Curso de Extensão

Tecnologias Microsoft

UNICAMP

INF-0998

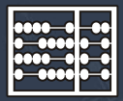
Programação segura (Segurança de software) Aula 4

Prof. Dr. Alexandre Braga, CISSP, CSSLP, PMP

alexbraga@ic.unicamp.br / ambraga@cpqd.com.br / braga.alexandre.m@gmail.com

br.linkedin.com/in/alexmbraga

3 de Dezembro de 2022



INSTITUTO DE COMPUTAÇÃO

Falhas de projeto inseguro de software

Falhas de projeto de software

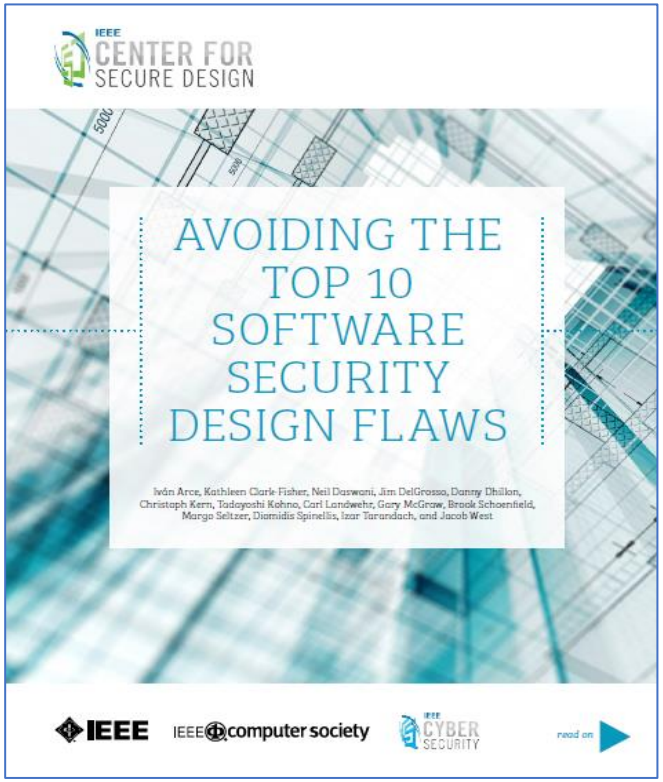
- Muitos problemas de segurança de software podem ser identificados e resolvidos antes da codificação

- Há diversas falhas de segurança comuns em projetos de software que podem ser evitadas antecipadamente (antes da codificação e da implantação)

Top 10 Secure Design Flaws

As dez falhas de projeto de software seguro e como evita-las

1. Ganhar ou dar confiança, mas nunca suponho que ela existe
2. O mecanismo de autenticação não pode ser desviado nem adulterado
3. Autorizar somente após autenticar
4. Separar controle e dados, não executar comandos não confiáveis
5. Garantir que todos os dados são validados explicitamente
6. Usar criptografia corretamente
7. Identificar os dados sensíveis e como eles devem ser mantidos
8. Sempre considerar a criatividade do usuário final
9. Os componentes externos mudam a superfície de ataque
10. Prever alterações futuras em componentes, objetos e atores



Fonte: https://ieeecs-media.computer.org/media/technical-activities/CYBSI/docs/Top-10-Flaws.pdf

DF-01: Há confiança implícita

Ganhe ou dê, nunca suponha que existe confiança

Colocar no cliente as funções de segurança do servidor expõe as funções ao ambiente menos confiável

Designs que colocam essas funções de segurança no cliente são inerentemente fracos e inseguros

- **autorização, controle de acesso, verificação de regras da política de segurança, dados confidenciais**

Exemplos de clientes pouco confiáveis:

- **web browser**
- **Thick-client**
- **mobile device**
- **embedded software**
- **chamadas de APIs por parceiros**

Dados enviados por clientes não confiáveis são considerados comprometidos até prova do contrário

- **Devem ser validados antes de usados**

Essa falha de design acontece quando projetistas fazem o seguinte (incorretamente):

- **Assumem que APIs do servidor sempre serão chamadas na mesma ordem**
- **Acreditam que a interface do usuário é sempre capaz de restringir o que o usuário é capaz de enviar para o servidor**
- **Tentam construir a lógica de negócios ou validações exclusivamente no lado do cliente**
- **Tentam armazenar segredos no cliente, mesmo com criptografia forte**

CWE-300: Channel Accessible by Non-Endpoint https://cwe.mitre.org/data/definitions/300.html
CWE-602: Client-Side Enforcement of Server-Side Security https://cwe.mitre.org/data/definitions/602.html

DF-02: Falta de confiança na autenticação

Mecanismo de autenticação não pode ser desviado nem adulterado

- **Autenticação é o ato de validar a identidade de algo ou alguém**
- **Um software seguro deve impedir que um usuário tenha acesso ao sistema/serviço sem ser autenticado**
- **Depois que o usuário é autenticado, o sistema seguro também deve impedir que o usuário altere sua identidade sem nova autenticação**
- **Em geral, um sistema deve considerar a força da autenticação que um usuário forneceu antes de agir (senha, token, biometria, duplo fator)**
- **Exemplo: autenticação por token de sessão mantido em cookie poder ser suficiente em alguns casos, mas não em outras**

Exemplo: um sistema que possui autenticação é vulnerável ao desvio de autenticação quando

- **permite que um usuário acesse o serviço / API diretamente por uma URL "escondida",**
- **sem exigir uma credencial de autenticação**

É preferível ter um único método, componente ou sistema responsável pela autenticação de usuários

- **Evitar inconsistências nas bases de usuários**
- **Tal mecanismo único pode servir como um gargalo lógico que não pode ser contornado**
- **Uma vez que um único mecanismo tenha sido escolhido, deve ser utilizado em todas autenticações**

CWE Authentication Errors https://cwe.mitre.org/data/definitions/1211.html

DF-03: É Autenticação ou Autorização?

Autorizar o usuário somente após autenticar a identidade dele

A autorização sempre deve ser feita por verificação explícita da permissão, mesmo depois que uma autenticação acabou de ser realizada

A autorização depende dos privilégios associados ao usuário autenticado e também do contexto da solicitação

- **Horário, lugar, equipamento, etc.**

Às vezes, a autorização de um usuário para um sistema ou serviço precisa ser revogada

- **Se a autorização não permitir a revogação, o sistema é vulnerável aos abusos de usuários com autorizações desatualizadas**

Para operações confidenciais, sensíveis ou críticas, a autorização pode exigir a reautenticação do usuário

Autenticação não é binária

- **Usuários podem ser obrigados a apresentar evidências mínimas (mais fortes) de sua identidade**

Autenticação geralmente não é contínua

- **Um usuário pode ser autenticado, mas se afastar do dispositivo ou entregá-lo para outra pessoa**

A autorização de uma operação sensível pode exigir reautenticação ou autenticação mais forte

Políticas podem exigir duas ou mais autorizações em ações críticas (separação de responsabilidades)

Uma infraestrutura comum (um único mecanismo) deve ser usada para as verificações de autorização

CWE Authorization Errors https://cwe.mitre.org/data/definitions/1212.html

DF-04: Misturar controle e dados

Separar comandos de controle e dados da aplicação

Misturar dados e instruções de controle em uma única estrutura (p.ex. strings em requisição http), pode levar a vulnerabilidades de injeção

Falta de separação estrita entre dados e código geralmente leva a dados não confiáveis

- **O atacante manipula os dados para controlar o fluxo de execução de uma aplicação**

No baixo nível, a mistura entre dados e controle pode resultar na corrupção de memória

- **Pode levar ataques de DoS e data leaks**

No alto nível, a mistura de controle e dados pode ocorrer na interpretação de comandos em tempo de execução (runtime) das linguagens de programação

Contexto: software monta strings combinando dados não confiáveis com instruções de controle confiáveis

- **Vulnerabilidades de injeção surgem se os dados não confiáveis não são validados e nem jogados fora**

APIs propensas a injeção de comandos possuem risco alto de exploração da vulnerabilidade de injeção

- **Exemplos de tais vulnerabilidades incluem injeção de SQL, injeção de JavaScript (p.ex. XSS), injeção de XML e injeção de comando de shell**

Software que ignora o princípio da separação estrita entre dados e código, sem separação explícita entre dados e instruções, são inerentemente inseguros

CWE-94: Improper Control of Generation of Code ('Code Injection')
https://cwe.mitre.org/data/definitions/94.html

DF-04 (2): Comandos não confiáveis

Nunca execute comandos de fontes não confiáveis

Nas APIs, evite métodos / endpoints que consumam strings com controle e dados ao mesmo tempo

Prefira expor métodos ou endpoints que consumam tipos estruturados (com segregação estrita entre dados e comandos de controle)

- **Tipos fortes: inteiros, booleanos, alfabéticos, etc.**

Nos aplicativos, evite APIs que misturam dados e flags de controle em parâmetros de strings

Um app deve usar uma API legada (propensa a injeção) por meio de uma API interna (um proxy ou wrapper de API) com segregação de dados e controle

Aplicativos que transformam dados em códigos

- **validar os dados com cuidado e restringir ações executadas com os comandos gerados**

Comandos eval ou exec

- **Nas linguagens interpretadas: Java (exec), Python, Ruby, JavaScript**
- **Consome uma string e invoca o interpretador da linguagem ou executa um comando do S.O.**

Use API intermediária (wrapper de API) entre o código da app e as interfaces de uso do eval ou exec

- **Precaução expõe API mais segura para a aplicação**

Uso de reflexão é escolha de design arriscada porque defeitos podem levar a execução de código por um atacante

- **Ex.: inclusão ou modificação de métodos/funções em um objeto**

CWE OWASP Top 10 2017 A1 (Injection): https://cwe.mitre.org/data/definitions/1027.html
CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection'): https://cwe.mitre.org/data/definitions/95.html

DF-05: Dados validados, será?

Garanta que todos os dados são validados explicitamente

Usar validadores centralizados para validar todos os dados de entrada do mesmo jeito

- **O validador é um filtro ou interceptador na arquitetura da aplicação**
- **Ele transforma dados em formato interno, antes das validações sintáticas ou semânticas**

Protocolos de comunicação devem validar todos os campos de todas as mensagens recebidas (antes de processá-las)

Arquivos de dados em formatos complexos (XML, imagens, textos ricos, etc.) devem passar por validações sintáticas e semânticas

Validação sintática: de formato e estrutura

Validação semântica: lógica de negócios

Validação sintática

- **Use bibliotecas de validação que reconhecem formatos de endereços de e-mail, URLs, etc.**
- **Use tipos de dados estruturados para capturar suposições sobre validade de dados.**
- **Por exemplo, string que representa data/hora deve ser validada se é uma data/hora bem formada (p.ex., no formato ISO 8601)**

Validação semântica

- **A validação de entrada geralmente depende do estado da aplicação (da lógica de negócios)**
- **Reavaliar suposições, premissas e pré-condições sobre os dados, nos trechos de código próximos ao uso dos dados**

CWE-20: Improper Input Validation https://cwe.mitre.org/data/definitions/20.html

DF-06: Uso incorreto de criptografia

Maioria das vulnerabilidades criptográficas está no uso incorreto

Criptografia de qualidade está disponível para todos

Fonte mais comum de problemas com criptografia é o software em torno da criptografia, escrito por desenvolvedores comuns

Usos incorretos de criptografia mais comuns

- **Não prever adaptação/evolução da criptografia**
- **Usar/inventar sua própria criptografia**
- **Uso incorreto de bibliotecas/APIs criptográficas**
- **Gestão (geração, distribuição, armazenamento) inadequada de chaves criptográficas**
- **Aleatoriedade insuficiente ou falsa**
- **Usar diversas implementações criptográficas diferentes e descentralizadas**

Tamanhos de chaves e algoritmos

AES	RSA	DH (k)	DH (p)	ECC	hash
80	1024	160	1024	160–163	SHA1 (160)
112	2048	224	2048	224–233	SHA-224/512 SHA3-224
128	3072	256	3072	256–283	SHA-256/512 SHA3-256
192	7680	384	7680	384–409	SHA-384 SHA3-384
256	15360	512	15360	512–571	SHA-512 SHA3-512

CWE Cryptographic Issues https://cwe.mitre.org/data/definitions/310.html

CWE Key Management Errors https://cwe.mitre.org/data/definitions/320.html

DF-07: Não reconhecer dados sensíveis

Identifique os dados sensíveis e saiba como protegê-los

Projetistas devem identificar dados confidenciais / privados e determinar como protegê-los

- **P. ex. classificação da informação para LGPD**

Sensibilidade dos dados depende do caso (legislação, política, contratos, vontade do usuário, etc.)

- **A política de classificação em níveis orienta o manuseio seguro dos dados**
- **Níveis: público, restrito, privado, secreto, etc.**

Muitos tipos de dados sensíveis

- **fornecidos pelo usuário, derivados pela aplicação,**
- **coletados de sensores, material criptográfico,**
- **informações bancárias, de compras, de cartões e informações privadas (ex. PII), etc.**

Dados existem em repouso (armazenados), em trânsito e em uso

- **Componentes, sistemas, organizações**

O software deve proteger os dados sensíveis nas diversas situações

- **A sensibilidade dos dados e as proteções decorrentes dela podem mudar com o tempo**

Tópico muito propenso a mudanças

- **Negócios evoluem, regulamentações mudam, sistemas são interconectados e novas fontes de dados são incorporadas à aplicação**

CWE-200: Exposure of Sensitive Information to an Unauthorized Actor
https://cwe.mitre.org/data/definitions/200.html

DF-08: Subestimar o usuário criativo

Sempre considere como os usuários usam a aplicação

A segurança da aplicação depende do que os usuários fazem com ela

Os usuários do software variam desde o pessoal da implantação, operação, configuração e manutenção até os usuários finais

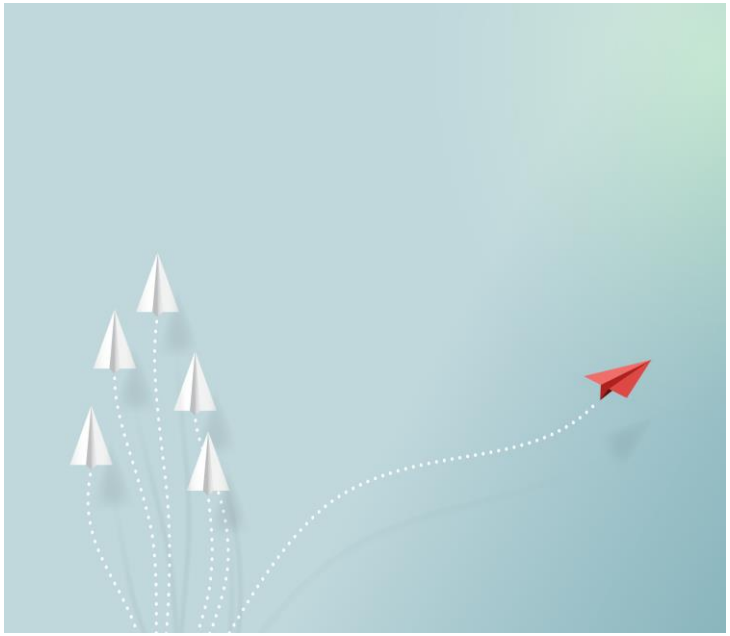
Usabilidade e experiência do usuário são geralmente importantes para a operação segura do software

Há usuários legítimos sem interesse em segurança

- **Eles podem ser mal-intencionados**

Software deve ser configurado e usado com segurança

- **Interfaces intuitivas e fáceis de usar**
- **Controles de segurança expressivos e sem excesso**



CWE User Interface Security Issues: https://cwe.mitre.org/data/definitions/355.html

DF-08 (2): Subestimar o usuário criativo

Problemas ao ignorar o usuário criativo no projeto de segurança

A escalada de privilégios pode resultar de falha ou falta de autorização

- **Autorização sem vínculo com usuário autenticado (supõe errado que o usuário não tem acesso)**

Falta de compartimentação e isolamento entre usuários

- **Usuário acessa dados de outro usuário**

Configurações padrão "abertas" favorecem navegação "livre" do usuário legítimo (autenticado e autorizado) que é um atacante

Quando a segurança é muito difícil de configurar para uma grande população de usuários, ela nunca será configurada ou não será configurada corretamente

Programadores que usam APIs podem ser atacantes

"Danos colaterais" podem ocorrer na implementação de segurança na interface do usuário, com vazamentos de dados

- **P.ex. shoulder surfing em aplicativos móveis usados no transporte público**

Não considerar situações raras e exceções

- **Não considerar apagamento e revogação do usuário durante todo o ciclo de vida: configuração, uso e rescisão**
- **Não considerar as classes diferentes de usuários (usuários cegos, proficiência em idiomas, crianças, idosos, pessoas com diferentes capacidades mentais, etc.)**

CWE User Interface Security Issues: https://cwe.mitre.org/data/definitions/355.html

DF-09: Confiar em componentes de terceiros

Integração de componentes externos muda a superfície de ataque

- **Incluir no SDLC a avaliação do impacto na segurança de componente externo novo**

Exemplos de possíveis problemas de segurança com componentes de terceiros:

- **Carregar uma biblioteca com vulnerabilidades conhecidas (CWE, CVE, etc.)**
- **Incluir uma biblioteca com recursos extras que envolvem riscos de segurança**
- **Reutilizar uma biblioteca que não atenda aos padrões atuais de segurança**
- **Utilizar serviço de terceiros e passar a responsabilidade da segurança para ele**
- **Errar na configuração da segurança de uma biblioteca – p. ex., padrões seguros**
- **Incluir biblioteca que envia requisições para o site do criador ou algum parceiro**
- **Incluir biblioteca que recebe solicitações de alguma fonte externa**
- **Usar componente externo com muitos níveis de inclusão, dependência "recursiva"**
- **Incluir componentes com interfaces desconhecidas (p.ex. CLI, interface web, autenticação própria, uma interface de depuração, backdoor, etc.)**

CWE-829: Inclusion of Functionality from Untrusted Control Sphere https://cwe.mitre.org/data/definitions/829.html
CWE-1104: Use of Unmaintained Third-Party Components https://cwe.mitre.org/data/definitions/1104.html

DF-10: Esquecer que sempre há novidades

Seja flexível ao considerar alterações futuras

- **Projete as aplicações e os componentes de segurança com uma visão das mudanças futuras**
- **Para atualizações seguras de partes ou do todo**
- **Para propriedades de segurança que mudam no tempo**
- **Para quando o código é atualizado para versões novas**
- **Para isolar ou alternar funcionalidades comprometidas**
- **Para alterações em objetos mantidos em segredo**
- **Para alterações nas propriedades de segurança de componentes fora de controle (p.ex., externos)**
- **Para alterações nos tipos de permissões e de usuários**

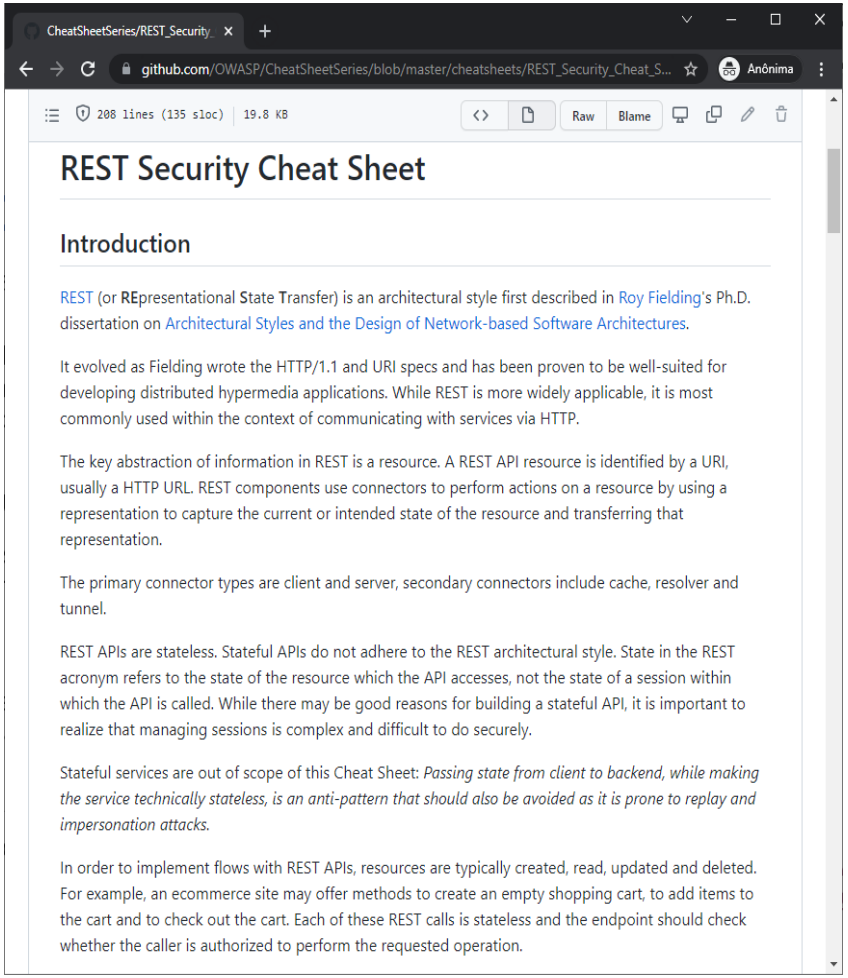
CWE-477: Use of Obsolete Function https://cwe.mitre.org/data/definitions/477.html
CWE-547: Use of Hard-coded Constants https://cwe.mitre.org/data/definitions/547.html

Segurança de APIs

Segurança de APIs

Segredos e a segurança de chaves de APIs

- Os aplicativos da web modernos usam várias APIs para se conectar a outros aplicativos e manter a comunidade on-line conectada
- APIs reduzem o tempo de desenvolvimento e facilitam o desenvolvimento de aplicativos
- Se não forem usadas com segurança, as APIs desprotegidas podem representar sérios riscos à segurança dos dados, como data breaches e negação de serviço (DoS)
- Desenvolvedores inexperientes deixam as chaves de API expostas na Internet, seja em sites de coding ou fóruns de suporte
- Se o invasor acha uma chave de API, ele pode usá-la para extrair informações confidenciais dos aplicativos ou usar serviços pagos



https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/REST_Security_Cheat_Sheet.md

Segredos no código fonte

Senhas e chaves no código fonte e gestão de segredos

Era uma vez uma **chave de API** e uma senha de e-mail

- Estavam embutidas no código fonte
- Foram para o Github (da empresa ou pessoal)
- Não foram detectadas em casa
- Foram detectadas pelo Threat Inteligence do parceiro
- Foram implantadas em um webserver inseguro
- Podem ter sido exploradas em ataques automatizados

Quais processos falharam?

- **OWASP SAMMv2 Secrets Management!**
- https://owaspsamm.org/

Secret Management	1	**Do you limit access to application secrets according to the least privilege principle?**
		You store production secrets protected in a secured location Developers do not have access to production secrets Production secrets are not available in non-production environments
	2	**Do you inject production secrets into configuration files during deployment?**
		Source code files no longer contain active application secrets Under normal circumstances, no humans access secrets during deployment procedures You log and alert to any abnormal access to secrets
	3	**Do you practice proper lifecycle management for application secrets?**
		You generate and synchronize secrets using a vetted solution Secrets are different between different application instances Secrets are regularly updated

Detecção de segredos

- Gestão de segredos em Azure/Visual Studio/ dotNET
- https://devblogs.microsoft.com/visualstudio/managing-secrets-securely-in-the-cloud

- Ferramentas de busca/detecção de segredos em repositórios do tipo git:

- truffleHog - *Searches through git repositories for secrets, digging deep into commit history and branches*
  - https://github.com/dxa4481/truffleHog

- GitLeaks - *Scan git repos (or files) for secrets using regex and entropy*
  - https://github.com/zricethezav/gitleaks

- SecretFinder - *A python script for find sensitive data (apikeys, accesstoken,jwt,..) and search anything on javascript files*
  - https://github.com/m4ll0k/SecretFinder

Proteção das APIs

Boas práticas de proteção de APIs no ASVS

V3.5 Token-based Session Management

- **3.5.1** Verify the application allows users to revoke OAuth tokens that form trust relationships with linked applications
- **3.5.2** Verify the application uses session tokens rather than static API secrets and keys, except with legacy implementations
- **3.5.3** Verify that stateless session tokens use digital signatures, encryption, and other countermeasures to protect against tampering, enveloping, replay, null cipher, and key substitution attacks

V13: API and Web Service Verification Requirements

- V13.1 Generic Web Service Security
- V13.2 RESTful Web Service
- V13.3 SOAP Web Service
- V13.4 GraphQL

Medidas básicas de proteção de API

- API gateway para controle de acesso declarativo
- HTTP sobre SSL/TLS (HTTPS) para o canal de comunicação
- Web Application Firewall (WAF) para as portas abertas
- Geração de chaves de API aleatórias e controladas

Segurança de APIs e lógica de negócios

Vulnerabilidades de APIs são vulnerabilidades semânticas/lógicas ou de projeto!

- Em geral, a lógica das aplicações é defeituosa de alguma forma e as falhas lógicas são bastante variadas
- As falhas lógicas vão de defeitos simples em trechos de código até vulnerabilidades complexas na interoperação de componentes
- Às vezes, elas são óbvias e fáceis de detectar; outras vezes, são muito sutis e não são percebidas em revisões de código ou testes de intrusão
- **As falhas lógicas dificilmente são eliminadas por meio de programação segura, análise estática de código ou testes de intrusão normais**
- Detectar e prevenir falhas lógicas muitas vezes requer pensamento lateral (fora da caixa)

Fontes: OWASP Tesing Guide V4
The Web Application Hacker's Handbook, 2nd Ed.

Segurança de APIs e lógica de negócios

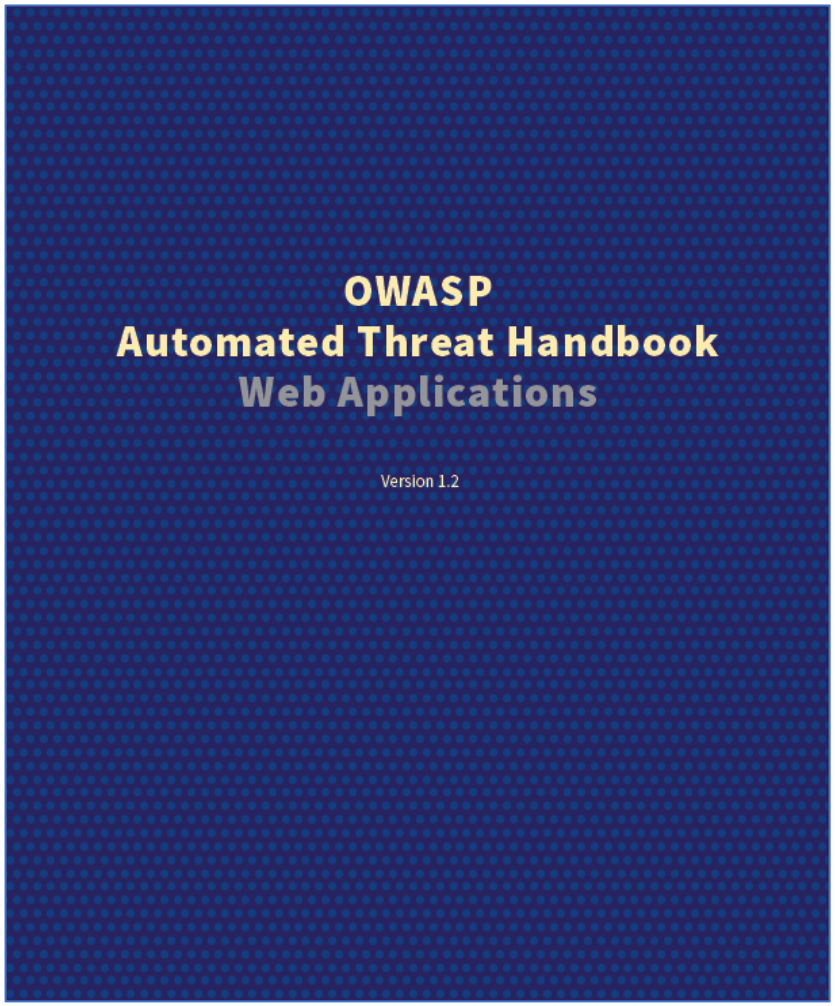
Vulnerabilidades de APIs são vulnerabilidades semânticas/lógicas ou de projeto!

- Recomendações para evitar as vulnerabilidades semânticas:
- Validação (sintática e semântica) dos dados de entrada da lógica de negócios
- Proteção contra falsificação de transações, requisições, solicitações, pedidos, etc.
- Autotestes (automáticos) de integridade do processamento
- Proteções contra variações de tempo e canais laterais de tempo
- Imposição de limites ao número de vezes que funções críticas são chamadas
- Controle do “passo a passo” dos fluxos de trabalho (workflows)
- Controles internos contra “maus usos” das aplicações e APIs
- Controles contra upload de arquivos de tipos errados ou maliciosos
- Projetar aplicações com as boas práticas de projeto seguro!

Fontes: OWASP Tesing Guide V4
The Web Application Hacker's Handbook, 2nd Ed.

Ameaças automatizadas

- Estão relacionadas ao mau uso ou ao abuso de funcionalidades legítimas das aplicações
- Podem ser causadas por falhas de projeto e não por bugs/defeitos de codificação insegura
- Se manifestam na lógica de negócios da aplicação e são associadas às vulnerabilidades semânticas
- Exemplos de automated threats:
- Account enumeration
- Aggregation
- Clickfraud
- Comment span
- Content scraping
- Etc.



OWASP Automated Threat Handbook Web Applications
- Version 1.2 published 15th February 2018
- ISBN 978-1-329-42709-9
- © 2015-2018 OWASP Foundation

https://owasp.org/www-project-automated-threats-to-web-applications

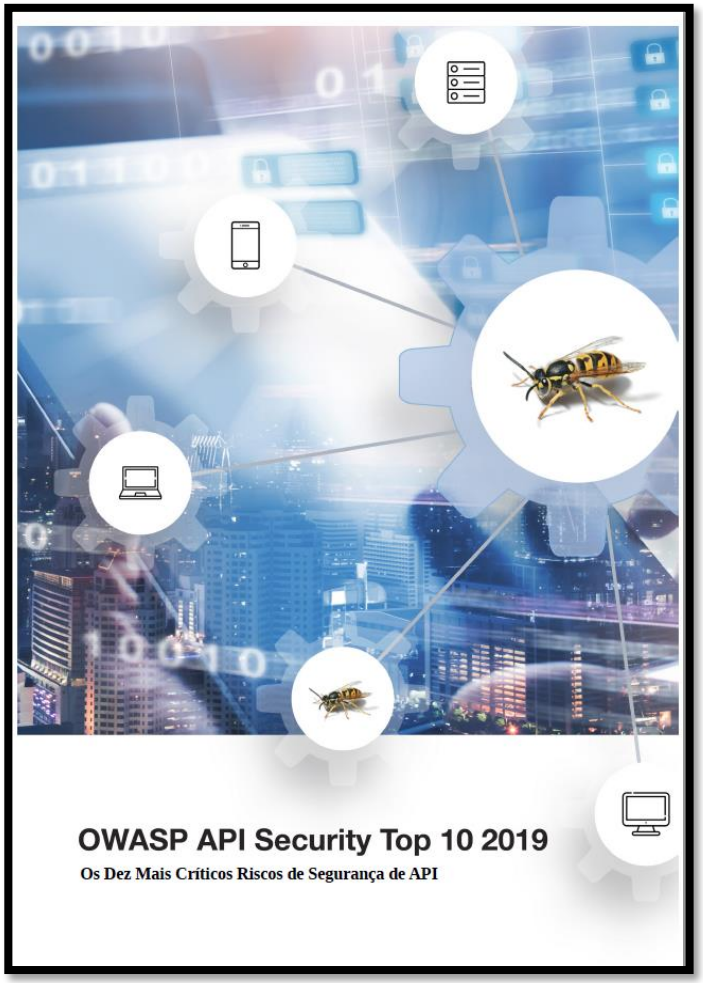
Segurança de APIs no OWASP

OWASP API Security 2019

https://owasp.org/www-project-api-security/

OWASP Top 10 API Security Risks

- API 01: Broken Object Level Authorization
- API 02: Broken User Authentication
- API 03: Excessive Data Exposure
- API 04: Lack of Resources & Rate Limiting
- API 05: Broken Function Level Authorization
- API 06: Mass Assignment
- API 07: Security Misconfiguration
- API 08: Injection
- API 09: Improper Assets Management
- API 10: Insufficient Logging & Monitoring



Segurança de APIs

OWASP API Security 2019

API 01: 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.
API 02: 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.
API 03: 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.
API 04: Lack of Resources & Rate Limiting	Não raramente APIs não impõem restrições ao tamanho ou à quantidade de recursos que podem ser requisitados pelo cliente e/ou 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.
API 05: 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.

Segurança de APIs

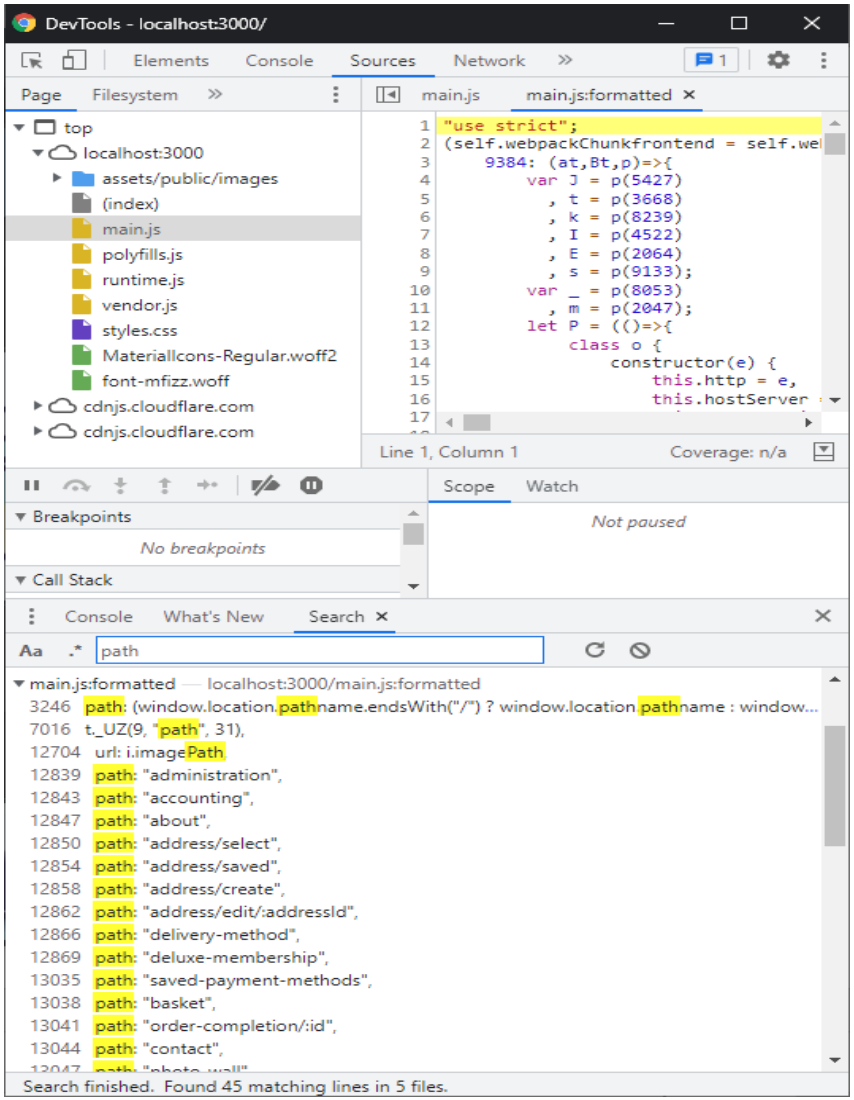
OWASP API Security 2019

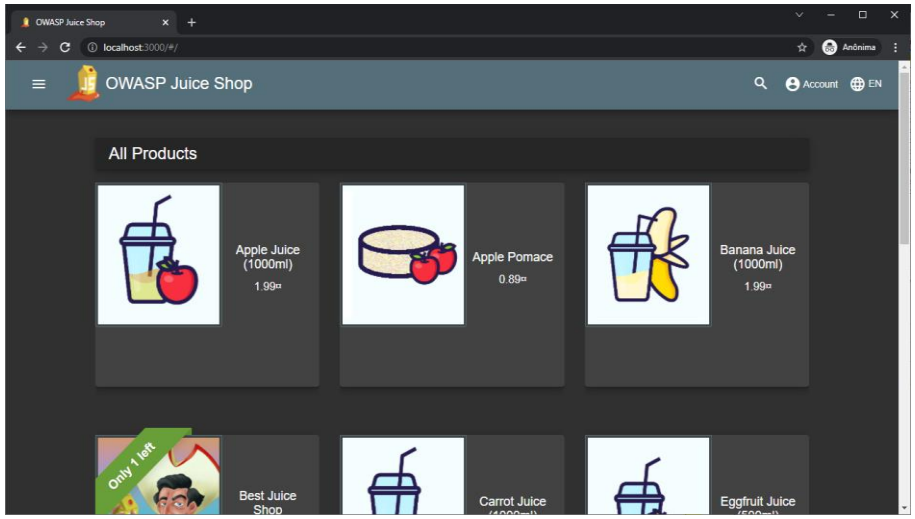
API 06: Mass Assignment	Conectar dados entregues pelo cliente (ex. JSON) diretamente em modelos de dados sem filtrar apropriadamente propriedades em whitelist podem eventualmente levar a atribuição em massa. Mesmo buscando adivinhar propriedades, explorando outros endpoints, consultando documentação e enviando propriedades adicionais em payload os atacantes podem modificar propriedades de objetos os quais não deveria.
API 07: 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.
API 08: 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.
API 09: 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.
API 10: 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.

Parte 4-2 (Laboratório)

Descobrindo a API do Juice Shop

- Na tela/página inicial da aplicação Juice Shop:
- Ferramentas do desenvolvedor → Sources
- Visualizar o arquivo main.js
  - Usar Pretty printing ou botão {}
- Buscar (menu três pontos verticais → buscar) pela palavra ***path***.
  - O resultado da busca é a API da aplicação!
- Identificar a API do dashboard de desafios e acessá-la.
  - Spoiler alert!
    - http://localhost:3000/#/score-board
- Outras buscas : path, API, REST





API 01: Broken Object Level Authorization

Na Juice Shop → desafio ***View another user's shopping basket***

- O objetivo é visualizar a cesta de compras de um usuário, qualquer um, sem estar logado como ele
- Criar dois usuários com login/senha, por exemplo, user1@juice/user1 e user2@juice/user2
- Encher a cesta do user2 com diversos produtos do juice shop
- Descobrir e explorar a API da cesta de compras
- Exploração 1
    - abrir as ferramentas de desenvolvedor
    - Entrar o App Juice Shop logado como user1
    - acessar a cesta de compras do user1
    - em DevTools, em Network, observar que a API REST da cesta é muito simples
        - P.ex., http://localhost:3000/rest/basket/7
        - O que é o número 7?
    - em DevTools, Application → Session Storage
        - aparecem dois elementos bid = 7 e ItemTotal = 0.
            - será o mesmo número 7?
    - Conclusão deduzida: bid quer dizer BasketID
    - Alterar bid para algum valor qualquer e atualizar a página.
        - por exemplo, o valor bid = 6 !
        - o que acontece?
    - Spoiler Alert!
        - Aparece a cesta de compras do user2.

API 01: Broken Object Level Authorization

Exploração 2

- no OWASP Zap Proxy do JuiceShop
- interceptar e reenviar a requisição do basket
  - botão direito na requisição do histórico;
  - Na tela de reenvio, substituir o valor de bid na requisição por outro valor qualquer até achar um valor válido.
- Esta exploração pode ser automatizada para baixar as cestas de compras de todos os usuários. (como?)
- Por que este ataque acontece?

API 02: Broken User Authentication
API 08: Injection

- Exploração 1
  - Explorando a API da aplicação
    - Logar na aplicação como user1@juice
    - no OWASP Zap Proxy do JuiceShop
      - interceptar e reenviar a requisição de products/search
    - GET http://localhost:3000 /rest/products/search?q=
      - botão direito na requisição do histórico
      - Na tela de reenvio, substituir q= por q='))--
  - Por que não q='-- ?
    - Spoiler alert! A requisição products/search sofre de SQLi!
    - Na tela de reenvio, substituir q= pelo seguinte
- **')) UNION SELECT id, email, password, '4', '5', '6', '7', '8', '9' FROM Users--**
    - Reenviar a requisição injetada!
    - Pergunta 1: Como sei que o nome da tabela é Users?
  - Educated guess! na API REST, serviços = tabelas?!
    - Pergunta 2: Como sei a quantidade de campos?
  - Tentativa e erro...

Descobrindo a senha dos usuários
- Na resposta da requisição, procurar pelo user1@juice
- A senha desse usuário está no campo description
  - por exemplo, 24c9e15e52afc47c225b757e7bee1f9d
  - No website https://crackstation.net/
- Digitar o hash e apertar o botão CrackHashes
- deu um match perfeito com md5 e user1
  - Descobrimos o algoritmo de hash usado na criptografia da senha!
- Repetindo o processo para o usuário admin
  - O hash é 0192023a7bbd73250516f069df18b500
  - O website https://crackstation.net/ diz que a senha é admin123
- Tarefa: Descobrir a senha dos outros administradores!

API 02: Broken User Authentication
API 08: Injection

Exploração 2

- Logar na aplicação com email ' or 1=1-- e qualquer senha
  - a aplicação vai logar como a primeira entrada na tabela de usuários Users
  - que é o Admin

Exploração 3

- Logar na aplicação com email admin@juice-sh.op'-- e qualquer senha
  - 2a. parte da query SQL é comentada e a senha não é verificada

Exploração 4

- Logar na aplicação com email admin@juice-sh.op e senha admin123
- Senha default descoberta por tentativa e erro ou busca exaustiva com dicionário

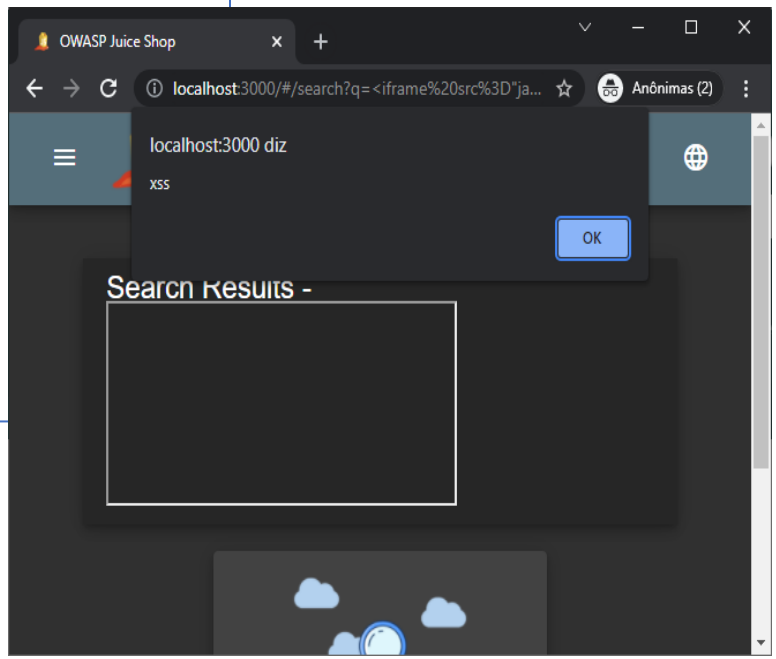
API 08: Injection – Cross Site Script (XSS)

Exploração de XSS no Juice Shop
Um passo-a-passo dessa exploração pode ser o seguinte:

- Entrar na aplicação Juice Shop
- Logar na aplicação com usuário e senha (criar usuário, se necessário)

Este desafio explora vulnerabilidades no campo de busca

- Fazer uma busca por um nome de fruta (em inglês), O que acontece?
- O campo de busca exibe de volta para o usuário as palavras buscadas!
- O campo de busca aceita tags HTML?
  - Testar com: <h1> orange
- O campo de busca aceita javascript?
  - Testar com: <script>alert(1)<\scritp>
- A busca aceita DOM?
  - Testar com: <iframe src="javascript:alert(`xss`)">
  - O resultado do XSS está na figura ao lado!!!



API 03: Excessive Data Exposure
API 05: Broken Function Level Authorization

- Logar na aplicação com email ' or 1=1-- e qualquer senha
  - a aplicação vai logar como a primeira entrada na tabela de usuários Users
  - que é o Admin

Em DevTools do browser

- Visualizar o arquivo main.js
  - Usar ***Pretty printing*** ou botão ***{}***
  - Buscar (menu dos três pontos verticais → buscar) pela palavra *admin.*
    - *Existe um path administration*
  - Buscar (menu três pontos verticais → buscar) pela palavra *user.*
    - *Existe uma api rest user*

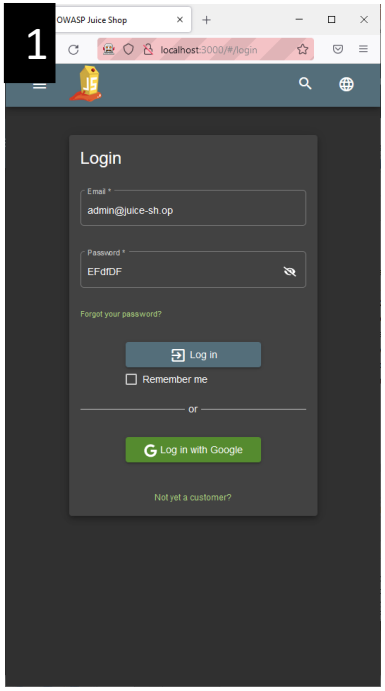
      O resultado da busca dá o serviços administrativo da loja

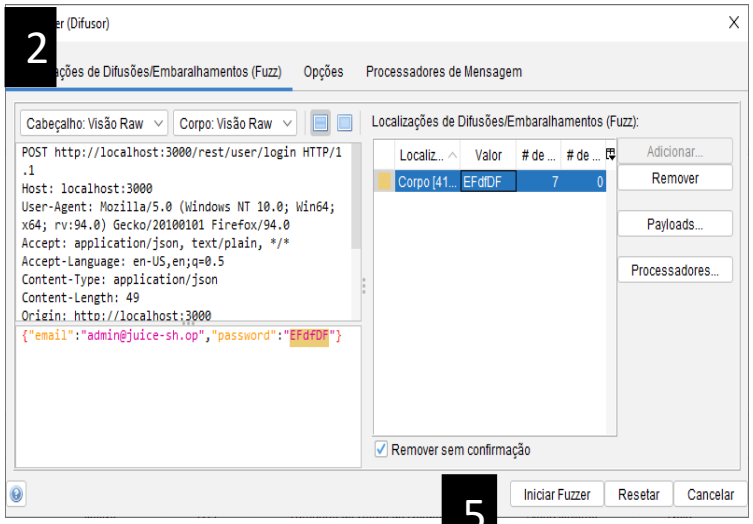
API 03: Excessive Data Exposure
API 05: Broken Function Level Authorization

- Na aplicação Juice Shop com OWASP Zap Proxy
  - Logar como admin na aplicação com email ' or 1=1-- e qualquer senha
  - Acessar http://localhost:3000/#/administration
    - aparece uma lista de usuários
  - clicar em qualquer um dos usuários da lista (ícone de olho)
- No OWASP ZAP
  - Verificar no histórico de requisições que a API Users foi ativada
    - p.ex., http://localhost:3000/api/Users/2
    - o que significa o 2?
  - Reenviar esta requisição com outros parâmetros
    - por exemplo, 3, 5, 8
      - devolve o Json do usuário correspondente
    - sem parâmetros
      - Esta opção devolve o json de toda a base de dados de usuário!

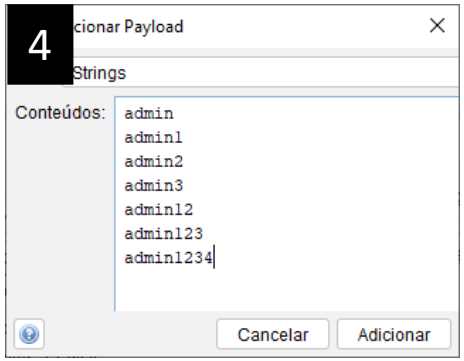
API 04: Lack of Resources & Rate Limiting

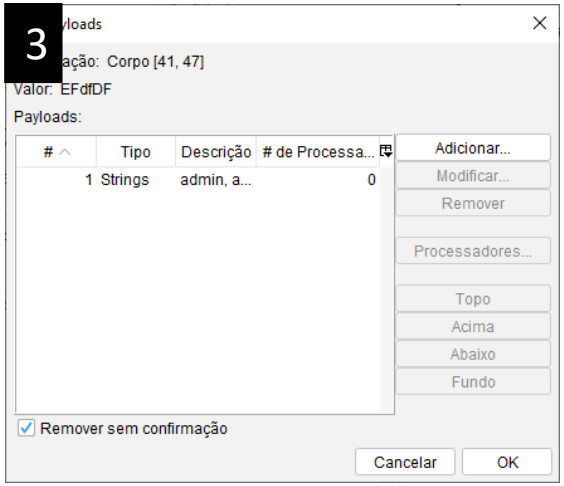
Brute force de login com OWASP ZAP: OAT-07 (Credential Cracking) e OAT-08 (Credential Stuffing)





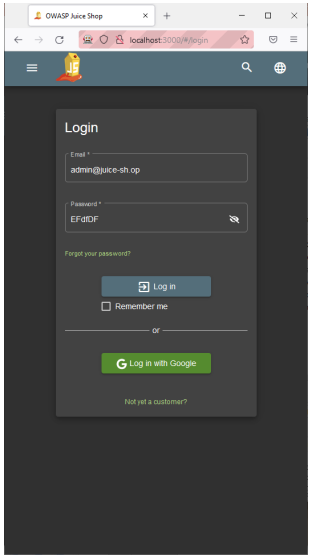
1. Gera uma requisição de login para fuzzer
2. Ativa o fuzzer na requisição no histórico
3. Adiciona um fuzzer
4. Fuzzer de Strings com as senhas testadas
5. Inicia o fuzzer

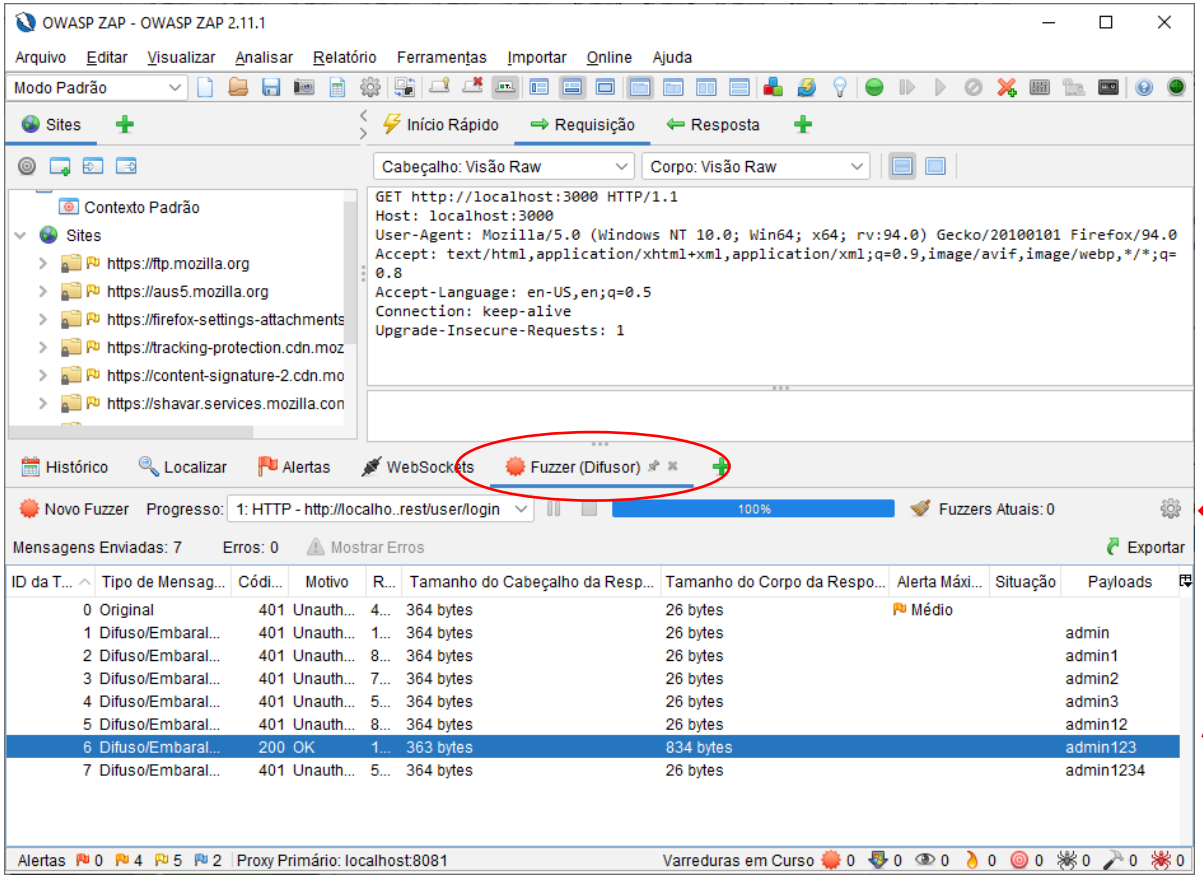




API 04: Lack of Resources & Rate Limiting

Brute force de login com OWASP ZAP: OAT-07 (Credential Cracking) e OAT-08 (Credential Stuffing)





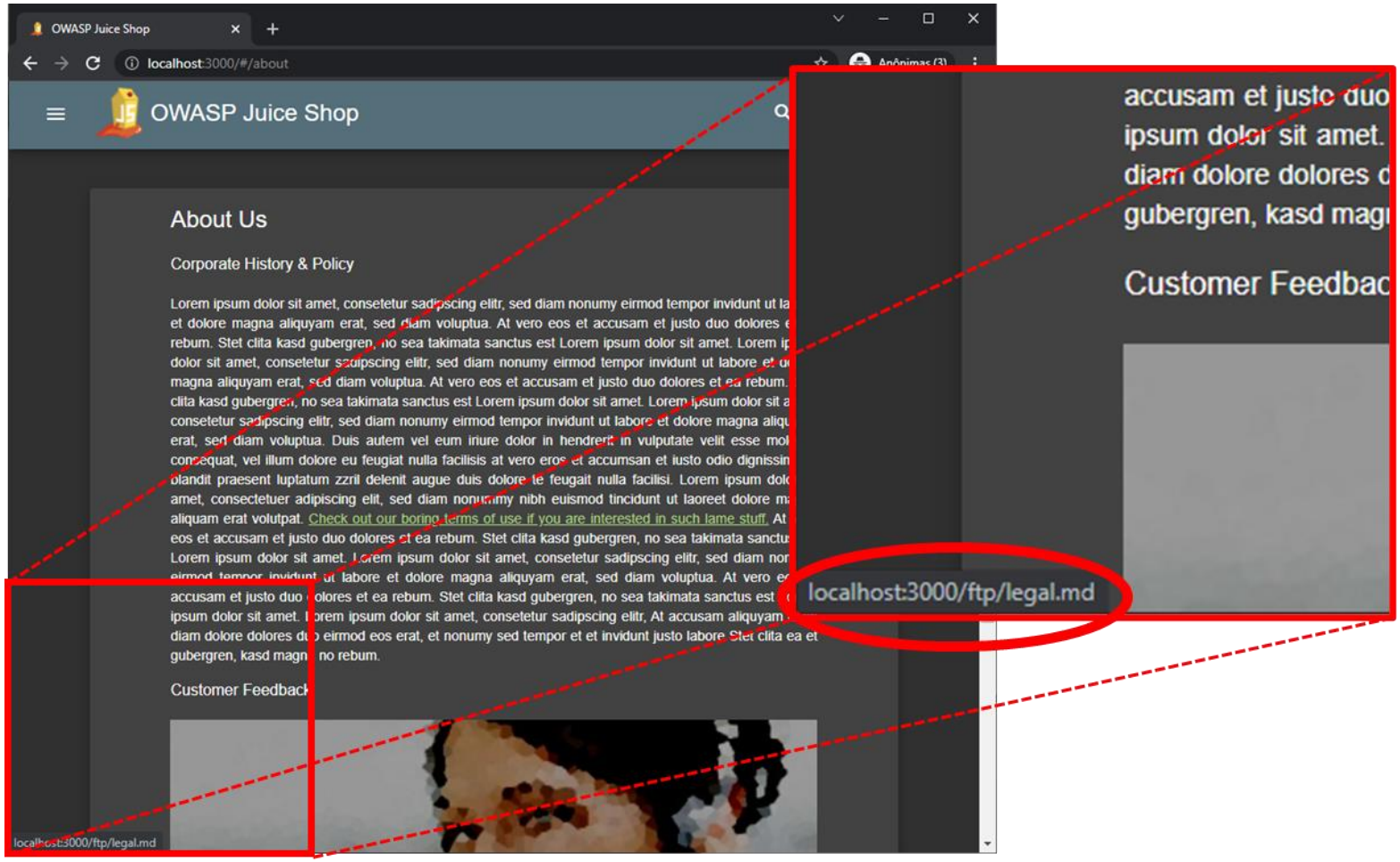
O fuzzer contorna (by-pass) o fron-end da aplicação, que fica no mesmo ponto...

O resultado do fuzzer aparece aqui

Autenticação bem sucedida

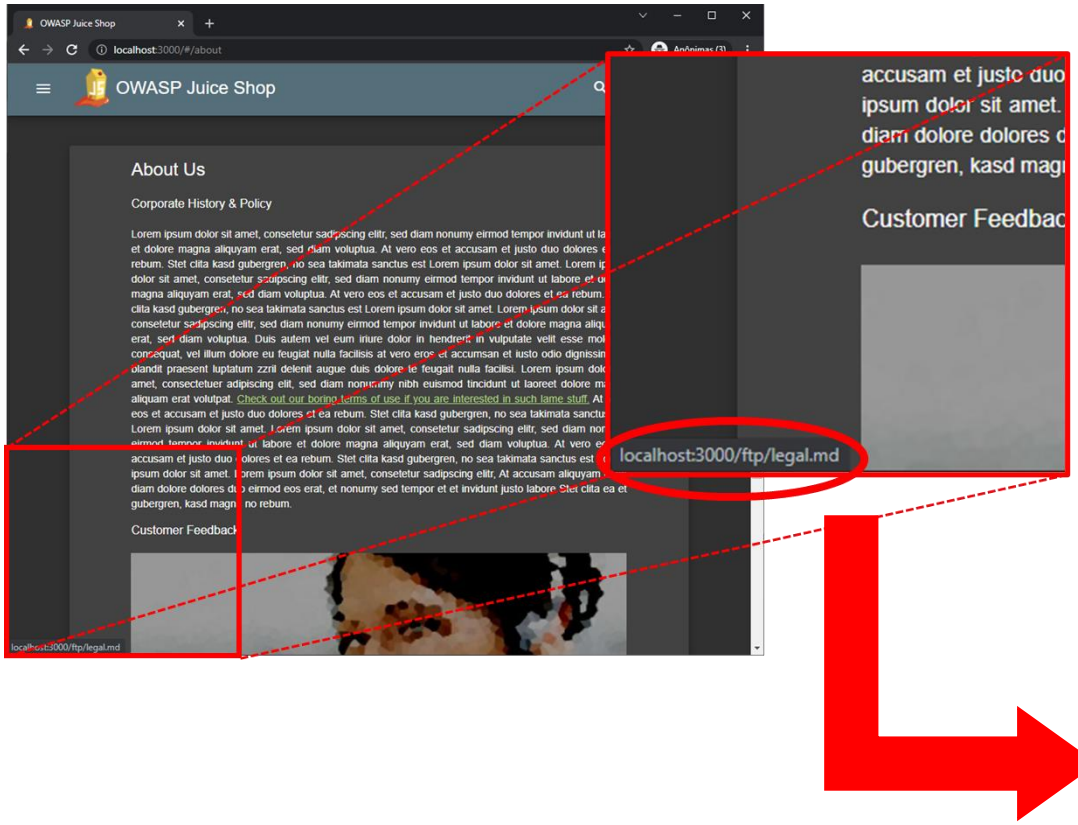
API 07: Security Misconfiguration

Configuração insegura do servidor web: path traversal no Juice Shop



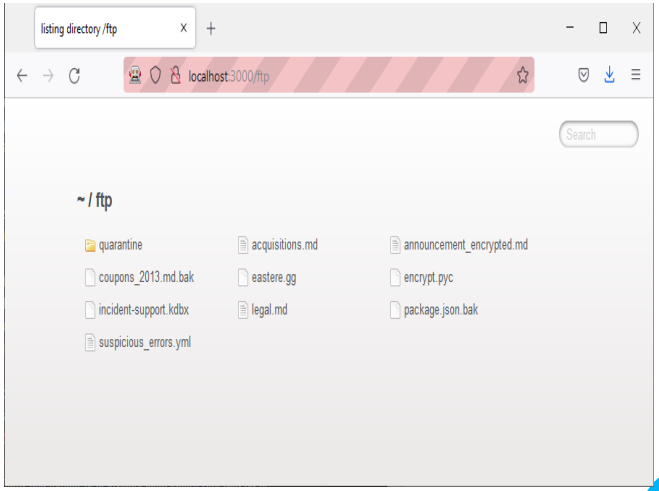
API 07: Security Misconfiguration

Configuração insegura do servidor web: path traversal no Juice Shop



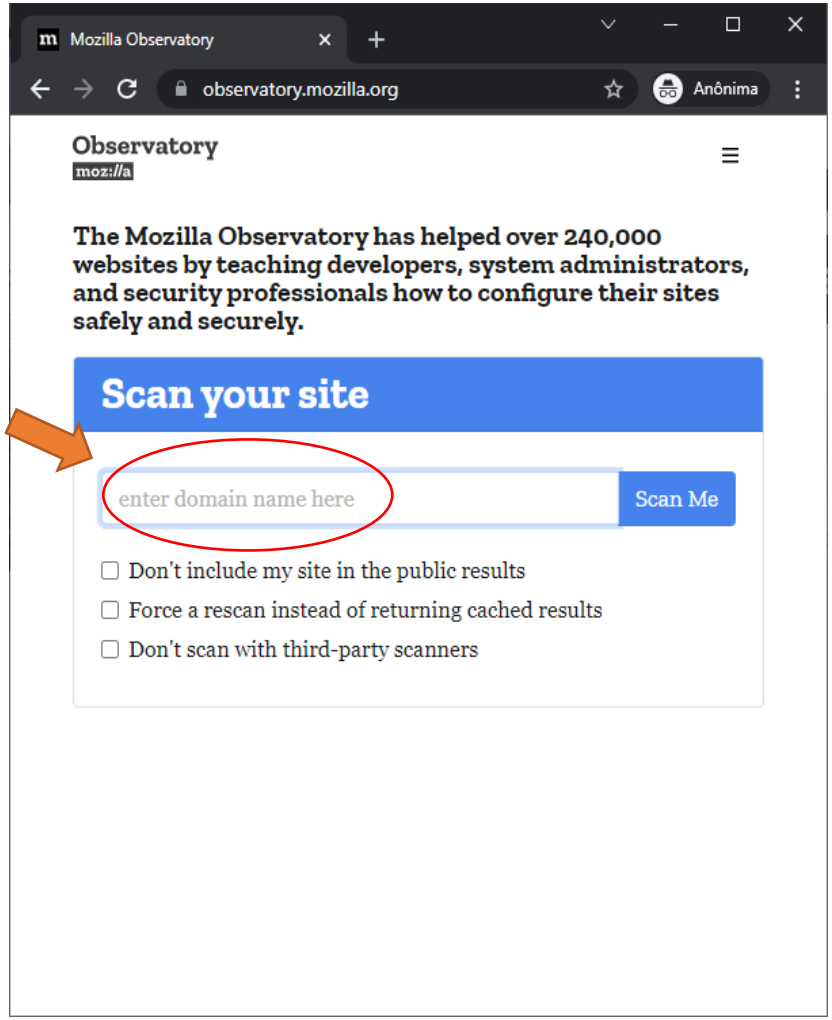
1. Path para uma pasta/diretório no servidor
2. Testar o acesso direto
3. Testar a navegação para a pasta anterior
4. Testar navegação para pasta interior

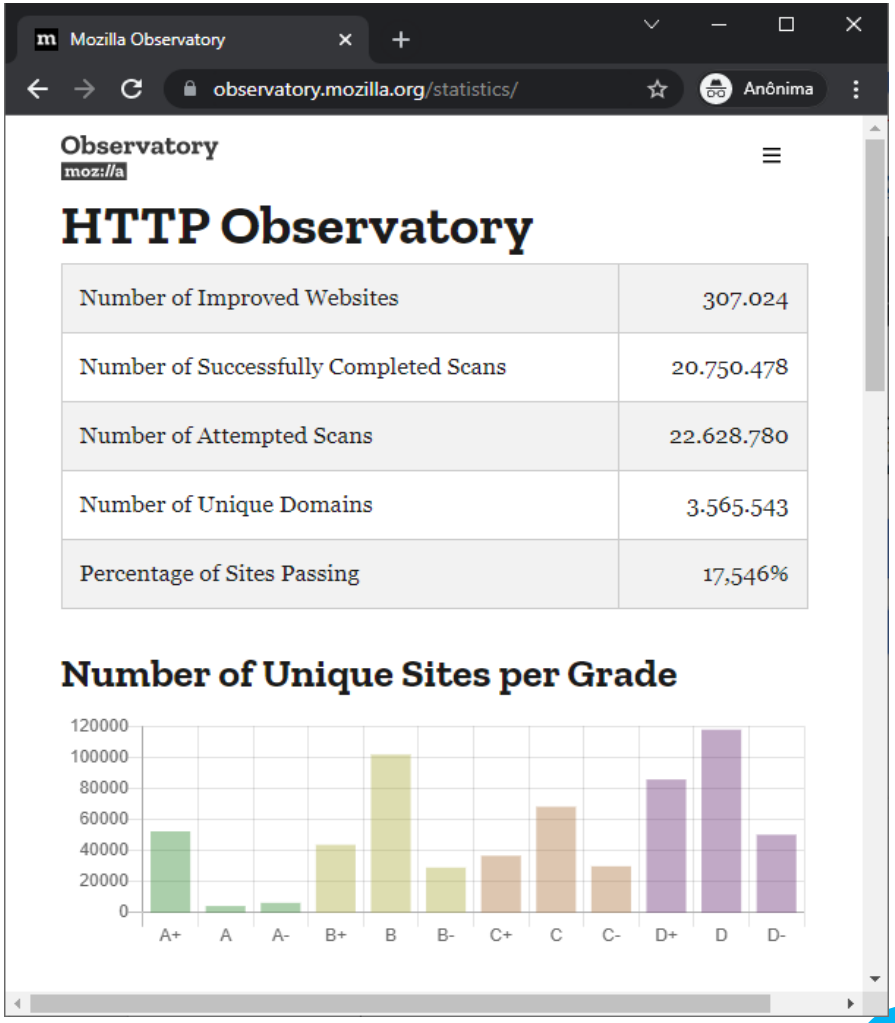
Causa pode ser configuração insegura no webserver e/ou na aplicação



API 07: Security Misconfiguration

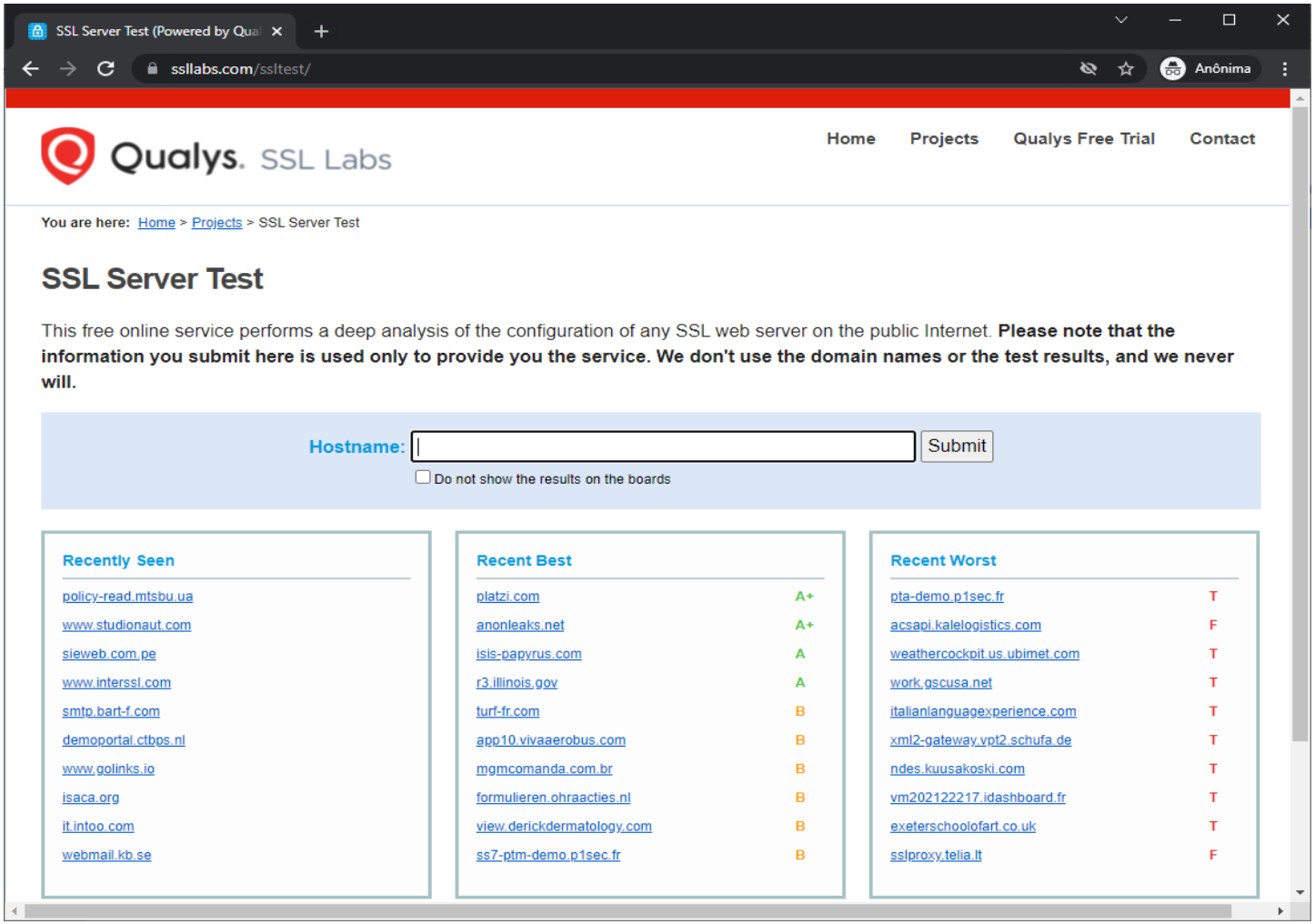
Varredura de HTTP/HTTPS com Mozilla Observatory https://observatory.mozilla.org





API 07: Security Misconfiguration

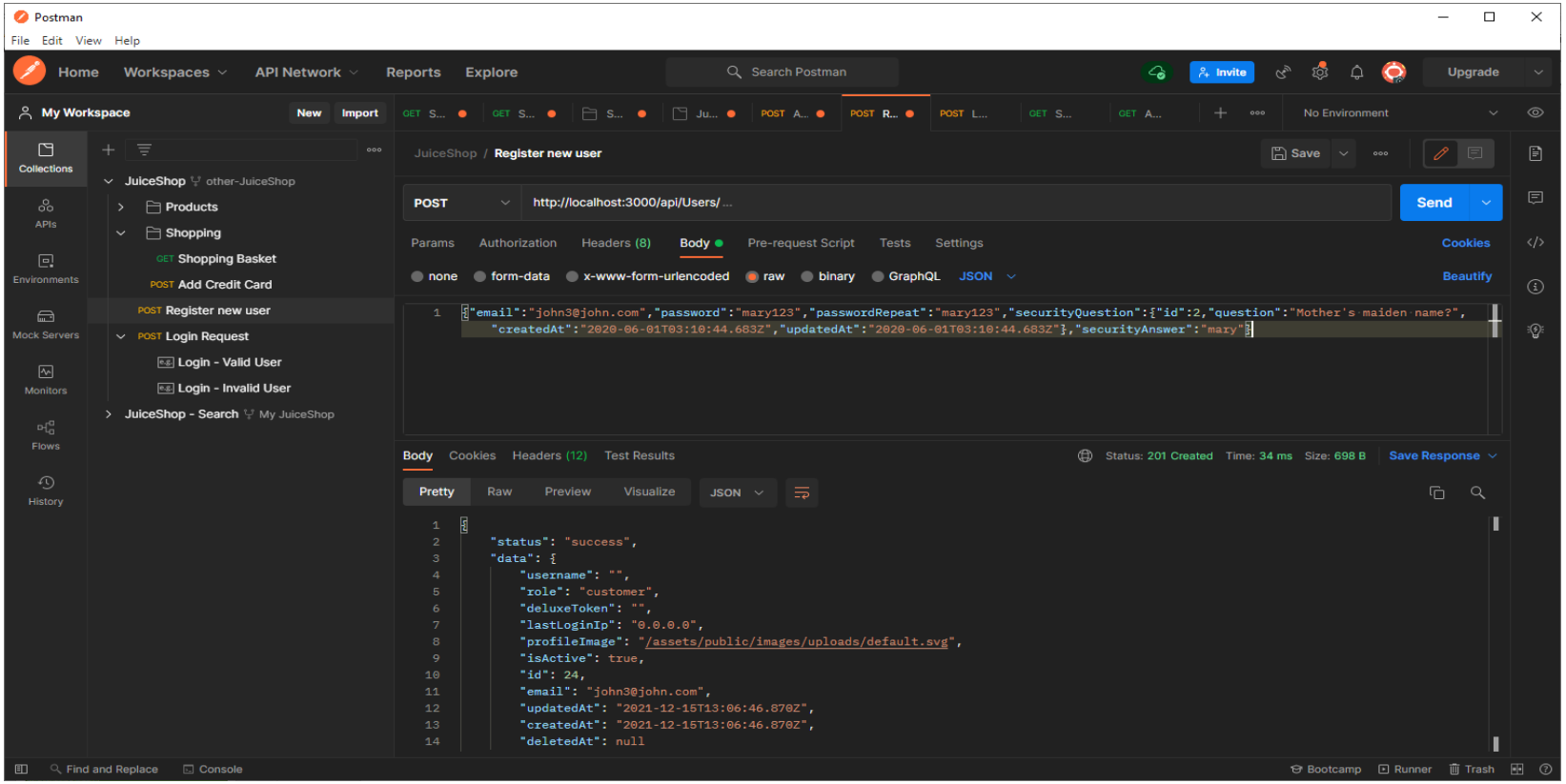
Varredura de HTTPS com Qualys SSL Labs – SSL Server Test:
https://www.ssllabs.com/ssltest/



OAT-019: Account Creation

Criação de vários usuários com POSTMan

1. No Juice Shop, a API de criação de usuário tem acesso livre
2. Acesso sem autenticação porque o usuário ainda não existe
3. É possível bypassar o front-end e criar vários usuários



Obrigado!