentials

API3:2019 Excessive Data Exposure

Específico da API	Explorabilidade: 3	Prevalência: 2	Detecção: 2	Técnico: 2	Específico do negócio
A exploração por excesso de exposição de dados é simples, e usualmente realizada com o monitoramento do tráfego investigando as respostas da API, buscando dados sensíveis que não deveriam ser entregues ao usuário.		APIs confiam em clientes para ações de filtro de informação. Desde que APIs são utilizadas como fonte de informações, algumas vezes os desenvolvedores as implementam de maneira genérica sem considerar o quão sensíveis são os dados que elas expõem. Ferramentas de análise automatizada geralmente não podem detectar este tipo de vulnerabilidade em razão de ser difícil da legitimidade dos dados retornados pela API, e dados considerados sensíveis não devem ser retornado sem uma profunda análise e compreensão da aplicação.		Excesso de exposição de dados geralmente levam ao vazamento de dados sensíveis.	

A API está vulnerável?

A API retorna dados sensíveis ao cliente por padrão. Este dado então é filtrado no lado do cliente antes de ser apresentado ao usuário. Um atacante pode facilmente investigar o tráfego e enxergar os dados sensíveis.

Cenários de exemplo de ataques

Cenário #1

O time de desenvolvimento móvel utiliza o *endpoint* /api/articles/{articleId}/comments/{commentId} para exibir metadados dos comentários em artigos. Investigando o tráfego do aplicativo móvel, um atacante encontra outros dados sensíveis relacionados ao autores de comentários que também são entregues. Isto acontece por uma implementação genérica do *endpoint* utilizando o método de serialização toJson() ao modelo User, que também inclui dados sensíveis.

Cenário #2

Um sistema de vigilância baseado em IoT permite aos administradores a criação de usuários com diferentes níveis de permissão. Um usuário admin criou uma conta para um novo guarda de segurança que deve ter acesso somente a algumas áreas específicas do prédio. Uma vez que utiliza um aplicativo móvel, uma chamada de API é realizada em /api/sites/111/cameras para que sejam recebidas informações a respeito das câmeras disponíveis para apresentação em um painel de controle. A resposta da API contém uma lista com detalhes a respeito das câmeras no seguinte formato: /api/sites/111/cameras. Enquanto a interface gráfica exibe apenas as câmeras às quais deveria o guarda ter acesso, a resposta da API contém uma lista de todas as câmeras em uso naquele prédio.

API3:2019 Excessive Data Exposure

Como prevenir

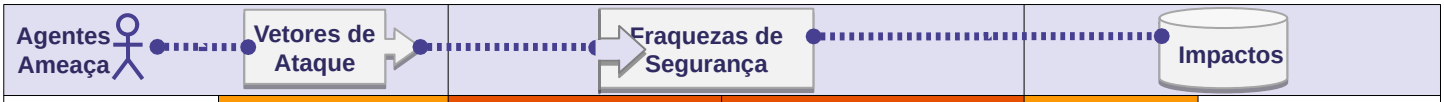
- Nunca confie no cliente para filtrar dados sensíveis.
- Revise as respostas da API para ter certeza que elas contenham apenas informações necessárias.
- Engenheiros e arquitetos de *endpoints* sempre devem se perguntar: quem irá utilizar esta informação? Antes de expor um novo *endpoint* de API.
- Tenha cuidado ao utilizar métodos genéricos como to_json() e to_string(). Ao contrário, seja criterioso com cada propriedade que seja necessário retornar.
- Classifique dados sensíveis e dados pessoais que sua aplicação armazena, revise todas as chamadas de API se estas chamadas podem significar um problema de segurança.
- Implemente respostas com mecanismos baseados em *schema* como uma camada extra de segurança. Aplique o mecanismo e o imponha a todos os dados retornados pela API, inclusive erros.

Referências

Externas

- CWE-213: Intentional Information Exposure

API4:2019 Lack of Resources & Rate Limiting



Específico da API	Explorabilidade: 2	Prevalência: 3	Detecção: 3	Técnico: 2	Específico do negócio
A exploração requer simples requisições na API. Não é necessária autenticação. Requisições múltiplas e concorrentes podem ser executadas de um único ponto utilizando um único computador ou ainda utilizando recursos em nuvem.		É comum encontrar APIs que não implementam limites ou estes limites não estão implementados corretamente.		A exploração pode levar ao DoS, levando a API à lentidão ou à completa indisponibilidade.	

A API está vulnerável?

Requisições de API consome recursos como rede, processador, memória e armazenamento. A quantidade de recursos necessária para um desempenho satisfatório depende da entrada do usuário e da lógica de negócio da API. Considere também que o fato de múltiplas chamadas na API a partir de diversos clientes irão competir pelos recursos. Um API é vulnerável se algum dos seguintes parâmetros estiverem ausentes ou mal configurados (em excesso ou muito baixos):

- Tempo limite de execução (*timeout*)
- Limite máximo de alocação de memória
- Número de *file descriptors*
- Número de processadores
- Tamanho de *payload* (ex. *uploads*)
- Número de requisições por cliente/recurso
- Número de registros por página a retornar em uma única requisição

Cenários de exemplo de ataques

Cenário #1

Um atacante faz *uploads* de imagens grandes enviando *request* POST em /api/v1/images, quando o *upload* é finalizado, a API cria múltiplos *thumbnails* com diferentes tamanhos. Devido ao grande tamanho da imagem enviada por *upload*, a memória disponível é exaurida durante a criação dos *thumbnails* e a API fica indisponível.

Cenário #2

Uma aplicação contém uma lista de usuários em uma interface com o limite de 200 usuários por página. A lista é solicitada ao servidor utilizando a seguinte query /api/users?page=1&size=100. Um atacante modifica o parâmetro *size* de 200 para 200000, provocando problemas de desempenho no banco de dados. Enquanto isso, a API torna-se indisponível e portanto incapaz de responder outras requisições deste e de todos os demais clientes (também conhecido como DoS).

Este mesmo cenário pode ser utilizado para provocar erros de *Integer Overflow* ou *Buffer Overflow*.

API4:2019 Lack of Resources & Rate Limiting

Como prevenir

- Docker torna mais fácil limitar memória, CPU, quantidade de *restarts*, *file descriptors*, e processos.
- Implemente um limite de frequência para um cliente chamar a API em um determinado espaço de tempo.
- Notifique o cliente quando o limite for excedido, informando o limite e quando o limite alcançado será reiniciado.
- Adicione validações do lado do servidor para validação de parâmetros, principalmente controles de quantidade de registros a serem retornados.
- Defina e implemente tamanhos máximos de dados recebidos por parâmetro e *payloads*.

Referências

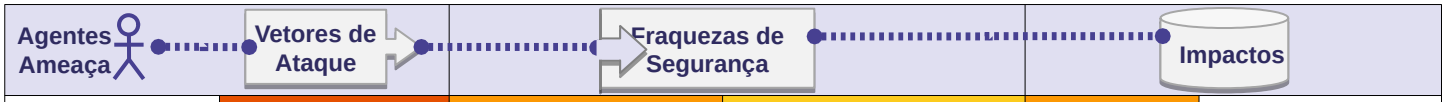
OWASP

- Blocking Brute Force Attacks
- Docker Cheat Sheet - Limit resources (memory, CPU, file descriptors, processes, restarts)
- REST Assessment Cheat Sheet

Externas

- CWE-307: Improper Restriction of Excessive Authentication Attempts
- CWE-770: Allocation of Resources Without Limits or Throttling
- "*Rate Limiting (Throttling)*" - Security Strategies for Microservices-based Application Systems, NIST

API5:2019 Broken Function Level Authorization



Específico da API	Explorabilidade: 3	Prevalência: 2	Detecção: 1	Técnico: 2	Específico do negócio
A exploração requer que o atacante envie chamadas legítimas ao *endpoint* da API ao qual não deveria ter acesso. Estes *endpoints* estão expostos para usuários anônimos ou à usuários regulares e/ou não privilegiados. É fácil encontrar estas falhas uma vez que APIs melhor estruturadas possuem funções mais previsíveis (ex. modificando métodos HTTPS de GET para PUT ou mudando a URL de "users" para "admins").		Verificações de autorização para uma função ou recurso geralmente são gerenciadas por configuração e em alguns casos a nível de código. Implementar verificações apropriadas pode ser uma tarefa confusa, uma vez que aplicativos modernos podem conter muitos tipos de papéis ou grupos e ainda uma complexa hierarquia de usuários (ex.: sub-usuários, usuários com mais de um papel).		Este tipo de falha pode permitir que atacantes tenham acesso a funções não autorizadas. Funções administrativas geralmente são os alvos deste tipo de ataque.	

A API está vulnerável?

A melhor maneira de encontrar quebras de função e autorização é executar uma profunda análise do mecanismo de autorização, ao mesmo tempo mantendo em mente a hierarquia de usuário, diferentes papéis ou grupos da aplicação e perguntando-se as seguintes questões:

- Pode um usuário regular acessar *endpoints* administrativos?
- Pode um usuário executar ações sensíveis (ex.: criação, modificação ou exclusão), mesmo apenas alterando o método HTTP (ex.: trocando GET para DELETE)?
- Pode um usuário do grupo de acesso X acessar uma função que deve ser acessível apenas para usuários do grupo Y apenas adivinhando a URL (ex.: /api/v1/users/export_all)?

Nunca considere apenas a URL como separação de *endpoints* regulares e administrativas.

Uma vez que desenvolvedores podem optar pela exposição de *endpoints* administrativos por um determinado caminho como api/admins, também é muito comum encontrar *endpoints* administrativos em caminhos similares como api/users.

API5:2019 Broken Function Level Authorization

Cenários de exemplo de ataques

Cenário #1

Durante o processo de registro de uma aplicação que permite apenas usuários convidados se cadastrarem, o aplicativo móvel realiza uma chamada de API para GET /api/invites/{invite_guid}. A resposta no formato JSON contém os detalhes sobre o convite, incluindo o papel e endereço de e-mail do usuário.

Um atacante duplica a requisição e, modifica o método HTTP e o *endpoint* para POST /api/invites/new. Este *endpoint* deveria ser acessível apenas por administradores utilizando a console administrativa, que não implementa uma autorização de nível de função.

O atacante então explora este problema enviando a si mesmo um convite para criar uma conta administrativa:

POST /api/invites/new

{“email”:”hugo@malicious.com”,”role”:”admin”}

Cenário #2

Uma API contém um endpoint que deveria estar exposta apenas para administradores: GET /api/admin/v1/users/all. Este *endpoint* retorna detalhes sobre todos os usuários e não implementa uma verificação de nível de função. Um atacante que estudou a estrutura da API e consegue encontrar este *endpoint* que expõe detalhes de todos os usuários da aplicação.

Como prevenir

Sua aplicação deve possuir um consistente módulo de autorização a ser invocado por todas suas funções. Frequentemente este tipo de proteção é provida por um ou mais componentes externos ao código da aplicação.

- O mecanismo de verificação deve negar tudo por padrão, requerendo permissões explícitas para papéis de cada função.
- Revise seus *endpoints* de API para validar falhas de autorização, enquanto mantém em mente a lógica de negócio da aplicação e hierarquia de grupos.
- Certifique-se que todos seus controles administrativos sejam herdados por um *controller* de abstração que implemente as verificações de autorização.
- Tenha certeza que funções administrativas dentro de *controllers* regulares implementem verificação de autorização baseado no usuário, grupos e papéis..

Referências

OWASP

- OWASP Article on Forced Browsing
- OWASP Top 10 2013-A7-Missing Function Level Access Control
- OWASP Development Guide: Chapter on Authorization

Externas

- CWE-285: Improper Authorization

API6:2019 Mass Assignment

Agentes Ameaça	Vetores de Ataque	Fraquezas de Segurança		Impactos	
Específico da API	**Explorabilidade: 2**	**Prevalência: 2**	**Detecção: 2**	**Técnico: 2**	**Específico do negócio**
A exploração usualmente requer compreensão da lógica de negócio, relação entre objetivos e estrutura da API. Este tipo de exploração é mais fácil em APIs, uma vez que por design elas expõem implementações internas do aplicativo junto de suas propriedades.		*Frameworks* modernos encorajam desenvolvedores a utilizar funções que automaticamente conectam a entrada do cliente em variáveis de código e objetos internos. Atacantes podem utilizar esta metodologia para atualizar ou sobrescrever propriedades em objetos sensíveis que os desenvolvedores nunca tiveram intenção de expor.		A exploração pode levar a escalação de privilégios, adulteração de dados, desvio de mecanismos de segurança, entre outros.	

A API está vulnerável?

Objetos em aplicativos modernos podem conter muitas propriedades. Algumas dessas propriedades podem ser diretamente atualizadas pelo cliente (ex.: user.first_name ou user.address), outras propriedades por sua vez podem não ser (ex.: o flag user.is_vip).

Um endpoint de API está vulnerável se ele automaticamente converte parâmetros recebidos do cliente em propriedades de objeto internos, sem considerar o quão sensível são estas mesmas propriedades. Isto pode permitir a um atacante atualizar propriedades de um objeto ao qual ele não deveria ter acesso.

Exemplos de propriedades sensíveis:

- **Propriedades relacionadas à permissões:** user.is_admin, user.is_vip devem ser escritas somente por admins.
- **Propriedades que depende de processamento:** user.cash deve ser escrita somente após a efetivação/verificação do pagamento.
- **Propriedades internas:** article.created_time deve ser escrita apenas internamente pela aplicação.

Cenários de exemplo de ataques

Cenário #1

Um aplicativo de compartilhamento de corridas permite ao usuário a opção de editar informações e dados básicos do seu perfil. Durante este processo, uma chamada à API é enviada para PUT /api/v1/users/me com o seguinte, e legítimo, objeto JSON:

{"user_name":"inons","age":24}

A requisição GET /api/v1/users/me inclui uma propriedade adicional chamada "credit_balance property":

{"user_name":"inons","age":24,"credit_balance":10}.

O atacante repete a primeira requisição com o *payload* abaixo:

{"user_name":"attacker","age":60,"credit_balance":99999}

Uma vez que o *endpoint* está vulnerável, o atacante recebe créditos sem pagar.

API6:2019 Mass Assignment

Cenário #2

Um portal de compartilhamento de vídeos permite aos usuários o envio de conteúdo e download de conteúdo em diferentes formatos. Um atacante explora a API no endpoint GET /api/v1/videos/{video_id}/meta_data que retorna um objeto com propriedades do vídeo. Uma das propriedades é "mp4_conversion_params":"-v codec h264", que indica que a aplicação usa um comando shell para converter o vídeo.

Este mesmo atacante encontrou o endpoint POST /api/v1/videos/new que está vulnerável e permite ao cliente atribuir qualquer propriedade ao objeto vídeo, então o atacante atribui um valor malicioso como o exemplo a seguir: "mp4_conversion_params":"-v codec h264 && format C:/". Este valor poderá causar a execução de um comando shell quando o atacante pedir o download do vídeo no formato mp4.

Como prevenir

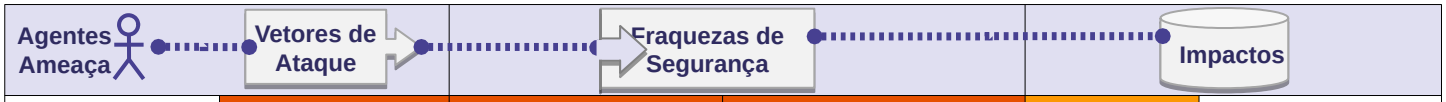
- Se possível evite usar funções que automaticamente conectam entradas de usuário em variáveis no código ou em objetos internos.
- Desenvolva controles para permitir que apenas determinadas propriedades possam ser atualizadas pelo cliente.
- Use recursos *built-in* para criar listas de proibição do que não deve ser acessado pelo cliente.
- Se aplicável, aplique de maneira explícita *schemas* nas entradas de *payload* dos usuários.

Referências

Externas

- CWE-915: Improperly Controlled Modification of Dynamically-Determined Object Attributes

API7:2019 Security Misconfiguration



Específico da API	Explorabilidade: 3	Prevalência: 3	Detecção: 3	Técnico: 2	Específico do negócio
Atacantes eventualmente procuram falhas não corrigidas, *endpoints* comuns, ou diretórios não protegidos para ganhar acesso não-autorizado ou realizar um reconhecimento do sistema.		Configurações inadequadas de segurança podem ocorrer a qualquer nível do *stack* da API, desde o nível da rede até o nível da aplicação. Ferramentas de automação estão disponíveis para detectar e explorar erros de configuração, como serviços desnecessários e opções de suporte ao legado.		Configurações inadequadas podem não apenas expor dados sensíveis de usuários, como também podem revelar detalhes do sistema e comprometer servidores como um todo.	

A API está vulnerável?

Sua API pode estar vulnerável se:

- Configurações apropriadas de *hardening* faltam a qualquer parte da aplicação, ou quando há permissões mal configuradas em provedores de nuvem.
- Os últimos *patches* de segurança não estão aplicados ou os sistemas estão desatualizados.
- Recursos não necessários estão habilitados (ex.: certos verbos HTTP).
- Transporte criptografado (TLS) não configurado.
- Diretivas de segurança não enviadas aos clientes (ex., Cabeçalhos de Segurança).
- Configurações de política CORS (*Cross-Origin Resource Sharing*) não configuradas ou configuradas inadequadamente.
- Mensagens de erro incluindo *stack trace* ou informações sensíveis.

Cenários de exemplo de ataques

Cenário #1

Um atacante encontra o arquivo .bash_history no diretório root do servidor, o qual contém comandos utilizados pelo time de DevOps para acesso à API:

$ curl -X GET 'https://api.server/endpoint/' -H 'authorization: Basic Zm9vOmJhcg=='

Um atacante pode também encontrar novos *endpoints* da API que são utilizados apenas pelo time de DevOps os quais não constam na documentação.

Cenário #2

Para mirar um serviço em específico, um atacante utiliza uma popular ferramenta de pesquisa na web para encontrar servidores que estão diretamente acessíveis na internet. Este atacante encontra um *host* executando um popular serviço de gerenciamento de banco de dados, o qual está ouvindo na porta padrão. Este mesmo *host* utiliza configurações padrão do sistema de gerenciamento de banco de dados, o qual a autenticação de acesso é desabilitada por padrão, então o atacante consegue acesso à milhares de registros com dados pessoais sensíveis e dados de autenticação.

API7:2019 Security Misconfiguration

Cenário #3

Inspecionando o tráfego de um aplicativo móvel, um atacante encontra que nem todo o tráfego HTTP está sendo executado em protocolo seguro (ex.: TLS). O atacante confirma esta condição ao realizar o download de imagens de perfis. Como a interação do usuário nesse caso é binária, apesar do fato de o tráfego da API ser realizado com protocolo seguro, o atacante encontra um padrão no tamanho das respostas da API e utiliza isso para monitorar preferências de usuário sobre o conteúdo renderizado (Ex. Imagens de perfil).

Como prevenir

O ciclo de vida da API deve incluir:

- Um processo de *hardening* contínuo levando a um rápido e fácil modelo de entrega a um ambiente apropriadamente protegido.
- Uma tarefa de revisão e atualização de configurações em todo o *stack* da API, essa revisão deve incluir: Arquivos de orquestração, componentes de API, serviços de nuvem (ex.: permissões de *buckets*).
- Um canal de comunicação segura para todos os pontos de interação da API, inclusive objetos estáticos (Ex.: Imagens).
- Processo automatizado para continuamente avaliar a efetividade das configurações e preferências em todos os ambientes.

Além disso:

- Para prevenir que detalhes de erros e outras informações sejam enviados de volta aos atacantes, se aplicável, defina e aplique *schemas* aos responses da API.
- Certifique-se que a API só pode ser acessada por verbos HTTP específicos. Todos os demais verbos devem estar desabilitados (ex: HEAD).
- APIs que devem ser acessadas por navegadores (ex.: *front-end* de aplicação web) devem implementar uma política CORS apropriada.

Referências

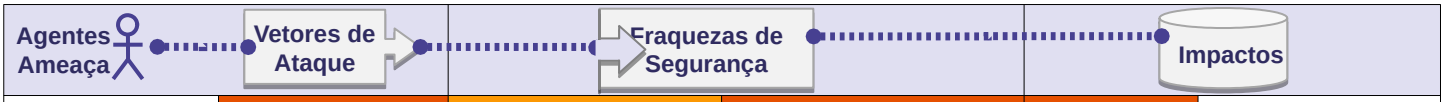
OWASP

- OWASP Secure Headers Project
- OWASP Testing Guide: Configuration Management
- OWASP Testing Guide: Testing for Error Codes
- OWASP Testing Guide: Test Cross Origin Resource Sharing

Externas

- CWE-2: Environmental Security Flaws
- CWE-16: Configuration
- CWE-388: Error Handling
- Guide to General Server Security, NIST
- Let’s Encrypt: a free, automated, and open Certificate Authority

API8:2019 Injection



Específico da API	Explorabilidade: 3	Prevalência: 2	Detecção: 3	Técnico: 3	Específico do negócio
Atacantes vão entregar à API dados maliciosos a quaisquer vetores de injeção disponíveis (ex.: dados, parâmetros, integrações e etc.), esperando que estes dados sejam entregues a um interpretador.		Falhas por injeção são muito comuns e geralmente encontrados em consultas SQL, LDAP ou noSQL, comando em SO, *parsers* de XML e ORM. São falhas simples de encontrar ao revisar código fonte, e atacantes podem utilizar *scanners* e *fuzzers*.		Injeção pode levar ao vazamento de informação e perda de dados. Também podem levar à DoS ou a perda completa de um *host*.	

A API está vulnerável?

A API está vulnerável às falhas de injeção se:

- Dados enviados pelo cliente não são validados, filtrados ou sanitizados pela API.
- Dados enviados pelo cliente são utilizados diretamente ou concatenados para consultas SQL/NoSQL/LDAP, comandos de sistema operacional, *parsers* XML, ORM (*Object Relational Mapping*) ou ODM (*Object Document Mapper*).
- Dados vindos de sistemas externos (ex.: sistemas de integração) não são validados, filtrados ou sanitizados pela API.

Cenários de exemplo de ataques

Cenário #1

O firmware de um dispositivo de controle parental provê o endpoint /api/CONFIG/restore o qual espera um appId a ser enviado como um parâmetro *multipart*. Utilizando um descompilador, o atacante encontra que o parâmetro appId é repassado diretamente para uma chamada de sistema sem qualquer sanitização:

snprintf(cmd, 128, "%srestore_backup.sh /tmp/postfile.bin %s %d",

"/mnt/shares/usr/bin/scripts/", appid, 66);

system(cmd);

Com o seguinte comando o atacante consegue desligar qualquer dispositivo que estiver com o *firmware* vulnerável:

$ curl -k "https://${deviceIP}:4567/api/CONFIG/restore" -F 'appid=$(/etc/pod/power_down.sh)'

Cenário #2

Estamos com uma aplicação com funcionalidades básicas de CRUD para operações de agendamento. Um atacante identifica que uma injeção NoSQL pode ser possível por meio do um parâmetro bookingId informado via *querystring* na requisição de exclusão de pedido de agendamento. Eis a requisição em questão: DELETE /api/bookings?bookingId=678.

API8:2019 Injection

O servidor da API utiliza a seguinte função para executar a requisição de exclusão:

```
router.delete('/bookings', async function (req, res, next) {
  try {
     const deletedBooking = await Bookings.findOneAndRemove({'_id' : req.query.bookingId});
     res.status(200);
  } catch (err) {
    res.status(400).json({error: 'Unexpected error occured while processing a request'});
  }
});
```

O atacante intercepta a requisição de modifica o valor da *querystring* bookingId como demonstrado abaixo. Neste caso, o atacante consegue excluir todos os demais agendamentos de usuários:

```
DELETE /api/bookings?bookingId[$ne]=678
```

Como prevenir

Prevenir injeção requer manter os dados separados de comandos e consultas.

- Execute validação de dados utilizando uma biblioteca única, confiável e de ativa manutenção.
- Valide, filtre e sanitize todos os dados providos pelo cliente, e também dados vindos de sistemas integradores.
- Caracteres especiais devem ser avaliados utilizando a sintaxe específica do interpretador dos comandos.
- Prefira APIs seguras, que entreguem interfaces seguras e parametrizadas.
- Sempre limite o número de registros retornados para prevenir vazamento em massa em caso de injeção.
- Valide os dados recebidos utilizando filtros suficientes para permitir que apenas valores válidos cheguem aos interpretadores.

Defina tipos de dados de padrões *strict* em todos os parâmetros do tipo *string*.

Referências

OWASP

- OWASP Injection Flaws
- SQL Injection
- NoSQL Injection Fun with Objects and Arrays
- Command Injection

Externas

- CWE-77: Command Injection
- CWE-89: SQL Injection

API9:2019 Improper Assets Management

Específico da API	Explorabilidade: 3	Prevalência: 3	Detecção: 2	Técnico: 2	Específico do negócio
Versões desatualizadas de APIs geralmente carecem de patches e são um meio fácil de comprometer sistemas sem a presença de mecanismos de segurança no estado-da-arte, e que podem existir também para proteger versões atualizadas de APIs.		Documentação desatualizada torna mais difícil encontrar e/ou corrigir vulnerabilidades. Falta de um inventário de ativos e estratégias de retirada levam ao cenário de sistemas sem atualização permanecerem em execução, podendo resultar no vazamento de dados sensíveis. É bastante comum encontrar APIs expostas sem necessidade em razão de conceitos modernos como os de microsserviços, que permite que aplicativos sejam lançados independentemente (ex.: computação em nuvem, kubernetes).		Atacantes têm acesso a dados sensíveis e até tomar o controle de servidores por meio de uma velha e desatualizada API conectada ao mesmo banco de dados.	

A API está vulnerável?

A API pode estar vulnerável se:

- O propósito do host da API não for claro, e se não há respostas explícitas para as seguintes questões:
  - Em qual ambiente está rodando à API? (Ex.: produção, *staging*, teste, desenvolvimento)?
  - Quem deve ter acesso via rede à API (Ex.: pública, interna, parceiros)?
  - Em qual versão está a API em execução?
  - Que tipo de informação acessa a API (Ex.: Dados pessoais sensíveis)?
  - Qual é o fluxo da informação?
- Não existe documentação, ou a documentação existente está desatualizada.
- Não há um plano de retirada para cada versão da API.
- Inventário de *hosts* não existe ou está desatualizado.
- Inventário de serviços de integração, seja interna ou de parceiros, não existe ou está desatualizado.
- Versões antigas da API continuam rodando sem *patches*.

Cenários de exemplo de ataques

Cenário #1

Após um redesenho de suas aplicações, um serviço de pesquisa local deixou uma versão antiga da API (api.someservice.com/v1) em execução, não protegida, e com acesso ao banco de dados. Enquanto buscava como alvo a última versão do aplicativo, um atacante percebeu o endereço da API (api.someservice.com/v2). Substituindo v2 por v1 na URL, o atacante acessou a versão antiga e não protegida, a qual expõe dados pessoais sensíveis de mais de 100 milhões de usuários.

API9:2019 Improper Assets Management

Cenário #2

Uma rede social implementou um nível de classificação mínimo que bloqueia atacantes do uso de força bruta para conseguir acesso por meio de adivinhação de *tokens* de redefinição de senhas de acesso. Este mecanismo não foi implementado no código próprio da API, mas em um componente separado entre o cliente e a API oficial (www.socialnetwork.com). Um pesquisador encontrou a versão beta da API (www.mbasic.beta.socialnetwork.com) que executa a mesma API, incluindo o mecanismo de redefinição de senha, onde o nível de classificação mínimo não está ativado. Dessa maneira ele pode redefinir a senha de qualquer usuário com um mecanismo simples de força bruta para adivinhar o *token* de seis dígitos..

Como prevenir

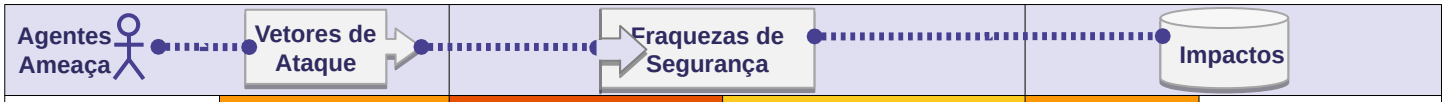
- Faça o inventário de todos os *hosts* de API e documente aspectos importantes de cada um deles, com foco no ambiente das APIs (Ex.: produção, *staging*, testes, desenvolvimento), e qual desses ambientes deve ter acesso à quais redes (Ex.: pública, interno, parceiros) e as versões da API.
- Faça o inventário de todos os serviços de integração e documente os aspectos importantes como o papel de cada um deles no sistema, qual tipo de dado é trocado e se estes dados são sensíveis.
- Documente todos os aspectos da sua API, como autenticação, erros, redirecionamentos, limite de classificação, política de *cross-origin resource sharing* (CORS) e seus *endpoints*, incluindo os parâmetros, requisições e respostas.
- Faça documentações automatizadas utilizando padrões abertos, inclua a documentação de compilação no pipeline de CI/CD.
- Faça a documentação disponível para aqueles autorizados à utilizá-la.
- Utilize métricas de proteção como *firewalls* de APIs para todas as versões expostas e não apenas para a versão em produção.
- Evite utilizar dados de produção em *deployments* de API em ambientes de não produção. Caso seja impossível, estes *endpoints* devem possuir o mesmo tratamento de segurança daqueles que estão em produção.
- Quando novas versões da API incluir melhorias de segurança, faça uma análise de risco para auxiliar a decisão de mitigação de ações necessárias para a versão antiga da API. Por exemplo: sempre que for possível utilizar versões antigas sem quebrar compartibilidade, você precisa trabalhar para que todos os clientes façam o movimento para a última versão..

Referências

Externas

- CWE-1059: Incomplete Documentation
- OpenAPI Initiative

API10:2019 Insufficient Logging & Monitoring



Específico da API	Explorabilidade: 2	Prevalência: 3	Detecção: 1	Técnico: 2	Específico do negócio
Atacantes podem tirar proveito de pouco log e monitoramento para abusar de sistemas sem serem notados..		Sem log e monitoramento, ou log e monitoramento insuficiente, é quase impossível rastrear atividades suspeitas e dar respostas à elas tem tempo hábil.		Sem visibilidade do que está ocorrendo de atividades maliciosas, atacantes possuem tempo para comprometer completamente sistemas..	

A API está vulnerável?

A API está vulnerável se:

- Não produz qualquer tipo de log, ou se o nível de log não é configurado adequadamente, ou ainda, se as mensagens de log não incluem informações suficientes.
- A integridade do log não é garantida (ex.: Injeção de Log).
- Logs não estão em contínuo monitoramento.
- A infraestrutura da API não é continuamente monitorada..

Cenários de exemplo de ataques

Cenário #1

Chaves de acesso de administração da API são vazados em um repositório público. O proprietário do repositório é notificado por e-mail a respeito do provável vazamento, mas, até que uma ação seja realizada em reposta ao incidente se passaram 48 horas. Em razão de logs insuficientes, a companhia não é capaz de identificar quais informações foram acessadas durante o período por atores maliciosos..

Cenário #2

Uma plataforma de compartilhamento de vídeos foi atingida por um ataque de *credential stuffing* de larga escala. Mesmo com os logins que falharam sendo logados, não houveram alertas disparados durante o tempo de duração do ataque. Como uma resposta à reclamação dos usuários, os logs de API foram analisados e o ataque foi detectado. A companhia fez um anúncio público solicitando aos seus usuários que façam atualização de suas senhas, e reportam o incidente às autoridades regulatórias..

API10:2019 Insufficient Logging & Monitoring

Como prevenir

- Faça log de todas tentativas de logon mal sucedidas, acessos negados e erros de validação de entradas de usuários.
- Logs devem ser escritos em um formato apropriado para serem consumidos por soluções de gerenciamento de logs, e devem incluir detalhes suficientes para ajudar a identificar o ator malicioso.
- Logs devem ser manipulados como dados sensíveis, e sua integridade deve ser garantida tanto em trânsito como em repouso.
- Configure o sistema de monitoramento a monitorar continuamente a infraestrutura, rede e o funcionamento da API.
- Utilize um SIEM (*Security Information and Event Management*) para agregar e gerenciar logs oriundos de todos componentes da arquitetura da API e seus *hosts*.
- Configure painéis de controle e alertas, possibilitando de atividades suspeitas sejam detectadas e respondidas de forma breve..

Referências

OWASP

- OWASP Logging Cheat Sheet
- OWASP Proactive Controls: Implement Logging and Intrusion Detection
- OWASP Application Security Verification Standard: V7: Error Handling and Logging Verification Requirements

Externas

- CWE-223: Omission of Security-relevant Information
- CWE-778: Insufficient Logging

+D Próximos passos para Desenvolvedores

A tarefa de criar e manter software seguro, ou, corrigir software existente, pode ser uma tarefa difícil. APIs não são diferentes.

Nós acreditamos que a educação e conscientização são fatores chave para escrever software seguro. Tudo o que é necessário para alcançar este objetivo depende de **estabelecer e usar procedimentos de segurança reprodutíveis e padronizar controles de segurança**.

O OWASP possui um grande número de fontes abertas para endereçar a segurança desde o princípio dos projetos. Por favor, visite a página de projetos do OWASP para uma extensa lista de projetos disponíveis.

Educação	Você pode iniciar lendo os materiais de educação do OWASP de acordo com sua área ou interesse. Para aprendizado mão-na-massa, nós adicionamos crAPI - Completely Ridiculous API (API Ridiculamente Vulnerável) em nosso *roadmap*. Enquanto isso, você pode praticar segurança de aplicações web utilizando o OWASP DevSlop Pixi Module, um aplicativo web vulnerável e um serviço API com a intenção de ensinar usuário como testar aplicações web modernas e também APIs com relação a problemas de segurança. Você pode também participar das conferências AppSec do OWASP com sessões de treinamento, ou ainda juntar-se a seu capítulo local.
Requisitos de Segurança	Segurança deve fazer parte de qualquer projeto desde o princípio. Ao fazer a escolha de requisitos, é importante definir o que a "segurança" representa para o projeto. O OWASP recomenda o uso do OWASP Application Security Verification Standard (ASVS) como um guia para atribuir requisitos de segurança. Se você trabalha com *outsourcing*, considere o projeto OWASP Secure Software Contract Annex, o qual deve ser adaptado de acordo com as leis e regulamentos locais.
Arquitetura de Segurança	Seguraça deve permanecer uma preocupação durante todas as fases de um projeto. O OWASP Prevention Cheat Sheets é um bom ponto de partida e um guia sobre como o design de segurança durante a fase de arquitetura. Junto de vários outros, você encontrará REST Security Cheat Sheet e também REST Assessment Cheat Sheet com abordagem de aspetos de APIs.
Controle de Segurança Padrão	A adoção de padrões de controles de segurança reduzem o risco da introdução de fraquezas durante o desenvolvimento da lógica específica do negócio no software. Fora o fato que *frameworks* modernos incluírem por padrão controles de segurança efetivos, considere o OWASP Proactive Controls que entrega uma boa visão geral sobre quais controles de segurança você deve avaliar e incluir em seu projeto. O OWASP também entrega algumas bibliotecas e ferramentas que podem ser úteis, como validação de controles.
Ciclo de Vida do Software Seguro	Você pode utilizar o OWASP Software Assurance Maturity Model (SAMM) para melhorar seu processo enquanto constrói APIs. Muitos outros projetos do OWASP possuem valor para ajudá-lo com as diferentes fases do desenvolvimento de suas API, por ex.: o OWASP Code Review Project.

+DSO Próximos passos para DevSecOps

Considerando sua importância na arquitetura de aplicações modernas, a construção de APIs seguras é crucial. A segurança não pode ser negligenciadas, e deve fazer parte de todo o ciclo de vida de desenvolvendo. Executar verificações e testes de penetração anualmente não é mais suficiente.

DevSecOps deve se juntar aos esforços de desenvolvimento, facilitando a execução de testes contínuos de segurança durante todo o ciclo de desenvolvimento de software. O objetivo é melhorar o pipeline de desenvolvimento com automação de segurança, sem contudo impactar negativamente a velocidade do desenvolvimento.

Em caso de dúvidas, mantenha-se informado, e reveja o Manifesto DevSecOps frequentemente.

Compreensão do modelo de ameaça	Teste de prioridades vem do modelo da ameaça. Se você não possui um, considere usar os projetos OWASP Application Security Verification Standard (ASVS), e também OWASP Testing Guide como entrada. Envolver a equipe de desenvolvimento pode ajudar a torná-los mais conscientes da segurança.
Compreensão do SDLC	Junte-se ao time de desenvolvimento para melhor compreensão do ciclo de desenvolvimento de software (SDLC - Software Development Life Cycle). Sua contribuição com testes de segurança contínuos deve ser compatível com pessoas, processos e ferramentas. Todos devem concordar com o processo, assim evita-se atritos e resistências.
Estratégias de testes	Uma vez que seu trabalho não deve impactar negativamente a velocidade do desenvolvimento, você deve escolher com cuidado a melhor (simples, rápida e precisa) técnica de verificação de requisitos de segurança. Consulte os projetos OWASP Security Knowledge Framework e OWASP Application Security Verification Standard que podem ser excelentes fontes de requisitos de segurança funcionais e não funcionais. Outras ótimas fontes de consulta são os projetos e ferramentas similares ao oferecidos pela comunidade DevSecOps.
Alcançando cobertura e precisão	Você é a ponte entre os times de desenvolvimento e operações. Para alcançar cobertura, não dê atenção somente a funcionalidade mas também à orquestração. Trabalhe próximo ao dois times desde o início e assim você consegue otimizar seu tempo e esforço. Você deve também encontrar um momento onde o essencial da segurança seja continuamente verificado.
Comunique claramente problemas	Contribua com menos ou nenhum atrito. Distribua os problemas encontrados em tempo hábil, utilizando as ferramentas que os times utilizam em sua rotina (nunca documentos PDF). Junte-se ao time de desenvolvimento para a correção de problemas encontrados. Construa oportunidades para educá-los, descrevendo claramente as fraquezas e como estas podem ser exploradas, incluindo cenários de ataques que demonstre cenários reais.

+DAT Dados e Metodologia

Visão Geral

Como o setor de segurança de aplicação não se concentrou especificamente em mais recentes arquiteturas de aplicativos, na qual as API possuem um papel importante, compilar uma lista dos dez mais críticos riscos de segurança de API, baseados em uma chamada pública por informações, teria sido uma árdua tarefa. Apesar de não ter havido esta chamada por dados, a lista dos dez mais críticos ainda é baseada em dados públicos e com contribuições de *experts* em segurança, além de discussões abertas da comunidade.

Metodologia e Dados

Na primeira fase, dados disponíveis publicamente sobre incidentes de segurança envolvendo APIs, os quais foram coletados por *experts* em segurança, e então foram revisados e categorizados. Estas informações foram coletadas de plataformas de *bug bounty* e bancos de dados de vulnerabilidades durante um ano inteiro, como propósito estatístico.

Na fase seguinte profissionais de segurança com experiência em testes de penetração foram questionados a colaborar com suas listas de dez maiores ameaças à segurança de API.

Então a Metodologia OWASP para classificação de risco foi utilizada para a elaboração da análise de risco. Os resultados foram discutidos e revisados entre os profissionais de segurança. Para considerações nesse sentido, por favor, consulte o item Riscos de Segurança de API.

O primeiro rascunho do projeto é resultado de um consenso entre os dados estatísticos adquiridos na primeira fase com a lista entregue pelos profissionais de segurança. Este rascunho foi submetido à apreciação e revisão a um segundo grupo de profissionais de segurança com relevante experiência em segurança de APIs.

O projeto OWASP API Security Top 10 2019 então foi apresentado pela primeira vez no evento OWASP Global AppSec Tel Aviv (Maio 2019), e desde então, está disponível no GitHub para discussões e contribuições.

A lista de colaboradores está disponível na seção Agradecimentos.

+ACK Agradecimentos

Agradecimentos aos colaboradores

Gostaríamos de agradecer os seguintes colaboradores que contribuiram publicamente via GitHub ou outros meios:

- 007divyachawla
- Abid Khan
- Adam Fisher
- anotherik
- bkimminich
- caseysoftware
- Chris Westphal
- dsopas
- DSotnikov
- emilva
- ErezYalon
- flascelles
- Guillaume Benats
- IgorSasovets
- Inonshk
- JonnySchnittger
- jmanico
- jmdx
- Keith Casey
- kozmic
- LauraRosePorter
- Matthieu Estrade
- nathanawmk
- PauloASilva
- pentagramz
- philippederyck
- pleothaud
- r00ter
- Raj kumar
- Sagar Popat
- Stephen Gates
- thomaskonrad
- xycloops123

Tradução para o português do Brasil

- Raphael Hagi
- Bruno Barbosa
- Eduardo Bellis