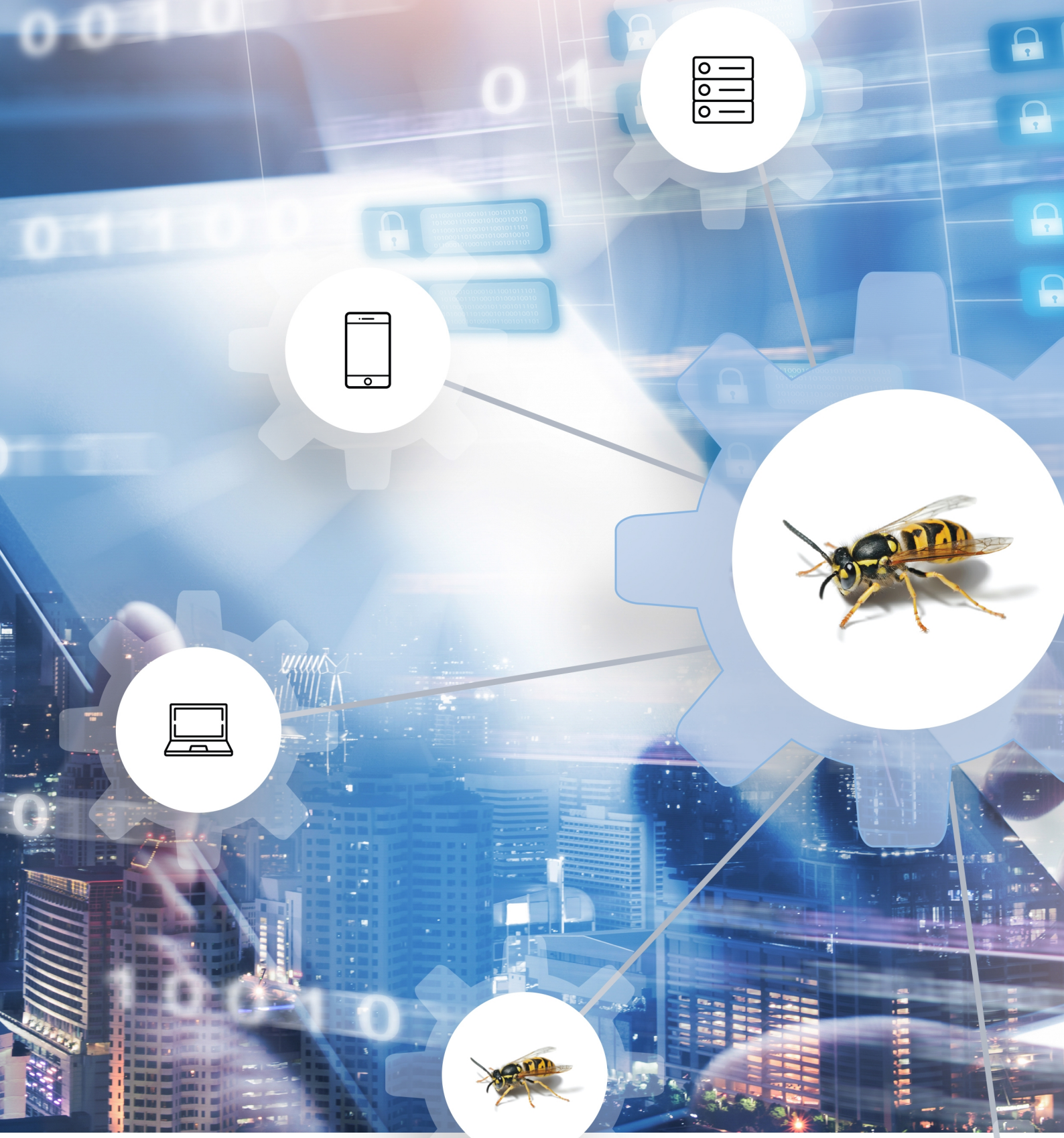


OWASP API Security Top 10 2019

Os Dez Problemas de Segurança Mais Críticos em APIs

TOC

Tabela de Conteúdos

Índice

Sobre a OWASP

OWASP - Open Web Application Security Project é uma comunidade aberta que se dedica a ajudar as organizações a desenvolver, adquirir e manter aplicações e APIs confiáveis.

A OWASP disponibiliza de forma livre e aberta:

- Ferramentas e normas de segurança aplicacional
- Livros completos sobre testes de segurança aplicacional, desenvolvimento de código seguro e revisão de código focada em segurança
- Apresentações e vídeos
- *Cheat Sheets* sobre assuntos diversos
- Controlos e bibliotecas de segurança *standard*
- Comunidades locais espalhados por todo o mundo
- Investigação de ponta
- Múltiplas conferências em todo o mundo
- Listas de discussão

Mais informação em: https://www.owasp.org.

Todas as ferramentas, documentos, vídeos, apresentações e comunidades locais da OWASP são livres e abertos a todos os interessados em melhorar a segurança aplicacional.

Aconselhamos uma abordagem à segurança aplicacional como sendo um problema de pessoas, processos e tecnologia, porque as abordagens mais eficazes à segurança aplicacional necessitam de melhorias em todas estas áreas.

A OWASP é um novo tipo de organização. A nossa independência em relação a pressões comerciais permite-nos fornecer informação imparcial, prática e economicamente adequada sobre a segurança aplicacional. A OWASP não está afiliada com nenhuma empresa tecnológica, embora suportemos o uso informado de tecnologias de segurança comerciais. A OWASP produz muitos tipos de materiais de uma forma colaborativa, transparente e aberta.

A fundação OWASP é uma entidade sem fins lucrativos o que assegura o sucesso a longo prazo do projeto. Quase todas as pessoas associadas à OWASP são voluntárias, incluindo a direção da OWASP, os líderes das comunidades locais, os líderes dos projetos e os seus membros. Suportamos investigação inovadora em segurança através de bolsas e infraestrutura.

Junte-se a nós!

Prefácio

As APIs - *Application Programming Interface* têm um papel fundamental na inovação que observamos nos dias de hoje ao nível das aplicações. Desde a banca, retalho e transportes à Internet das Coisas (IoT), veículos autónomos e *Smart Cities*, as APIs são hoje um elemento crítico nas aplicações móveis, *Software as a Service* (SaaS) e aplicações web, sejam elas destinadas ao público em geral, parceiros de negócio ou para uso interno das organizações.

Por definição as APIs expõem lógica aplicacional e dados sensíveis tais como informação pessoal (PII - *Personally Identifiable Information*), motivo pelo qual se têm vindo a tornar um alvo para os atacantes. Se não conseguirmos garantir a segurança das APIs será impossível continuar a inovar a um ritmo acelerado.

Apesar de continuar a fazer sentindo manter uma lista dos 10 principais problemas de segurança em aplicações web, devido à natureza particular das APIs, é importante haver também uma tal lista específica para APIs. A segurança das APIs foca-se nas estratégias e soluções para compreender e mitigar as vulnerabilidades e risco de segurança associado às APIs.

Se estiver familiarizado com o projeto OWASP Top 10 com certeza notará as semelhanças entre os documentos: elas são propositadas para facilitar a leitura e adoção deste. Se por outro lado for a primeira vez que tem contacto com um documento da série OWASP Top 10, sugerimos que comece por ler as secções Riscos de Segurança em APIs e Metodologia e Dados antes de aprofundar a lista dos dez problemas de segurança mais críticos em APIs.

Pode contribuir para o OWASP API Security Top 10 com perguntas, comentários e ideias no repositório do projeto no GitHub:

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

Pode ainda encontrar o OWASP API Security Top 10 em:

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

Gostaríamos de agradecer a todos os que participaram neste projeto, tornando-o possível com o seu empenho e contribuições. A lista de contribuidores encontra-se na secção Agradecimentos.

Obrigado!

I Introdução

Bem-vindo ao OWASP API Security Top 10 - 2019!

Bem-vindo à primeira edição do OWASP API Security Top 10. Se estiver familiarizado com a série OWASP Top 10 seguramente notará as semelhanças: elas são propositadas em prol da compreensão e adoção deste documento. Caso este seja o seu primeiro contacto com este tipo de documento, considere visitar a página do OWASP API Security Project antes de prosseguir para os principais problemas de segurança de APIs.

As APIs desempenham um papel muito importante na arquitetura das aplicações modernas. Uma vez que a consciencialização para a segurança e a inovação têm ritmos diferentes, é importante concentrarmo-nos nas falhas mais comuns em APIs.

O objetivo principal do OWASP API Security Top 10 é educar todos aqueles envolvidos no desenvolvimento e manutenção de APIs, como por exemplo, programadores, *designers*, arquitetos, gestores ou organizações.

Na secção Metodologia e Dados pode ler mais sobre como esta primeira edição foi criada. Nas versões futuras queremos envolver a industria de segurança através duma chamada pública para contribuição de dados. Por agora encorajamos todos a contribuírem com perguntas, comentários e ideias no nosso repositório no GitHub ou através da *Mailing list*.

RN Notas da Versão

Esta é a primeira edição do OWASP API Security Top 10, que prevemos atualizar periodicamente a cada três ou quatro anos.

Ao contrário desta versão, em futuras versões, queremos fazer uma chamada pública para contribuição de dados, envolvendo a industria de segurança neste esforço. Na secção Metodologia e Dados encontrará mais detalhes sobre como construímos esta versão. Para mais informação sobre os riscos de segurança, por favor consulte a secção Riscos de Segurança em APIs.

É importante tomar consciência que nos últimos anos a arquitetura das aplicações sofreu alterações significativas. Atualmente as APIs desempenham um papel muito importante, em particular em arquitetura de micro-serviços, *Single Page Applications* (SPAs), aplicações móveis, Internet da Coisas (IoT), etc.

Era imperativo criar o OWASP API Security Top 10 para consciencializar a comunidade sobre os atuais problemas de segurança em APIs. Isto foi apenas possível graças ao enorme esforço dum conjunto de voluntários, todos eles mencionados na secção Agradecimentos. Obrigado!

RISK Riscos de Segurança em APIs

Para a análise de risco usámos a metodologia de avaliação de risco da OWASP.

A tabela seguinte resume a terminologia associada à pontuação correspondente ao nível de risco.

Agentes Ameaça	Abuso	Prevalência	Deteção	Impacto Técnico	Impacto Negócio
Específico da API	Fácil **3**	Predominante **3**	Fácil **3**	Grave **3**	Específico do Negócio
	Moderado **2**	Comum **2**	Moderado **2**	Moderado **2**	
	Difícil **1**	Incomum **1**	Difícil **1**	Reduzido **1**	

Nota: Esta abordagem não toma em consideração a verosimilhança do Agente de Ameaça. Também não toma em consideração nenhum detalhe técnico associado à sua API. Qualquer um destes fatores podem ter impacto significativo na probabilidade dum atacante encontrar e abusar duma falha de segurança particular. Estes indicadores não tomam em consideração o impacto atual no seu negócio. Terá de ser a sua organização a decidir qual o nível de risco para a segurança das suas aplicações e APIs que está disposta a aceitar, baseado na cultura, indústria e regulação a que está sujeita. O propósito do OWASP API Security Top 10 não é fazer essa análise por si.

Referências

OWASP

- OWASP Risk Rating Methodology
- Artigo sobre 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	As APIs tendem a expor mais *endpoints* que manipulam identificadores de objetos, tornando as falhas no controlo de acessos mais suscetíveis a ataques. A verificação da autorização para acesso aos objetos deve ser tida em consideração em todas as funções que acedem a dados com base em informação fornecida pelo utilizador.
API2:2019 - Broken User Authentication	Com frequência os mecanismos de autenticação são implementados de forma incorreta, permitindo aos atacantes comprometer os *tokens* de autenticação ou abusar das falhas na implementação por forma a assumir a identidade de outros utilizadores de forma temporária ou permanente.
API3:2019 - Excessive Data Exposure	Na tentativa de fazer implementações genéricas os programadores tendem a expor todas as propriedades dum objeto sem ter em consideração quão sensível é cada uma delas, delegando nos clientes a filtragem daquelas que devem ser apresentadas ao utilizador.
API4:2019 - Lack of Resources & Rate Limiting	Com frequência as APIs não impõem quaisquer restrições no tamanho ou número de recursos que um cliente/utilizador pode solicitar. Não só isto pode ter impacto no desempenho do servidor da API, conduzindo à negação do serviço (DoS), mas também deixa a porta aberta para problemas de autenticação tais como ataques de força bruta.
API5:2019 - Broken Function Level Authorization	Política de controlo de acesso complexas com diferentes níveis hierárquicos, grupos e perfis e uma não tão clara separação entre o que são ou não funcionalidades administrativas tendem a conduzir a falhas de autorização. Abusando destas falhas os atacantes podem ganhar acesso a recursos doutros utilizadores e/ou a funcionalidades administrativas.
API6:2019 - Mass Assignment	Atribuir a informação fornecida pelo cliente (e.g., JSON) aos modelos de dados sem a devida filtragem das propriedades com base em *whitelists*, conduzem tipicamente a problemas de atribuição em massa (*Mass Assignment*). Quer seja através da adivinhação das propriedades do objeto, explorando outros *endpoints* da API ou consulta da documentação, fornecendo propriedades adicionais no conteúdo dos pedidos permite aos atacantes modificar propriedades dos objetos que não eram supostos.
API7:2019 - Security Misconfiguration	Tipicamente as más configurações de segurança resultam de configurações por omissão, incompletas ou que se destinam a um fim específico, armazenamento na nuvem aberto, falha na configuração dos cabeçalhos HTTP de segurança, métodos HTTP não utilizados, política permissiva de Partilha de Recursos Entre Origens (CORS) e mensagens de erro contendo informação sensível.
API8:2019 - Injection	Falhas de injeção tais como SQL, NoSQL, injeção de comandos, etc., ocorrem quando dados não confiáveis são enviados a um interpretador como parte dum comando ou consulta. Desta forma o interpretador acaba por executar comandos que não era expectável executar ou aceder a dados sem a devida autorização.
API9:2019 - Improper Assets Management	As APIs tendem a expor mais *endpoints* do que as aplicações web tradicionais, fazendo com que a documentação se torne ainda mais importante. Um inventário dos *hosts* e APIs em execução também têm um papel importante na mitigação de falhas tais como versões de APIs descontinuadas e exposição de *endpoints* para análise de problemas.
API10:2019 - Insufficient Logging & Monitoring	A insuficiência no registo e monitorização, em conjugação com a falta ou ineficácia da integração com a resposta a incidentes, permite aos atacantes continuar a sua prática, persistir os seus ataques, alcançar outros sistemas, extrair ou destruir dados. A maioria dos estudos demonstra que o tempo de deteção duma quebra de segurança vai além dos 200 dias, sendo tipicamente detetada por entidades externas, ao invés de processo internos ou monitorização.

API1:2019 Broken Object Level Authorization

Específico da API	Abuso: 3	Prevalência: 3	Deteção: 2	Técnico: 3	Específico Negócio
Os atacantes podem abusar dos *endpoints* vulneráveis da API através da manipulação do identificador dum objeto que é enviado como parte do pedido. Isto pode conduzir ao acesso não autorizado a informação sensível. Este problema é extremamente comum porque os componentes do servidor normalmente não mantém o estado do cliente, baseando-se essencialmente em parâmetros tais como o identificador do objeto que é enviado para decidir qual o objeto a aceder.		Este tem sido o ataque mais comum e com maior impacto em APIs. Os mecanismos de autorização e controlo de acessos em aplicações modernas são complexos e abrangentes. Ainda que a aplicação implemente uma infraestrutura adequada para validação de autorização, os programadores podem esquecer-se de a realizar antes de aceder a informação sensível. A identificação de problemas no controlo de acessos não é de fácil deteção através de análise estática ou dinâmica.		Acesso não autorizado pode resultar na divulgação de informação a entidades não autorizadas, perda ou manipulação de dados. O acesso não autorizado a objetos pode ainda conduzir à usurpação de contas de utilizador.	

A API é vulnerável?

A autorização de acesso ao nível do objeto é um mecanismo de controlo tipicamente implementado ao nível do código para validar que um utilizador só pode aceder aos objetos aos quais tem acesso.

Todos os *endpoints* duma API que recebem identificadores de objetos e que executam algum tipo de ação sobre os mesmos, devem implementar verificações de autorização a esse nível. A verificação de acesso deve validar que o utilizador com a sessão ativa tem permissão para realizar a ação solicitada sobre o objeto em questão.

Falhas neste mecanismo tipicamente conduzem à divulgação não autorizada de informação, modificação ou destruição de todos os dados.

Exemplos de Cenários de Ataque

Cenário #1

Uma plataforma de comércio eletrónico para criar lojas online oferece uma página de listagem com gráficos relativos à receita das lojas. Inspecionando os pedidos realizados pelo navegador um atacante identifica os *endpoints* da API usados para obter os dados a partir dos quais são gerados os gráficos bem como o seu padrão `/shops/{shopName}/revenue_data.json`. Utilizado outro *endpoint* da API o atacante obtém a lista com o nome de todas as lojas. Com recurso a um *script* simples para substituir `{shopName}` no URL pelos nomes que constam da lista, o atacante consegue acesso aos dados relativos às vendas de milhares de lojas online.

Cenário #2

Monitorizando o tráfego de rede dum dispositivo *wearable*, o pedido HTTP `PATCH` capta a atenção dum atacante devido à utilização do cabeçalho não-standard `X-User-ID: 54796`.

Substituindo o valor do cabeçalho `X-User-Id` por `54795` o atacante recebe uma resposta afirmativa, conseguindo manipular os dados da conta doutro utilizador.

API1:2019 Broken Object Level Authorization

Como Prevenir

- Implementar um mecanismo de autorização baseado nas políticas de utilizador e hierarquia.
- Utilizar um mecanismo de autorização para verificar se o utilizador com sessão ativa tem permissão para realizar a ação pretendida sobre o registo. Esta verificação deve ser feita por todas as funções que utilizem informação fornecida pelo cliente para aceder a um registo na base de dados.
- Utilizar preferencialmente valores aleatórios e não previsíveis (e.g., GUID) como identificador para os registos.
- Escrever testes para avaliar o correto funcionamento do mecanismo de autorização. Não colocar em produção alterações vulneráveis que não passem nos 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	Abuso: 3	Prevalência: 2	Deteção: 2	Técnico: 3	Específico Negócio
A autenticação em APIs é um mecanismo complexo e confuso. Engenheiros de software e segurança podem ter conceções erradas com relação ao âmbito da autenticação e como implementá-la corretamente. Por outro lado os mecanismos de autenticação são um alvo fácil para os atacantes uma vez que estão expostos publicamente. Estes dois pontos tornam o componente responsável pela autenticação potencialmente vulnerável a diferentes tipos de abuso.		Podemos dividir os problemas de autenticação em duas partes: 1. Falta de mecanismos de proteção: os *endpoints* responsáveis pela autenticação devem ser tratados de forma diferente dos demais *endpoints*, implementando camadas de proteção adicionais 2. Falhas na implementação do mecanismo: este é utilizado/implementado sem considerar os vetores de ataque específicos ou baseado em casos de uso desadequados (e.g., um mecanismo de autenticação desenhado para clientes IoT pode não ser a melhor escolha para aplicações web).		Os atacantes podem obter o controlo sobre as contas doutros utilizadores, aceder aos seus dados pessoais e realizar ações sensíveis em seu nome, como por exemplo transferências financeiras ou envio de mensagens pessoais.	

A API é vulnerável?

Os *endpoints* e fluxos de autenticação são ativos que carecem de proteção. Mecanismos de recuperação de *password* devem ser tratados da mesma forma que os mecanismos de autenticação.

Uma API é vulnerável se:

- Permite ataques de *credential stuffing* em que o atacante tem uma lista de nomes de utilizador e *passwords* válidos.
- Permite ataques de força bruta a uma conta de utilizador específica, não implementando mecanismos de mitigação como *captcha* ou bloqueio da conta por excesso de tentativas de autenticação falhadas.
- Permite a utilização de *passwords* fracas.
- Envia informação de autenticação, tal como *tokens* e *passwords*, no URL.
- Não valida a autenticidade dos *tokens* de autenticação.
- Aceita *tokens* JWT sem que estes sejam assinados/usando algoritmos fracos (`"alg":"none"`) ou não valida a sua data de expiração.
- Utiliza *passwords* em texto, não encriptadas ou resumos fracos.
- Utiliza chaves de encriptação fracas.

Exemplos de Cenários de Ataque

Cenário #1

Ataques de *Credential Stuffing* utilizando listas de nomes de utilizador/*passwords* conhecidas são bastante comuns. Se uma API não implementa proteções contra ameaças automatizadas ou Credential Stuffing, esta pode ser usada como oráculo para identificar se as credenciais são válidas.

API2:2019 Broken User Authentication

Cenário #2

Um atacante inicia o fluxo de recuperação de *password*, enviando um pedido `POST` com o nome de utilizador para o *endpoint* `/api/system/verification-codes`. Um código de 6 dígitos é enviado para o telefone da vítima. Porque a API não implementa uma política de limitação do número de pedidos, com recurso a um *script multi-thread* que envia as combinações possíveis para o *endpoint* `/api/system/verification-codes/{smsToken}`, o atacante consegue em poucos minutos descobrir o código enviado na SMS.

Como Prevenir

- Certifique-se de que conhece todos os fluxos de autenticação possíveis (e.g. móvel/web/*deeplinks*/etc.).
- Pergunte aos engenheiros responsáveis quais os fluxos em falta/não identificados.
- Leia sobre os mecanismos de autenticação em uso. Certifique-se que compreende quais e como são usados. OAuth não é um mecanismo de autenticação, assim como também não o são as API *keys*.
- Não reinvente a roda em termos de autenticação, geração de *tokens*, armazenamento de *passwords*. Opte pela utilização de *standards*.
- *Endpoints* para recuperação de *password* devem ser tratados como os *endpoints* de *login* no que diz respeito à proteção contra ataques de força bruta, limitação do número de pedidos e bloqueio de conta.
- Utilize a OWASP Authentication Cheatsheet.
- Sempre que possível implemente autenticação de múltiplos fatores.
- Implemente mecanismos anti-força bruta para mitigar ataques do tipo *credential stuffing*, dicionário e força bruta nos *endpoints* de autenticação. Este mecanismo deve ter configurações mais restritivas do que para os demais *endpoints* da API.
- Implemente mecanismos de bloqueio de conta/*captcha* para prevenir ataques de força bruta contra utilizadores específicos. Implemente verificação da qualidade/força das passwords.
- As API *keys* não devem ser usadas para autenticação dos utilizadores, mas ao invés para autenticação dos clientes da API.

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	Abuso: 3	Prevalência: 2	Deteção: 2	Técnico: 2	Específico Negócio
Abusar desta falha de segurança é simples e tipicamente passa pela inspeção do tráfego de rede para analisar as respostas da API em busca de dados que não devessem ser devolvidos ao utilizador.		As APIs delegam nos clientes a responsabilidade de filtrar os dados. Uma vez que as APIs são usadas como fonte de dados, com frequência os programadores procuram fazer implementações genéricas sem ter em consideração a relevância dos dados expostos. Regra geral as ferramentas automáticas não conseguem detetar este tipo de vulnerabilidade por ser difícil distinguir dados legítimos retornados pela API doutros sensíveis que não deveriam ser expostos. Esta tarefa exige um profundo conhecimento da aplicação.		*Excessive Data Exposure* tipicamente conduz à exposição de dados sensíveis.	

A API é vulnerável?

Quando a API devolve dados sensíveis aos clientes. Estes dados são normalmente filtrados pelo cliente antes de serem apresentados ao utilizador. Um atacante pode facilmente inspecionar o tráfego de rede e aceder aos dados sensíveis.

Exemplo de Cenários de Ataque

Cenário #1

A equipa de desenvolvimento móvel usa o *endpoint* `/api/articles/{articleId}/comments/{commentId}` na interface de visualização dos artigos para apresentar os detalhes dos comentários. Inspecionando o tráfego de rede da aplicação móvel, um atacante descobre que outros dados sensíveis relacionados com o autor do comentário fazem ainda parte da resposta da API. O método `toJSON()`, pertencente ao modelo `User`, é usado na implementação do *endpoint* para preparar os dados a retornar, o qual inclui informação pessoal.

Cenário #2

Um sistema de vigilância baseado em IoT permite aos utilizadores com perfil de administrador criar outros utilizadores com diferentes permissões. Um administrador cria uma conta de utilizador para um segurança recém-chegado, o qual apenas tem acesso a câmaras específicas instaladas no edifício. A aplicação móvel usada pelo segurança realiza um pedido ao *endpoint* `/api/sites/111/cameras` para obter os dados relativos às câmaras a mostrar na interface. A resposta contém a lista com os detalhes das câmaras no formato `{"id":"xxx","live_access_token":"xxxx-bbbbb","building_id":"yyy"}`. Embora na interface apenas seja possível ver as câmaras às quais o guarda tem acesso, a resposta da API inclui informação sobre todas as câmaras instaladas.

API3:2019 Excessive Data Exposure

Como Prevenir

- Nunca delegar no cliente a responsabilidade de filtrar os dados.
- Rever as respostas da API, certificando-se que apenas incluem dados legítimos.
- Os engenheiros responsáveis devem questionar-se sempre sobre “quem são os consumidores dos dados” antes de exporem um *endpoint.*
- Evitar a utilização de métodos genéricos tais como `to_json()` e `to_string()`. Pelo contrário, escolher uma-a-uma as propriedades que realmente devem ser devolvidas na resposta.
- Classificar a informação sensível e pessoal (PII) que a API armazena e manipula, revendo todos as chamadas à API onde esta informação é devolvida por forma a avaliar se existe algum risco para a segurança.
- Utilizar *schemas* para validar as respostas da API enquanto camada adicional de segurança. Como parte desta abordagem, definir e assegurar a validação das respostas do diferentes métodos HTTP, incluindo erros.

Referências

Externas

- CWE-213: Intentional Information Exposure

API4:2019 Lack of Resources & Rate Limiting

Específico da API	Abuso: 2	Prevalência: 3	Deteção: 3	Técnico: 2	Específico Negócio
Para abusar destas falhas basta realizar pedidos simples à API. Não é necessária autenticação. Múltiplos pedidos concorrentes podem ser realizados por um único computador ou fazendo uso de recursos computacionais na nuvem.		É comum encontrar APIs que não limitam o número de pedidos ou que usam limites desajustados.		O abuso destas falhas pode conduzir à negação de serviço (DoS), deixando a API incapaz de satisfazer outros pedidos ou mesmo indisponível.	

A API é vulnerável?

Os pedidos a uma API consomem recursos tais como largura de banda, processador, memória e armazenamento. A quantidade de recursos necessária para satisfazer um pedido depende essencialmente da informação enviada pelo utilizador e da lógica de negócio implementa pelo *endpoint*. Deve ainda ter-se em consideração que os pedidos de diferentes clientes concorrem entre si por estes recursos. A API é vulnerável se pelo menos um dos seguintes limites não está definido ou foi configurado com um valor desajustado (e.g. demasiado baixo/alto):

- Tempo máximo de execução
- Quantidade máxima de memória alocada
- Número de ficheiros abertos
- Número de processos
- Tamanho do pedido (e.g., *uploads*)
- Número de pedidos por cliente/recurso
- Número de registos por página devolvidos numa única resposta a um pedido

Exemplos de Cenários de Ataque

Cenário #1

Um atacante carrega uma imagem de grandes dimensões realizando um pedido `POST` para o *endpoint* `/api/v1/images`. Após concluir o carregamento a API cria várias miniaturas de diferentes dimensões. Devido à dimensão da imagem carregada a memória disponível é esgotada durante a criação das miniaturas e a API fica indisponível.

Cenário #2

Uma aplicação apresenta uma listagem de utilizadores até ao limite de 200 por página. A lista dos utilizadores é obtida por meio dum pedido ao *endpoint* `/api/users?page=1&size=100`. Um atacante altera o valor do parâmetro `size` para `200000`, causando problemas de performance no servidor de base de dados. Enquanto se verificam estes problemas de performance a API fica indisponível e incapaz de satisfazer pedidos de qualquer utilizador (DoS).

O mesmo cenário pode ser usado para provocar erros do tipo *Integer Overflw* ou *Buffer Overflow.*

API4:2019 Lack of Resources & Rate Limiting

Como Prevenir

- Tecnologias como Docker tornam mais fácil a definição de limites de memória, processador, número de *restarts*, número de ficheiros abertos e processos.
- Limitar o número máximo de pedidos à API, por cliente, dentro dum determinado período de tempo.
- Notificar o cliente quando o limite de pedidos foi excedido, informando o valor desse limite e o tempo restante para poder voltar a realizar novos pedidos.
- Validar de forma adequada os parâmetros enviados na *query string* e corpo do pedido, em particular aqueles que controlam o número de registos a retornar na resposta.
- Definir e forçar um tamanho máximo de dados para todos os parâmetros e dados de entrada, tais como comprimento máximo para os texto ou número máximo de elementos duma lista.

Referências

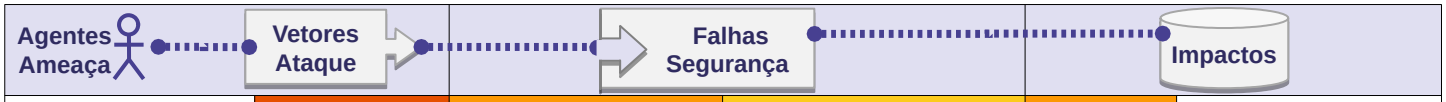
OWASP

- 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	Abuso: 3	Prevalência: 2	Deteção: 1	Técnico: 2	Específico Negócio
Para abusar deste tipo de falha o atacante tem de realizar pedidos legítimos ao *endpoint* da API ao qual não é suposto ter acesso. Estes *endpoints* podem estar disponíveis para utilizadores anónimos, ordinários ou não privilegiados. É fácil identificar estas falhas em APIs uma vez que estas são mais estruturada, sendo a forma de acesso a certas funcionalidades mais previsível (e.g., utilizar o método HTTP `PUT` ao invés de `GET` ou substituir a palavra "*user*" por "*admin*" no URL).		As verificações de autorização para aceder a uma determinada função ou recurso são normalmente geridas por configuração e às vezes ao nível da implementação. A correta implementação destes mecanismos pode tornar-se confusa, uma vez que, as aplicações modernas prevêem vários perfis ou grupos de utilizador, assim como complexos esquemas de hierarquia (e.g., sub-utilizadores, utilizadores com mais do que um perfil).		Estas falhas permitem aos atacantes aceder de forma não autorizada a certas funcionalidades. As funcionalidades administrativas são o alvo preferencial neste tipo de ataque.	

A API é vulnerável?

A melhor forma de identificar falhas de verificação de autorização de acesso a funções é através duma análise detalhada do mecanismo de autorização, devendo ter-se em consideração o esquema de hierarquia de utilizadores, diferentes perfis ou grupos e questionando continuamente:

- Utilizadores ordinários podem aceder aos *endpoints* de administração?
- Os utilizadores podem realizar ações sensíveis (e.g. criar, modificar ou apagar) para as quais não deveriam ter acesso, alterando simplesmente o método HTTP (e.g. alterando de `GET` para `DELETE`)?
- Um utilizador do grupo X pode aceder a uma função reservada ao grupo Y, adivinhando o URL do *endpoint* e os parâmetros (e.g. `/api/v1/users/export_all`)?

Nunca assuma o tipo dum *endpoint*, normal ou administrativo, apenas com base no URL.

Apesar dos programadores poderem ter decidido expor a maioria dos *endpoints* administrativos sob um mesmo prefixo, como por exemplo `api/admins`, é comum encontrarem-se *endpoints* administrativos sob outros prefixos, misturados com *endpoints* ditos ordinários e.g. `api/users`.

Exemplos de Cenários de Ataque

Cenário #1

Durante o processo de registo numa aplicação que apenas permite o registo de utilizadores por convite, é realizado um pedido `GET` ao *endpoint* `/api/invites/{invite_guid}`. A resposta em formato JSON contém detalhes sobre o convite, incluindo o perfil de utilizador e o seu endereço de email.

Um atacante duplica o pedido e altera o método HTTP e o *endpoint* do medido para `POST /api/invites/new`. Este *endpoint* deveria estar apenas disponível para administradores através da consola de administração, uma vez que não implementa verificações de autorização de acesso à função.

O atacante abusa da falha e envia para si próprio um convite para criar uma conta de administrador:

API5:2019 Broken Function Level Authorization

```
POST /api/invites/new
{“email”:”hugo@malicious.com”,”role”:”admin”}
```

Cenário #2

Uma API implementa um *endpoint* que é suposto estar apenas disponível para administradores - `GET /api/admin/v1/users/all`. Este *endpoint* devolve os detalhes de todos os utilizadores da aplicação e não realiza qualquer verificação de autorização de acesso à função. Com base no conhecimento adquirido sobre a estrutura da API um atacante consegue prever com um elevado grau de certeza o URL do *endpoint* que expõe estes dados sensíveis sobre os utilizadores da aplicação.

Como Prevenir

A sua API deve usar um módulo de autorização consistente e fácil de analisar, o qual deve ser invocado por todas as funções de negócio. Frequentemente, este tipo de proteção é oferecido por um ou mais componentes externos à lógica aplicacional.

- Por omissão todos os acesso devem ser negados, exigindo que permissões específicas sejam concedidas a perfis específicos para acesso a cada função.
- Rever todos os *endpoints* à procura de falhas ao nível da verificação de autorização de acesso a funções, tendo sempre em consideração a lógica de negócio da aplicação e hierarquia dos grupos.
- Assegurar que todos os controladores administrativos herdam dum controlador administrativo base que implementa as verificações de autorização com base no grupo/perfil do utilizador.
- Assegurar que funções administrativas num controlador ordinário implementam elas próprias as verificações de autorização baseadas no grupo e perfil do utilizador.

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

Específico da API	Abuso: 2	Prevalência: 2	Deteção: 2	Técnico: 2	Específico Negócio
O abuso requer na maioria das vezes um conhecimento da lógica de negócio, de relações entre objetos, e da estrutura da API. Abusar de Mass Assignment é mais fácil em APIs, visto que por design elas expõem a implementação interna da aplicação e nomes de propriedades.		*As frameworks* modernas incentivam os programadores a usar funções que ligam automaticamente dados do cliente a variáveis no código e objetos internos. Os atacantes podem usar esta metodologia para atualizar ou mudar propriedades sensíveis de objetos que os programadores não pretendiam realmente expor.		O abuso pode levar a elevação de privilégios, manipulação de dados, contornar mecanismos de segurança, etc.	

A API é vulnerável?

Os objetos em aplicações modernas podem conter muitas propriedades. Algumas destas propriedades podem ser atualizadas diretamente pelo cliente (e.g., `user.first_name` ou `user.address`) mas outras não (e.g., a *flag* `user.is_vip`).

Um *endpoint* é vulnerável se converter automaticamente parâmetros do cliente em propriedades internas de um objeto, sem considerar a sensibilidade e o nível de exposição destas propriedades. Isto pode permitir a um atacante atualizar propriedades de objetos, às quais ele não deveria ter acesso.

Exemplos de propriedades sensíveis:

- **Propriedades relacionadas com permissões**: `user.is_admin`, `user.is_vip` devem ser modificadas apenas por administradores.
- **Propriedades dependentes de processos**: `user.cash` deve ser modificada apenas internamente depois da verificação de pagamento.
- **Propriedades internas**: `article.created_time` deve ser modificada apenas internamente pela aplicação.

Exemplos de Cenários de Ataque

Cenário #1

Uma aplicação de partilha de transporte tem ao dispor do utilizador uma opção para editar informações básicas para o seu perfil. Durante este processo, um pedido à API é enviado para `PUT /api/v1/users/me` com o seguinte objeto JSON legítimo:

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

O pedido `GET /api/v1/users/me` incluí uma propriedade `credit_balance` adicional:

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

O atacante envia novamente o primeiro pedido com o seguinte conteúdo:

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

Dado que o *endpoint* é vulnerável a Mass Assignment, o atacante recebe crédito sem ter efetuado qualquer pagamento.

API6:2019 Mass Assignment

Cenário #2

Um portal de partilha de vídeo permite carregar e descarregar conteúdo em diferentes formatos. Um atacante que investigou a API descobriu que o *endpoint* `GET /api/v1/videos/{video_id}/meta_data` devolve um objeto JSON com as propriedades do vídeo. Uma das propriedades é `"mp4_conversion_params":"-v codec h264",` que revela que a aplicação utiliza um comando *shell* para converter o vídeo.

O atacante também descobriu o *endpoint* `POST /api/v1/videos/new` que é vulnerável a Mass Assignment, permitindo ao cliente modificar qualquer propriedade do objeto. O atacante atribui um valor malicioso como o seguinte: `"mp4_conversion_params":"-v codec h264 && format C:/"`. Este valor vai causar uma injeção de um comando *shell* assim que o atacante descarregar o vídeo no formato MP4.

Como Prevenir

- Se possível, evitar usar funções que convertem automaticamente parâmetros do cliente em variáveis de código ou objetos internos.
- Ter uma lista onde constam apenas os nomes das propriedades que o cliente pode atualizar.
- Usar funcionalidades já existentes para ter uma lista de propriedades que não devem ser acedidas por clientes.
- Se possível, definir explicitamente e forçar utilização de *schemas* para o conteúdo dos pedidos.

Referências

Externas

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

API7:2019 Security Misconfiguration

Específico da API	Abuso: 3	Prevalência: 3	Deteção: 3	Técnico: 2	Específico Negócio
Os atacantes vão regra geral tentar encontrar falhas, *endpoints* comuns ou ficheiros e diretórios desprotegidos para ganhar acesso não autorizado ou conhecimento do sistema.		Falhas nas configurações de segurança podem acontecer em qualquer camada da API, desde o nível de rede ao aplicacional. Existem ferramentas automáticas para detetar e abusar destas falhas, tais como serviços desnecessários que se encontram em execução ou configurações antigas.		Estas falhas podem não só expor dados sensíveis dos utilizadores, mas também detalhes do sistema que permitam comprometer o servidor.	

A API é vulnerável?

A API é vulnerável se:

- As configurações para proteger o sistema estão em falta em qualquer das partes constituintes da API, ou se existem serviços na nuvem indevidamente configurados.
- As últimas atualizações de segurança não foram aplicadas ou os sistemas estão desatualizados.
- Existem funcionalidades ativas que não estão em uso (e.g., verbos HTTP).
- *Transport Layer Security* (TLS) não está configurado.
- Diretivas de segurança não são enviadas aos clientes (e.g., cabeçalhos HTTP de segurança).
- Não existe um política de Partilha de Recursos entre Origens (CORS) ou esta está indevidamente configurada.
- As mensagens de erro incluem *stack traces* ou outra informação sensível.

Exemplos de Cenários de Ataque

Cenário #1

Um atacante encontra o ficheiro `.bash_history` na diretoria raiz do servidor, o qual contem os comandos usados pela equipa de DevOps para aceder à API:

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

O atacante pôde assim identificar novos *endpoints* da API, destinados exclusivamente ao uso pela equipa de DevOps e que não estão documentados.

Cenário #2

Tendo em vista um serviço específico, um atacante usa um conhecido motor de busca de dispositivos diretamente acessíveis através da internet. O atacante encontrou um *host* a correr um conhecido sistema de gestão de base de dados, à escuta na porta padrão. Como o *host* estava a utilizar a configuração padrão, a qual tem o mecanismo de autenticação desativado por omissão, o atacante teve acesso a milhões de registo com informação pessoal (PII), preferências e dados de autenticação dos utilizadores.

Cenário #3

Inspecionando o tráfego de rede duma aplicação móvel, um atacante percebe que nem todo o tráfego usa um protocolo seguro (e.g., TLS), em particular aquele associado às imagens de perfil de utilizador. Como as interações do utilizador na aplicação são binárias, apesar do tráfego da API ser enviado de forma segura, o atacante identifica um padrão no tamanho das respostas da API, o qual usa para mapear as preferências do utilizador em relação ao conteúdo visualizado (e.g., imagens de perfil).

API7:2019 Security Misconfiguration

Como Prevenir

O ciclo de vida da API deve incluir:

- Um processo de proteção reprodutível que possa ser implantado de forma fácil e rápida com vista a um ambiente de execução devidamente protegido.
- Um processo de revisão e atualização de todas as camadas da API. A revisão deve incluir: ficheiros de orquestração, componentes da API e serviços na nuvem (e.g., permissões dos *buckets* S3).
- Um canal de comunicação seguro para todas as interações da API no acesso a recursos estáticos (e.g., imagens).
- Um processo automatizado para verificar de forma continua as configurações e definições em todos os ambientes (produção, *staging*, testes, desenvolvimento).

E ainda:

- Para prevenir que *stack traces* sejam incluídas nas mensagens de erro ou outra informação sensível seja fornecida aos atacantes, quando aplicável, defina *schemas* e verifique que todas as respostas da API estão em conformidade.
- Assegure que a API só é acessível através do verbos HTTP especificados. Todos os demais verbos HTTP que não são utilizados deverão estar desativados (e.g., `HEAD`).
- As APIs destinadas a acessos por clientes a correr em navegadores (e.g., WebApps) devem implementar uma política de Partilha de Recursos Entre Origens (CORS) adequada.

Referências

OWASP

- OWASP Secure Headers Project
- OWASP Testing Guide: Configuration and Deployment Management
- OWASP Testing Guide: Testing for Error Handling
- 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	Abuso: 3	Prevalência: 2	Deteção: 3	Técnico: 3	Específico Negócio
Os atacantes fornecem à API dados maliciosos através dos vetores de injeção disponíveis (e.g., formulários, parâmetros, integrações, etc.), na expetativa que estes sejam enviados a um interpretador.		Falhas de injeção são bastante comuns e são geralmente encontradas em consultas SQL, LDAP ou NoSQL, comandos enviados ao sistema operativo, *parsers* XML e ORM. Estas falhas são fáceis de identificar durante a revisão do código. Os atacantes podem fazer uso de *scanners* e *fuzzers*.		As injeções podem conduzir à divulgação de informação e perda de dados. Podem ainda conduzir à negação de serviço (DoS) ou à tomada de controlo do *host*.	

A API é vulnerável?

A API é vulnerável se:

- Dados fornecidos pelo cliente não são validados, filtrados ou sanitizados pela API.
- Dados fornecidos pelo cliente são concatenados diretamente em consultas SQL/NoSQL/LDAP, comandos a enviar ao sistema operativo, *parsers* XML e *Object Relational Mapping* (ORM)/*Object Document Mapper* (ODM).
- Dados com origem em sistemas externos (e.g., sistemas integrados) não são validados, filtrados ou sanitizados pela API.

Exemplos de Cenários de Ataque

Cenário #1

O *firmware* dum dispositivo de controlo parental implementa o *endpoint* `/api/CONFIG/restore`, o qual espera que lhe seja enviado uma parâmetro `appId` no formato `multipart`. Com recurso a um descompilador, um atacante descobre que o parâmetro `appId` é passado diretamente numa chamada ao sistema sem qualquer tipo de sanitização:

```
snprintf(cmd, 128, "%srestore_backup.sh /tmp/postfile.bin %s %d",
         "/mnt/shares/usr/bin/scripts/", appid, 66);
system(cmd);
```

O comando abaixo permite ao atacante desligar qualquer equipamento que corra o mesmo *firmware* vulnerável:

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

Cenário #2

Perante uma aplicação de reservas com funcionalidades de criação, consulta, atualização e remoção, um atacante desconfia que talvez seja possível realizar injeções `NoSQL` através do parâmetro `bookingId` presente na *query string* dos pedidos de remoção. Este é o aspeto do pedido: `DELETE /api/bookings?bookingId=678`.

Esta é a função usada pelo servidor da API para atender tais pedidos:

```
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'
    });
  }
});
```

API8:2019 Injection

O atacante interceta o pedido e altera o parâmetro `bookingId` na *query string*, conforme apresentado abaixo. Neste caso o atacante consegue apagar a reserva doutro utilizador.

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

Como Prevenir

A prevenção de injeções exige que os dados sejam separados dos comandos e consultas.

- Usar uma única biblioteca para validação de dados que seja confiável e ativamente mantida.
- Validar, filtrar e sanitizar todos os dados fornecidos pelo cliente ou outros dados provenientes de sistemas integrados.
- Caracteres especiais devem ser neutralizados com recurso à sintaxe específica do interpretador para onde serão enviados.
- Opte por APIs de consulta seguras que oferecem interfaces parametrizadas.
- Limite sempre o número de registos a devolver por forma a prevenir a divulgação massiva de dados em caso de injeção.
- Valide os dados de entrada usando os filtros necessários para apenas permitir valores válidos para cada parâmetro.
- Defina tipos de dados e padrões bem definidos para todos os parâmetros textuais.

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	Abuso: 3	Prevalência: 3	Deteção: 2	Técnico: 2	Específico Negócio
Versões antigas da API tipicamente não são alvo de atualizações e podem ser usadas para comprometer sistemas sem ter que lidar com mecanismos de segurança mais avançados, os quais poderão estar ativos nas versões mais recentes.		A documentação desatualizada dificulta a identificação e/ou correção de falhas de segurança. A inexistência dum inventário e duma estratégica de descontinuação estão na génese dos sistemas sem atualizações de segurança que acabam por divulgar informação sensível. É comum encontrar-se APIs expostas desnecessariamente: o conceito de micro-serviços tornou o *deploy* das aplicações mais fácil e independente (e.g., *cloud*, kubernetes), podendo estar na origem deste fenómeno.		Os atacantes podem conseguir acesso a informação sensível ou até obter o controlo do servidor através de versões antigas e sem atualizações de segurança que estejam ligadas à mesma base de dados.	

A API é vulnerável?

A API pode ser vulnerável se:

- O propósito dum *host* da API não é claro, não havendo respostas explicitas para as seguintes perguntas:
  - Em que ambientes está a API a correr (e.g., produção, *staging*, testes, desenvolvimento)?
  - Quem deve ter acesso à API através da rede (e.g., público, interno, parceiros)?
  - Que versões da API estão a correr?
  - Que informação é recolhida e processada pela API (e.g., PII)?
  - Qual é o fluxo dos dados?
- Não existe documentação, ou a que existe não está atualizada.
- Não existe um plano para descontinuar cada uma das versões da API.
- Não existe um inventário de *hosts* ou o que existe está desatualizado.
- O inventário de integração de serviços, próprios ou de terceiros, não existe ou está desatualizado.
- Versões antigas ou anteriores estão a correr sem atualizações de segurança.

Exemplos de Cenários de Ataque

Cenário #1

Depois de re-desenhar as suas aplicações, um serviço de pesquisa local deixou uma versão antiga da API a correr (`api.someservice.com/v1`), desprotegida e com acesso à base de dados de utilizadores. Enquanto estava a analisar uma das últimas versões das aplicações, um atacante encontrou o endereço da API (`api.someservice.com/v2`). Substituindo `v2` por `v1` no URL, o atacante conseguiu acesso à versão antiga da API a qual expunha informação pessoal (PII) de mais de 100 milhões de utilizadores.

Cenário #2

Uma rede social implementou um mecanismo de limitação do número de pedidos para impedir os atacantes de usar ataques de força bruta para adivinhar os *tokens* de redefinição de *password*. Este mecanismo não foi implementado ao nível do código da API, mas sim como um componente entre o cliente e a API em uso (`www.socialnetwork.com`). Um investigador encontrou um *host* relativo à versão beta da API mas que corria agora a última versão desta, incluindo o mecanismo de redefinição da *password*, mas aqui sem o mecanismo de limitação do número de pedidos. O investigador seria capaz de redefinir a *password* de qualquer utilizador, recorrendo a força bruta para adivinhar o *token* de 6 dígitos.

API9:2019 Improper Assets Management

Como Prevenir

- Inventarie todos os *hosts* da API e documente os aspetos importantes de cada um deles, com especial enfoque no ambiente da API (e.g., produção, *staging*, testes, desenvolvimento), quem deve ter acesso pela rede ao *host* (e.g., público, interno, parceiros) e a versão da API.
- Inventarie as integrações de serviços e documente os aspetos mais importantes tais como o papel destes no sistema, que dados são trocados (fluxo de dados) e a sua suscetibilidade.
- Documente todos os aspetos da API, tais como autenticação, erros, redirecionamentos, política de limitação do número de pedidos, polícia de Partilha de Recursos Entre Origens (CORS) e *endpoints*, incluindo os seus parâmetros, pedidos e respostas.
- Gere a documentação de forma automática através da adoção de *standards*. Inclua a geração da documentação no seu processo de CI/CD.
- Torne a documentação da API disponível para aqueles autorizados a consultá-la.
- Utilize mecanismos de proteção externa, tais como *API Security Firewalls*, em todas as versões da API expostas e não exclusivamente a versão mais recente em produção.
- Evite a utilização de dados de produção em outros ambientes da API que não de produção. Se não puder evitá-lo, esses ambientes/versões/*endpoints* deverão ter o mesmo nível de segurança do que os de produção.
- Quando as novas versões da API incluem melhorias de segurança, realize a análise de risco para uma melhor tomada de decisão quanto às ações necessárias para a migração das versões antigas: por exemplo, se é possível aplicar as mesmas melhorias às versões anteriores sem quebrar compatibilidade ou se as deve retirar o quanto antes, forçando os clientes a migrar para a última versão.

Referências

Externas

- CWE-1059: Incomplete Documentation
- OpenAPI Initiative

API10:2019 Insufficient Logging & Monitoring

Específico da API	Abuso: 2	Prevalência: 3	Deteção: 1	Técnico: 2	Específico Negócio
Os atacantes podem tirar partido desta vulnerabilidade para abusar dos sistemas sem ser detetados.		Sem registo de eventos e monitorização, ou com um registo e monitorização insuficientes, é quase impossível detetar atividades suspeitas e responder às mesmas em tempo útil.		Sem visibilidade sobre atividades maliciosas os atacantes têm tempo suficiente para comprometer os sistemas.	

A API é vulnerável?

A API é vulnerável se:

- Não regista qualquer evento, o tipo de evento registado não é o correto ou os registos não incluem detalhe suficiente.
- A integridade do registo não é assegurada (e.g., Log Injection).
- Os registos não são monitorizados.
- A infraestrutura da API não é monitorizada ininterruptamente.

Exemplos de Cenários de Ataque

Cenário #1

As *access keys* duma API de administração foram expostas publicamente num repositório. O proprietário do repositório foi notificado por email sobre a potencial divulgação das chaves mas demorou mais de 48h a reagir ao incidente e a exposição das chaves pode ter permitido o acesso a informação sensível. Devido ao registo de eventos insuficiente, a empresa não foi capaz de averiguar que informação havia sido acedida por agentes mal intencionados.

Cenário #2

Uma plataforma de partilha de vídeo foi alvo um ataque de *credendial stuffing* em larga escala. Apesar das tentativas de autenticação falhadas constarem do registo de eventos, nenhum alerta foi gerado durante o tempo que o ataque decorreu. Em reação às queixas dos utilizadores os registos de eventos da API foram analisados e o ataque foi identificado. A empresa teve que emitir um comunicado público a pedir aos utilizadores para alterarem as suas *passwords* e comunicar o incidente às autoridades reguladores.

API10:2019 Insufficient Logging & Monitoring

Como Prevenir

- Registe todas as tentativas de autenticação falhadas, controlo de acesso negados e falhas na validação de dados fornecidos pelo utilizador.
- Os registos de eventos devem usar um formato que permita serem processados por ferramentas de gestão de registos e devem incluir detalhe suficiente para identificar os agentes maliciosos.
- Os registos de eventos deve ser tratados como informação sensível e a sua integridade deve ser assegurada tanto em repouso como em trânsito.
- Configure um sistema de monitorização para a infraestrutura, rede e API.
- Utilize um Sistema de Gestão e Correlação de Eventos de Segurança (SIEM) para agregar e gerir os registos de eventos de todos os componentes da API e *hosts*.
- Configure visualizações personalizadas sobre os alertas, permitindo que atividade suspeita seja detetada e endereçada o mais cedo possível.

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 O Que Se Segue Para Programadores

A tarefa de criar e manter software seguro, ou corrigir software existente, pode ser difícil. O mesmo se verifica em relação às APIs.

Acreditamos que educação e consciencialização são fatores chave para o desenvolvimento de software seguro. Tudo o mais necessário para alcançar este objetivo depende da **definição e utilização de processos de segurança reprodutíveis e do uso de controlos de segurança *standard***.

A OWASP disponibiliza uma grande quantidade de recursos gratuitos e abertos para abordar a segurança desde o início dum projeto. Por favor visite a página dos projetos OWASP para consulta da lista dos projetos existentes.

Educação	Pode começar por ler os conteúdos disponibilizados pelos projetos na categoria OWASP Education de acordo com a sua profissão e interesse. Para uma abordagem mais prática, adicionámos ao nosso plano de trabalho o projeto **crAPI** – **C**ompletely **R**idiculous **API** (API Completamente Ridícula). Enquanto isso, pode praticar segurança de aplicações web usando o Módulo OWASP DevSlop Pixi: uma WebApp e API intencionalmente vulneráveis com o objetivo de ensinar aos utilizadores como testar a segurança de WebApps modernas e APIs e como desenvolver APIs mais seguras. Poderá também participar nas sessões de treino das conferências OWASP AppSec ou juntar-se ao seu grupo OWASP local.
Requisitos de Segurança	A segurança deve fazer parte de qualquer projeto desde o início. É importante que, durante a fase de identificação de requisitos, seja definido o que é que "seguro" significa no contexto desse projeto. A OWASP recomenda a utilização do OWASP Application Security Verification Standard (ASVS) como guia para definir os requisitos de segurança. Se estiver a subcontratar, considere ao invés a utilização do OWASP Secure Software Contract Annex, o qual deverá adaptar às leis e regulamentações locais.
Arquitetura de Segurança	A segurança deve ser uma preocupação durante todas as fases dum projeto. O projeto OWASP Prevention Cheat Sheets é um bom ponto inicial de orientação sobre como contemplar a segurança durante a fase de arquitetura. Entre outros, o REST Security Cheat Sheet e o REST Assessment Cheat Sheet serão seguramente relevantes.
Controlos Standard de Segurança	A adoção de controlos standard de segurança reduzem o risco de introdução de falhas de segurança durante a implementação da lógica de negócio. Apesar de muitas *frameworks* modernas já incluírem controlos *standards*, o projeto OWASP Proactive Controls dá-lhe uma visão sobre que controlos de segurança deve incluir no seu projeto. A OWASP também disponibiliza algumas bibliotecas e ferramentas que pode achar úteis, tais como controlos de validação.
Ciclo de Desenvolvimento de Software Seguro	Pode usar o OWASP Software Assurance Maturity Model (SAMM) para melhorar o processo de desenvolvimento de APIs. Tem ainda disponíveis vários outros projetos OWASP para o ajudar durante as várias fases de desenvolvimento de APIs, por exemplo o OWASP Code Review Project.

+DSO O que Se Segue Para DevSecOps

Dada a sua importância na arquitetura das aplicações modernas, desenvolver APIs seguras é crucial. A segurança não pode ser negligenciada e deve estar presente durante todo o clico de vida do desenvolvimento. Já não basta a execução de *scanners* ou a realização de testes de penetração anualmente.

A equipa de DevSecOps deve fazer parte do esforço de desenvolvimento contribuindo para a realização de testes de segurança, de forma continuada, durante todo o ciclo de vida do desenvolvimento. Deve ter como objetivo melhorar a *pipeline* de desenvolvimento com automação de segurança e sem influenciar negativamente o ritmo do desenvolvimento.

Em caso de dúvida mantenha-se informado e reveja o Manifesto DevSecOps com frequência.

Compreenda o Modelo de Ameaças	As prioridades relativamente ao que deve ser testado têm origem no modelo de ameaças. Se não tem um, considere usar o OWASP Application Security Verification Standard (ASVS) e o OWASP Testing Guide como base. Envolver a equipa de desenvolvimento na elaboração do modelo de ameaças pode torná-la mais consciente para questões relacionadas com segurança.
Compreenda o Ciclo de Vida do Desenvolvimento do Software	Reúna a equipa de desenvolvimento para melhor compreender o ciclo de vida do desenvolvimento do software. O seu contributo para a realização continua de testes de segurança deve ser compatível com as pessoas, processos e ferramentas. Todos devem concordar com o processo, de forma a não provocar atrito ou resistência desnecessários.
Estratégias de Teste	Sendo que o seu trabalho não deverá condicionar o ritmo de desenvolvimento, deve escolher cuidadosamente a melhor (mais simples, rápida e precisa) técnica para verificar os requisitos de segurança. A OWASP Security Knowledge Framework e o OWASP Application Security Verification Standard podem ser importantes fontes de requisitos de segurança funcionais e não-funcionais. Existem outras fontes relevantes onde poderá encontrar projetos e ferramentas como aquelas disponibilizadas pela comunidade DevSecOps.
Procure Alcançar Cobertura e Precisão	Você é a ponte entre as equipas de desenvolvimento e operações. Para alcançar cobertura, deve não só focar-se na funcionalidade, mas também na orquestração. Trabalhe junto de ambas as equipas desde o início por forma a otimizar o seu tempo e esforço. Deve almejar um estado em que o essencial da segurança é verificado de forma continua.
Comunique as Falhas de Forma Clara	Entregue valor evitando qualquer atrito. Comunique as falhas identificadas atempadamente, usando as ferramentas que a equipa de desenvolvimento já utiliza (e não através de ficheiros PDF). Junte-se à equipa de desenvolvimento para resolver as falhas identificadas. Aproveite a oportunidade para educar os elementos da equipa de desenvolvimento, descrevendo de forma clara a falha e como esta pode ser abusada, incluindo um cenário de ataque para a tornar mais real.

+DAT Metodologia e Dados

Preâmbulo

Uma vez que a indústria de segurança aplicacional não tem estado focada especificamente nas arquiteturas aplicacionais mais recentes, nas quais as APIs têm um papel importante, compilar a lista dos riscos de segurança mais críticos para APIs com base numa consulta pública de dados teria sido uma tarefa árdua. Apesar desta consulta pública de dados não ter sido feita, a lista atual é ainda resultado de informação que se encontra disponível publicamente, assim como de contribuições de especialista em segurança e da discussão aberta à comunidade de segurança.

Metodologia e Dados

Numa primeira fase um grupo de especialistas em segurança recolheu, reviu e categorizou informação relativa a incidentes relacionados com APIs que se encontrava disponível publicamente. Esta informação foi recolhida de plataformas de *bug bounty* e bases de dados de falhas de segurança, restringida a incidentes ocorridos no último ano. Esta informação foi usada para fins estatísticos.

Na fase seguinte, foi pedido a um grupo de profissionais de segurança com experiência em testes de penetração que criassem a seu próprio Top 10.

A Metodologia de Classificação de Risco da OWASP foi usada para realizar a análise de risco e as classificações foram discutidas e revistas entre os profissionais de segurança. Para mais informação sobre este assunto consulte a secção Riscos de Segurança em APIs.

O primeiro rascunho do OWASP API Security Top 10 2019 resultou do consenso entre os dados estatísticos da primeira fase e as listas compiladas pelos profissionais de segurança. Este rascunho foi depois submetido à apreciação e revisão por outro grupo de profissionais de segurança com experiência relevante em segurança de APIs.

O OWASP API Security Top 10 2019 foi apresentado publicamente pela primeira vez na conferência OWASP Global AppSec Tel Aviv (Maio 2019). Desde então ele tem estado disponível no GitHub para discussão e contribuições.

A lista de todos os que contribuíram para esta versão encontra-se na secção Agradecimentos.

+ACK Agradecimentos

Agradecimento ao Contribuidores

Gostaríamos de agradecer às pessoas abaixo, as quais contribuíram publicamente no GitHub ou por 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