



Web API Exploitation

Mini Treinamento - 30/01/2021

Instrutor: Hélio Junior (M4v3r1ck)

- E-mail: helvio.junior@sec4us.com.br
- Twitter: @helvioju
- Mais de 20 anos de atuação com TI
- Foco de estudo e pesquisa:
 - Low Level Security
 - Segurança ofensiva (Red Team)
 - Criação de exploits
- CEO - Sec4US
- <https://sec4us.com.br/>
- <https://github.com/helviojunior>



Agenda

- Fundamentos
- Reconhecimento e ataque
- Abertura do CTF



Referências de estudo

- Burp WebSecurity Academy
 - <https://portswigger.net/web-security>
- W3Scools
 - <https://www.w3schools.com>

- OWASP API Security
 - <https://owasp.org/www-project-api-security/>
 - https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html
- OWASP Top 10 Web
 - <https://owasp.org/www-project-top-ten/>
- OWASP Testing Guide
 - https://owasp.org/www-project-web-security-testing-guide/assets/archive/OWASP_Testing_Guide_v4.pdf
 - <https://github.com/tanprathan/OWASP-Testing-Checklist>

Protocolo HTTP/S

Hypertext Transfer Protocol (HTTP) é o protocolo mais utilizado hoje em dia para navegação web, incluindo APIs.

HTTP é um protocolo cliente-servidor utilizado para transferência de páginas web e dados de aplicações. O Cliente comumente é um navegador web, que inicia a conexão para o servidor como Apache, Microsoft IIS, Nginx, Tomcat, entre outros.

**GET** / HTTP/1.1**Host:** www.google.com**User-Agent:** Mozilla/5.0 (Macintosh; Intel Mac OS X 10.16; rv:84.0) Gecko/20100101 Firefox/84.0**Accept:** text/html,application/xhtml+xml,application/xml**Accept-Language:** pt-BR,pt;q=0.8,en-US;q=0.5,en;q=0.3**Accept-Encoding:** gzip, deflate**Connection:** close

REQUEST METHOD

Este é o tipo da requisição (também conhecido como HTTP verb). **GET** é o tipo de requisição padrão quando digitamos uma URL no navegador e pressionamos ENTER.

Outros métodos são: POST, PUT, DELETE, OPTIONS, TRACE ...

**HTTP/1.1** 200 OK

Date: Thu, 31 Dec 2020 20:00:35 GMT
Cache-Control: private, max-age=0
Content-Type: text/html; charset=UTF-8
Server: gws
Content-Length: 204834

<CONTEÚDO da PAGINA>

STATUS-LINE

A primeira linha de uma resposta HTTP é o Status-Line, que consiste na **versão do protocolo** (HTTP 1.1) seguido por um código numérico (**status code** - 200) e o seu respectivo **texto de significado** (OK).

Os “status codes” mais comuns são:

- **200 OK:** o recurso foi encontrado e retornado com sucesso
- **301 Moved Permanently:** o objeto foi movido permanentemente, e a requisição deve ser realizada em outra URL
- **302 Found:** o recurso está temporariamente em outra URL
- **403 Forbidden:** O Cliente não tem autorização para acessar este recurso e o servidor recusou a requisição
- **404 Not Found:** o servidor não encontrou o recurso desejado
- **500 Internal Server Error:** o servidor encontrou um erro ou não suporta a funcionalidade requisitada.

Os status codes podem ser subdivididos em classes conforme abaixo:

- 1xx: Informacional
- 2xx: Sucesso
- 3xx: Redirecionamento
- 4xx: Erro de cliente
- 5xx: Erro de servidor

Listagem completa em:

<https://www.restapitutorial.com/httpstatuscodes.html>

JSON

JSON é um acrônimo para **JavaScript Object Notation** que foi criado para armazenamento e transporte de dados onde humanos e equipamentos possam entender.

Apesar do JSON levar o nome JavaScript e utilizar sua sintaxe o JSON é somente texto e pode ser lido ou escrito em qualquer linguagem de programação.

De fato atualmente o JSON tem se tornado o formato de armazenamento e transporte de dados mais utilizado, como veremos no decorrer do treinamento.

O JSON deriva do JavaScript object notation e detém a seguinte sintaxe:

- Dados possuem em formato chave/valor;
 - “name”:“Web API Exploitation”
- Dados são separados por vírgula;
 - “name”:“Web API Exploitation”, “version”:“v1”
- Objetos são postos entre {};
 - { “name”:“Web API Exploitation”, “version”:“v1” }
- Arrays são armazenados entre [];
 - “names”: [“Web API Exploitation”, “Exploits Development”]

Web Application Proxies

O **web application proxy** (proxy de aplicações web) atua na camada de aplicação (camada 7 do modelo OSI), permitindo assim, que o payload (requisições e respostas HTTP) sejam filtradas, analisadas e seus logs armazenados.

Para nosso estudo utilizaremos os **proxies de interceptação**, que são ferramentas que nos permite analisar e modificar as requisições e respostas entre o cliente HTTP e o servidor.

As aplicações de proxy mais utilizadas (com foco em pentest em aplicações web e API) são:

- Burp Suite (<https://portswigger.net/burp>)
- OWASP ZAP (<https://owasp.org/www-project-zap/>)

Os proxies são fundamentais para o processo de testes de invasão em aplicações web e API. Neste treinamento utilizaremos massivamente o **Burp Community Edition**.

Burp Suite

Com o Burp suite podemos:

- Interceptar requisições e respostas entre o navegador e o servidor web;
- Criar requisições manualmente;
- Replicar e modificar requisições realizadas anteriormente;
- Realizar web crawler: buscar páginas através de lista de palavras;
- Realizar fuzz: enviar padrões e textos (validos e inválidos) nos campos de entrada da aplicação para testar o seu comportamento;

TurboSearch

Com o TurboSearch podemos:

- Realizar buscas (Web Crawling) em sites e APIs
- Alterar o método HTTP para a busca (GET, POST, PUT e OPTIONS)
- Busca automática no /robots.txt
- Adicionar cabeçalhos (podendo ser de autenticação)
- Utilizar User-Agent randômico
- Pular um diretório durante o teste
- Reportar somente as URLs identificadas ao proxy (Burp)
- Realizar buscas Case Sensitive e Case Insensitive
- Salvar o resultado (URLs identificadas) em um banco de dados SQLite

O TurboSearch pode ser instalado diretamente através do GitHub.

pip3 install git+https://github.com/helviojunior/turbosearch.git#egg=turbosearch

```
[root@M4v3r1ck] ~
  pip install git+https://github.com/helviojunior/turbosearch.git#egg=turbosearch
Collecting turbosearch
  Cloning https://github.com/helviojunior/turbosearch.git to /tmp/pip-install-vtxg7mey/turbosearch_0633b5b3950949b08d639df6faf9be9c
Requirement already satisfied: bs4>=0.0.1 in /usr/local/lib/python3.8/dist-packages (from turbosearch) (0.0.1)
Requirement already satisfied: requests>=2.23.0 in /usr/lib/python3/dist-packages (from turbosearch) (2.23.0)
Requirement already satisfied: beautifulsoup4 in /usr/lib/python3/dist-packages (from bs4>=0.0.1->turbosearch) (4.9.2)
Requirement already satisfied: soupsieve>1.2 in /usr/lib/python3/dist-packages (from beautifulsoup4->bs4>=0.0.1->turbosearch) (2.0.1)
Building wheels for collected packages: turbosearch
  Building wheel for turbosearch (setup.py) ... done
  Created wheel for turbosearch: filename=turbosearch-0.1.6-py3-none-any.whl size=24349 sha256=7cba6f56d92e1b9b2daa270bb24139860b117185837cffa8d4e29b44fb243fe5
  Stored in directory: /tmp/pip-ephem-wheel-cache-7bjk3n4/wheels/8f/ac/03/878fc7b94cbc8182be5791e8f4c0f51a7eec8e6ab169311e75
Successfully built turbosearch
Installing collected packages: turbosearch
Successfully installed turbosearch-0.1.6
```

Os parâmetros de utilização podem ser visualizados com o comando `turbosearch -h` ou através da URL do GitHub

[https://github.com/helviojunior/turbosearch/.](https://github.com/helviojunior/turbosearch/)

```
[root@Mv3r1ck ~]# turbosearch -h
HHHHHHH      +HHH
HHHHHHH      +--+HH
HHHHHH      +----+
+-----+-----+-----+
+-----+-----+-----+
HHHHHH      +----+
HHHHHH      +---+HH
HHHHHH      +---+HHH

Turbo Search v0.1.6 by Helvio Junior
automated url finder
https://github.com/helviojunior/turbosearch

optional arguments:
  -h, --help            show this help message and exit

General Setting:
  -t [target url]        target url (ex: http://10.10.10.10/path)
  -w [word list]         word list to be tested
  -T [tasks]             number of connects in parallel (per host, default: 16)
  -o [output file]       save output to disk (default: none)
  -x [extensions]        Append each request with this extensions (comma-separated values)

Custom Settings:
  -R, --restore          restore a previous aborted/crashed session
  -I, --ignore           ignore an existing restore file (don't wait 10 seconds)
  --proxy [target proxy] target proxy URL (ex: http://127.0.0.1:8080)
  --report-to [target proxy] target proxy URL to report only successful requests (ex: http://127.0.0.1:8080)
  --deep                 Deep Search: Look for URLs inside of HTML results
  -v, --verbose          Shows more options (-h -v). Prints commands and outputs. (default: quiet)
  --full-log             Print full requested URLs (default: no)
  --no-forward-location  Disable forward to Location response address (default: no)
  --ignore-result [filter] ignore results by result code or/and size (ex1: 302 or ex2: 302:172 or ex3: 405,302:172 )
  --find [text to find]   Text to find in content or header (comma-separated values)
  --method [http method] Specify request method (default: GET). Available methods: GET, POST,
                        PUT, OPTIONS
  --random-agent          Use randomly selected HTTP User-Agent header value (default: no)
  --header [text to find] JSON-formatted header key/value
  --ci, --case-insensitive Case Insensitive search: put all wordlist in lower case
  --stats-db              Save reported URI at SQLite local database called stats.db (default: no)
  --no-robots             Not look for robots.txt (default: no)

Word List Options:
  --md5-search            Search for a MD5 Hash version of each word (default: no)
  --sha1-search            Search for a SHA1 Hash version of each word (default: no)
  --sha256-search          Search for a SHA256 Hash version of each word (default: no)
  --hash-upper             In case of Hash Search be enabled, also search by Uppercase of Hash Hex Text (default: no)
```

Parâmetros obrigatórios:

- **-t** URL que se deseja testar
- **-w** Arquivo texto com a lista de palavras a serem testadas

Principais parâmetros opcionais:

- **-T** numero de threads simultâneas
- **-o** Arquivo onde serão salvos os logs
- **-x** Extensões a serem testadas (Ex. .php, .asmx). Ao utilizar este parâmetro o TurboSearch continua testando para encontrar diretórios e adiciona os testes com as extensões definidas.

Principais parâmetros opcionais:

- **--proxy** Envia toda a comunicação web através do proxy definido
- **--report-to** Configuração de proxy para que o turbosearch realize uma requisição através deste, somente quando houver um resultado positivo da URL.

Principais parâmetros opcionais:

- **--deep** Realiza a busca em profundidade na página testada, na prática, lista o HTML da página buscando links presentes no HTML reportando links externos e criando uma fila de teste com os links internos encontrados (do mesmo domínio)
- **--ci** Utiliza o modo Case Insensitive, ou seja, converte todas as palavras testadas para minúsculo durante o teste. Este modo deve ser usado somente em sites que são case insensitive como o ASP.NET
- **--stats-db** Salva as URLs encontradas em um arquivo SQLite (stats.db)

Principais parâmetros opcionais:

- **--method** Define o método HTTP da requisição. O padrão é o GET., Porém com este parâmetro pode ser alterado para POST, PUT, PATCH e OPTIONS
- **--header** Adiciona nas requisições os cabeçalhos definidos.
 - Exemplo: { “Authorization”, “Bearer base64” }
- **--random-agent** Seleciona um novo User-Agent a cada execução do commando TurboSearch

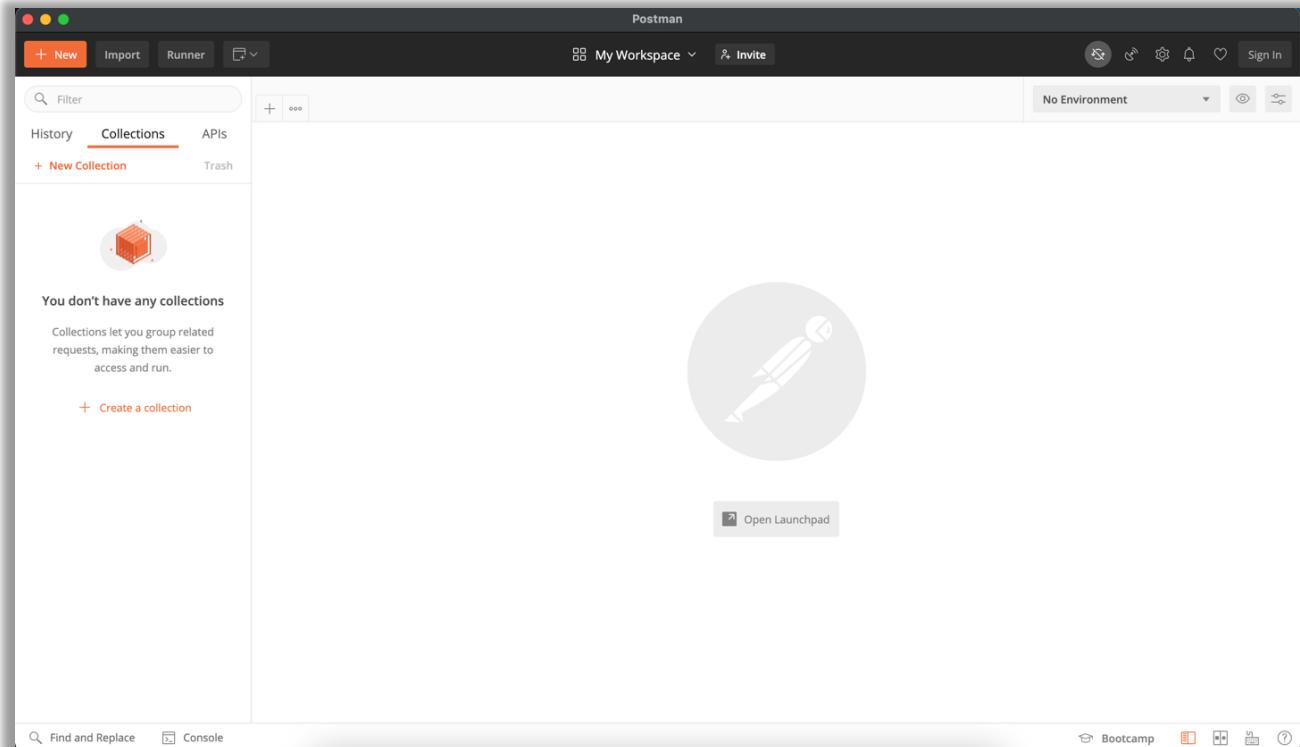
Postman

Com o Postman podemos:

- Criar requisições HTTP manualmente;
- Salvar requisições HTTP dentro de uma coleção e pastas;
- Facilmente realizar autenticação nos diversos padrões de mercado (API Key, Bearer, Basic Auth, Oauth, NTLM entre outros);
- Compartilhar o projeto;
- Muito mais...

Download: <https://www.postman.com/>

Tela inicial do Postman.



Reconhecimento

- Descobrindo APIs
 - Robots.txt
 - Burp
 - Javascript
 - Documentação
 - Buscadores (Google, Bing e Shodan)
 - Web Crawling
 - Informações tradicionais web



Robots.txt

O arquivo `/robots.txt` instrui aos robôs de busca (Google, Bing e etc...) quais áreas do site são permitidas (e quais são proibidas) a busca e indexação automatizada.

Para um atacante, o `robots.txt` pode indicar o caminho para outras áreas do site aumentando a superfície de ataque ou revelando áreas críticas como administrativas e de API

O TurboSearch pode facilmente identificar os caminhos presentes no **/robots.txt** e adicionar automaticamente estes caminhos na listagem de busca.



```
# robots.txt: Treinamento Web API Exploitation
# Sec4Us

User-agent: *
Disallow: /api/v1
```



```
[+] Connection test against http://api1.webapiexploitation.com.br OK! (CODE:200|SIZE:836)
[+] Scanning url http://api1.webapiexploitation.com.br
[+] Getting informations from /robots.txt at api1.webapiexploitation.com.br
[+] Loaded 1 path(es) and 2 unique word(s) from /robots.txt
==> ENTRY: /api/v1

[*] Calculated default not found http code for this folder is 200 with content size 836
==> DIRECTORY: http://api1.webapiexploitation.com.br/api/v1/ (CODE:200|SIZE:404)
==> DIRECTORY: http://api1.webapiexploitation.com.br/.htaccess/ (CODE:403|SIZE:153)
Testing [176/595]: http://api1.webapiexploitation.com.br/arquivos/
```

Executando o turbosearch com o parâmetro `--report-to`, todas as URLs encontradas são encaminhadas para o Burp, tendo então, o mapa das mesmas. Algumas delas com o caminho encontrado no robots.txt

```
[x]--[m4v3r1ck@M4v3r1ck-2]--[~]
└─ $turbosearch -t http://api1.webapiexploitation.com.br -w wl.txt --report-to "http://127.0.0.1:8080"
```

The screenshot shows the Burp Suite interface. In the top navigation bar, 'Target' is selected. Below it, the 'Site map' tab is active, showing a tree view of the website structure under 'http://api1.webapiexploitation.com.br'. The tree includes nodes for .htaccess, /, api, assets, main.js, media, polyfills.js, robots.txt, runtime.js, styles.css, and vendor.js. To the right of the tree, a table lists various API endpoints with their details: Host, Method, URL, Params, Status, Length, MIME type, Title, Comment, and Time requested. Several entries are visible, such as '/api/v1/' (Status 200, 1069 bytes, HTML, Vulnerable API - Web Api...), '/api/v1/help/' (Status 200, 785 bytes, JSON), and '/robots.txt' (Status 200, 321 bytes, text, Loading...). At the bottom of the table, there's a note about 403 Forbidden responses. The status bar at the bottom of the Burp window displays the URL 'api1.webapiexploitation.com.br/robots.txt'. The robots.txt file content is shown in the bottom right pane:

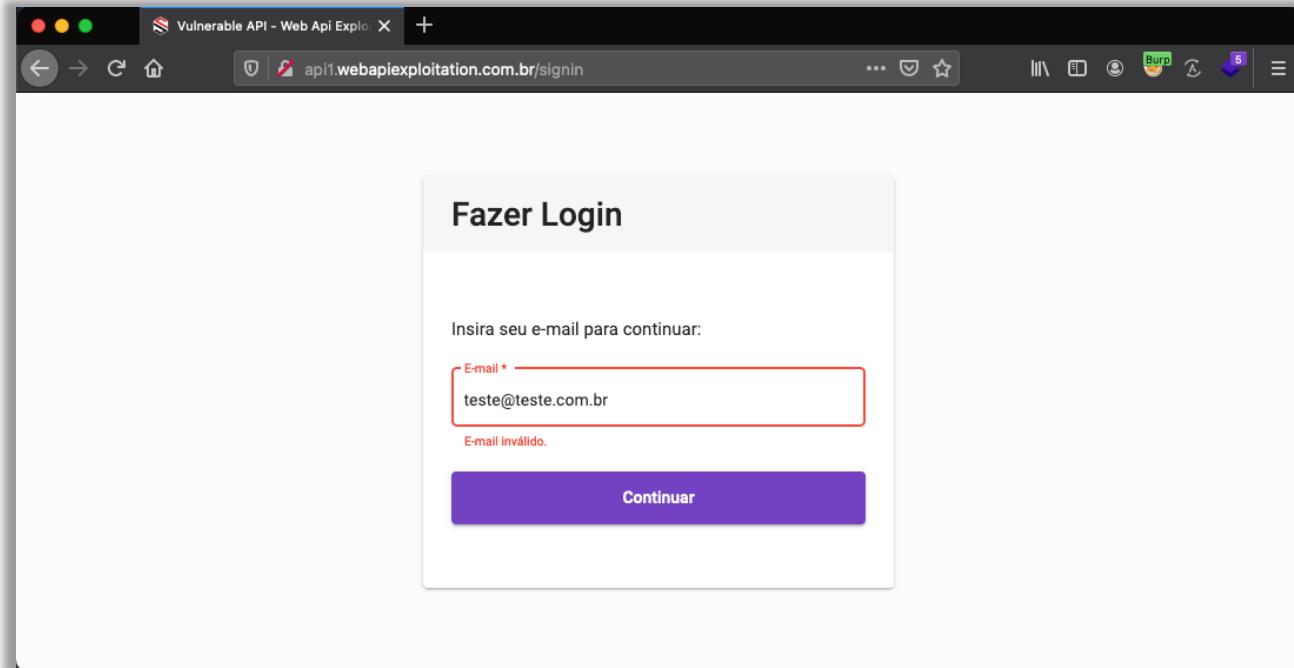
```
# robots.txt: Treinamento Web API Exploitation
# Sec4Us

User-agent: *

Disallow: /api/v1
```

Burp

Acessando o site do treinamento temos um campo para inserção de um e-mail. Colocando um e-mail qualquer de teste e clicando em continuar recebemos uma mensagem de erro.



Analisando o tráfego no Burp vemos a presença de uma chamada de API em <http://api1.webapiexploitation.com.br/api/v1/users/verify/teste@teste.com.br>

Burp Suite Community Edition v2020.12.1 - Temporary Project

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title	Comment	TLS	IP	Cookies	Time
1066	http://api1.webapiexploitation.co...	GET	/signin			200	1089	HTML		Vulnerable API - Web Api...			192.168.15.14		13:21:08 24.J...
1067	http://api1.webapiexploitation.co...	GET	/bundle.js			304	179	script	js				192.168.15.14		13:21:08 24.J...
1068	http://api1.webapiexploitation.co...	GET	/polyfills.js			304	177	script	js				192.168.15.14		13:21:08 24.J...
1069	http://api1.webapiexploitation.co...	GET	/Vendor.js			304	178	script	js				192.168.15.14		13:21:08 24.J...
1070	http://api1.webapiexploitation.co...	GET	/main.js			304	177	script	js				192.168.15.14		13:21:08 24.J...
1118	http://api1.webapiexploitation.co...	GET	/api/v1/users/verify/teste@teste.com.br			200	400	JSON	br				192.168.15.14		13:48:21 24.J...

Request

```
Pretty Raw \n Actions
GET /api/v1/users/verify/teste@teste.com.br HTTP/1.1
Host: api1.webapiexploitation.com.br
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.16; rv:84.0) Gecko/20100101 Firefox/84.0
Accept: application/json, text/plain, */*
Accept-Language: pt-BR,pt;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://api1.webapiexploitation.com.br/signin
Cookie: _ga_BB4BLH05FD=GS1.1.1611439323.14.0.1611439323.0; _ga=GA1.1.239309494.1609611931
```

Response

```
Pretty Raw Render \n Actions
HTTP/1.1 200 OK
Server: nginx/1.17.5
Date: Sun, 24 Jan 2021 16:48:21 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 11
Connection: close
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept, x-access-token, Authorization
Access-Control-Allow-Methods: GET, POST, PUT, DELETE, PATCH, OPTIONS
{
  "email_validated":false
}
```

Analisando o tráfego no Burp vemos a presença de uma chamada de API em:

<http://api1.webapiexploitation.com.br/api/v1/users/verify/teste@teste.com.br>.

The screenshot shows the Burp Suite interface with the following details:

HTTP history tab: Shows a list of 1118 requests. The last request, row 1118, is highlighted with a red border. The URL for this request is `/api/v1/users/verify/teste@teste.com.br`.

Request pane (bottom-left): Displays the raw HTTP request. The URL is highlighted with a red box.

```
1 GET /api/v1/users/verify/teste@teste.com.br HTTP/1.1
Host: api1.webapiexploitation.com.br
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.16; rv:84.0) Gecko/20100101 Firefox/84.0
Accept: application/json, text/plain, */*
Accept-Language: pt-BR,pt;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://api1.webapiexploitation.com.br/signin
Cookie: _ga_BB4BLH05FD=GS1.1.1611439323.14.0.1611439323.0; _ga=GAI.1.239309494.1609611931
```

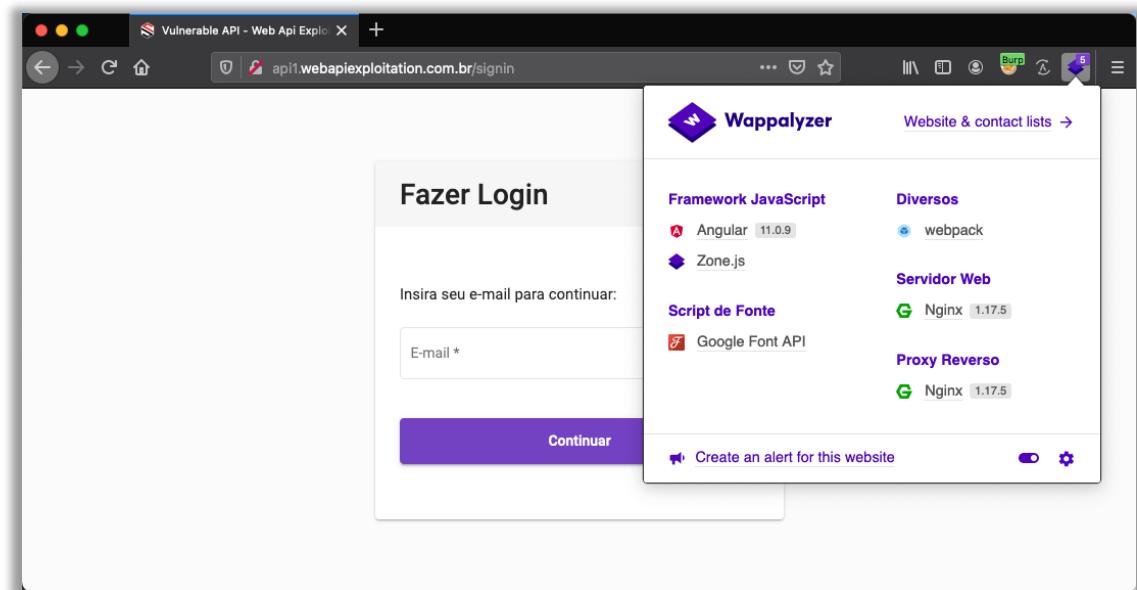
Response pane (bottom-right): Displays the raw HTTP response. The JSON payload is highlighted with a red box.

```
1 HTTP/1.1 200 OK
2 Server: nginx/1.17.5
3 Date: Sun, 24 Jan 2021 16:48:21 GMT
4 Content-Type: application/json; charset=utf-8
5 Content-Length: 25
6 Connection: close
7 Access-Control-Allow-Origin: *
8 Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept, x-access-token, Aut
9 Access-Control-Allow-Methods: GET, POST, PUT, DELETE, PATCH, OPTIONS
10
11 { "email_validated":false }
```

JavaScript

Muitos sites tem adotado uma estratégia onde o FrontEnd é somente uma página estática (utilizando frameworks como Angular) que consome uma API como back-end.

Acessando e analisando com o plugin Wappalyzer (por exemplo) podemos ver que o site foi desenvolvido utilizando o Angular.



Estes frameworks tem a característica de empacotar e ofuscar todas as chamadas de API em um único JavaScript. Analisando o tráfego web do nosso site observamos a presença deste JavaScript.

Analisando o JavaScript vemos a presença da URL da API. E um pouco mais a frente a definição dos endpoints da API.

```
363 // The list of file replacements can be found in angular.json
370 const environment = {
371   apiUrl: "http://api1.webapiexploitation.com.br/api/v1",
372   production: false,
373 };
374 /*
```

```
1092 class UserService {
1093   constructor(http) {
1094     this.http = http;
1095     this.basePath = _environments_environment__WEBPACK_IMPORTED_MODULE_1__["environment"].apiUrl + "/users";
1096   }
1097   getUsers() {
1098     const url = this.basePath + "/list";
1099     return this.http.get(url);
1100   }
1101   getUser(id) {
1102     const url = this.basePath + "/" + id;
1103     return this.http.get(url);
1104   }
1105   createUser(user) {
1106     const url = this.basePath + "/create";
1107     return this.http.post(url, user);
1108   }
1109   updateUser(id, payload) {
1110     const url = this.basePath + "/" + id;
1111     return this.http.patch(url, payload);
1112   }
1113   uploadPhoto(id, payload) {
1114     const url = this.basePath + "/" + id + "/profile";
1115     return this.http.post(url, payload);
1116   }
1117 }
```

Documentação

Como a API foi originalmente criada para integração entre sistemas, é muito comum que essas APIs sejam públicas, e assim necessitando que o proprietário da API disponibilize uma documentação da mesma.

Como já vimos anteriormente as APIs baseadas em SOAP geralmente já trazem nativamente esta documentação, mas os outros tipos de API dependem do Framework de desenvolvimento ou a geração separada da documentação por parte do desenvolvimento.

Mesmo em cenários onde a API é privada, muitos Frameworks como o ASP.NET, ao gerarem o pacote de publicação da API geram automaticamente uma documentação da mesma.

Observando o resultado da busca com o turbosearch podemos ver que ele reportou essa URL de documentação `/api/v1/help`. Ao acessar essa URL temos a documentação completa da API.

The screenshot shows a terminal window on the left and a browser window on the right. The terminal window displays the output of a turbosearch command:

```
[+] Entering directory: http://api1.webapiexploitation.com.br/api/v1
[*] Calculated default not found http code for this folder is 404 with content size 157
==> DIRECTORY: http://api1.webapiexploitation.com.br/api/v1/.htaccess/ (CODE:404)
==> DIRECTORY: http://api1.webapiexploitation.com.br/api/v1/users/ (CODE:200)
==> DIRECTORY: http://api1.webapiexploitation.com.br/api/v1/help/ (CODE:200)
```

The browser window shows the "Web API Exploitation" documentation page. The title bar says "Web API Exploitation - Docs". The page content includes:

- API** (button): Resgata informações gerais da API
- Autenticação** (button): Autentica o usuário
- Usuários** (button): Alteração de senha, Atualiza os dados do usuário, Cria um novo usuário, Envia arquivo de perfil, Exclui o usuário, Lista todos os usuários, Recuperação de senha, Retorna Informações do Usuário, Verifica a existência do e-mail
- API - Resgata informações gerais da API** (button): 1.0.0
Resgata informações gerais da API.
GET
`http://api1.webapiexploitation.com.br/api/v1/`
Exemplo de utilização:
`curl -i http://api1.webapiexploitation.com.br/api/v1/`
Success 200

Buscadores

Outros aliados bem importantes no processo de reconhecimento são os buscadores que podem nos trazer informações valiosas para encontrar e conhecer a estrutura da API a ser testada.

- Exemplos:
 - Google dorks: **inurl:.asmx filetype:asmx** que nos traz uma listagem de APIs escritas em ASP.NET no padrão asmx (em geral SOAP).
 - Bing: **"/api/" contains:json** nos retorna uma listagem de APIs ou documentação de APIs
 - Shodan: **ssl:"api*" country:"BR"** buscamos todos os registros no Brasil em que no nome do certificado digital contém a palavra api.

Web Crawling

Web Crawling é o processo de busca com a utilização de lista de palavras (metodologia de força-bruta), ou seja, é passado uma listagem de palavras a serem testadas e o buscador testa uma a uma. Em nosso treinamento utilizaremos o TurboSearch, do qual já vimos anteriormente.



Neste laboratório iremos:

- Navegar no site <http://api1.webapiexploitation.com.br>;
- Verificar no Burp todos as requisições geradas pelo site;
- Listar os endpoints da API no JavaScript;
- Criação de uma lista de palavras personalizadas;
- Web Crawling utilizando a lista de palavras gerada;
- Comparar o resultado encontrado com a documentação da API.

Listando os usuários

Utilizando uma lista de palavras extremamente pequena, porém assertiva, pois utilizamos os passos de enumeração anteriores juntamente com a informação presente no robots.txt. Segue abaixo a lista de palavras utilizada:

- verify
- users
- list
- create
- profile

Vemos um resultado assertivo trazendo uma URL crítica
`/api/v1/users/list`

```
[m4v3r1ck@m4v3r1ck-2:~]$ turbosearch -t http://api1.webapiexploitation.com.br -w wl.txt

HHHHHH      +-+HHH
HHHHHH      +--+HHH
HHHHHH      +----+
+-+-----+-----+
++|-----+-----+
HHHHHH      +----+
HHHHHH      +---+HH
HHHHHH      +-+HHH

Turbo Search v0.1.9 by Helvio Junior
automated url finder
https://github.com/helviojunior/turbosearch

[+] Startup parameters
  command line: /usr/local/bin/turbosearch -t http://api1.webapiexploitation.com.br -w wl.txt
  target: http://api1.webapiexploitation.com.br
  tasks: 16
  request method: GET
  word list: wl.txt
  forward location redirects: yes
  case insensitive search: no
  start time 2021-01-25 00:35:04
  duplicate 6 words

[+] Connection test against http://api1.webapiexploitation.com.br OK! (CODE:200|SIZE:836)

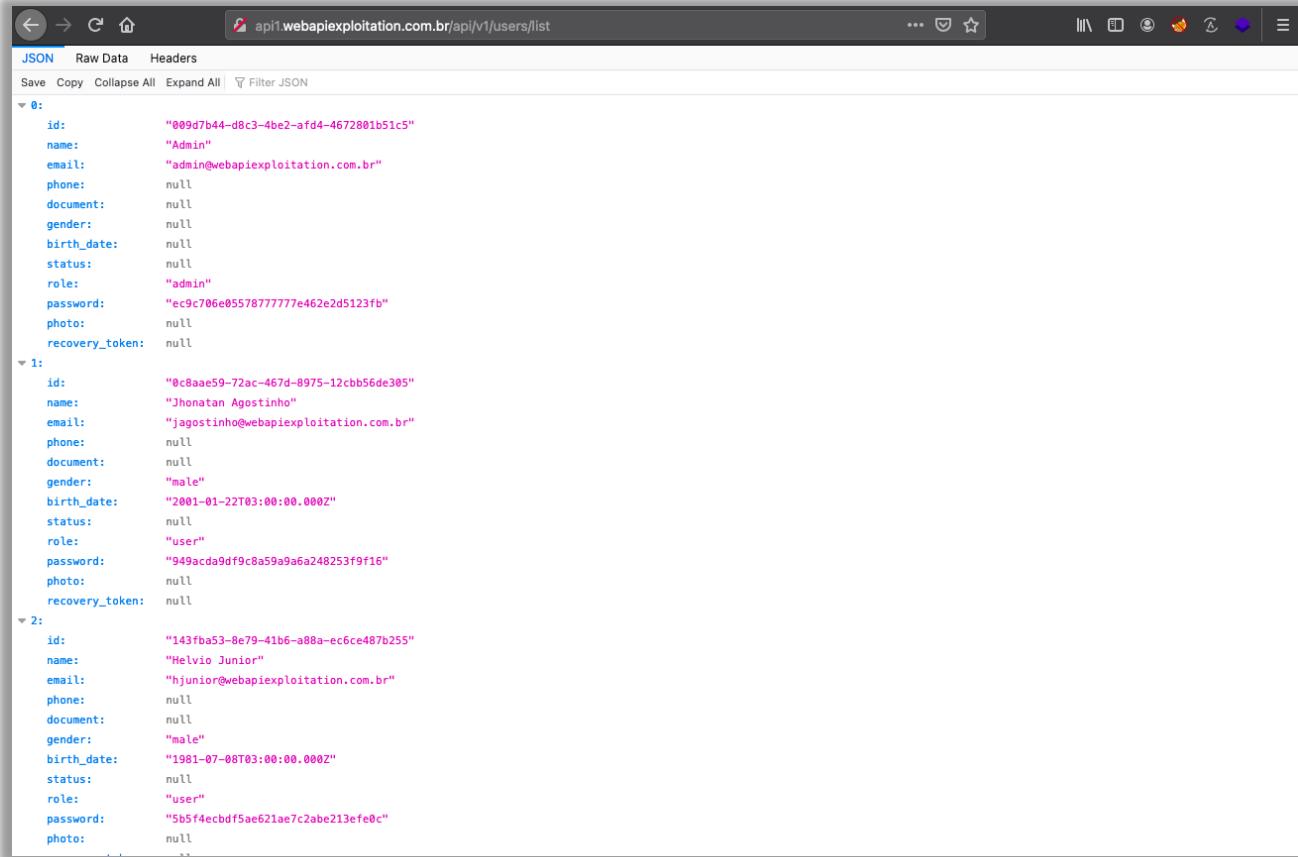
[+] Scanning url http://api1.webapiexploitation.com.br
[+] Getting informations from /robots.txt at api1.webapiexploitation.com.br
[+] Loaded 1 path(es) and 2 unique word(s) from /robots.txt
==> ENTRY: /api/v1

[+] Calculated default not found http code for this folder is 200 with content size 836
==> DIRECTORY: http://api1.webapiexploitation.com.br/api/v1/ (CODE:200|SIZE:404)

[+] Entering directory: http://api1.webapiexploitation.com.br/api/v1
[+] Calculated default not found http code for this folder is 404 with content size 157
==> DIRECTORY: http://api1.webapiexploitation.com.br/api/v1/users/ (CODE:200|SIZE:27)

[+] Entering directory: http://api1.webapiexploitation.com.br/api/v1/users
[*] Calculated default not found http code for this folder is 401 with content size 42
==> DIRECTORY: http://api1.webapiexploitation.com.br/api/v1/users/api/ (CODE:401|SIZE:42)
==> DIRECTORY: http://api1.webapiexploitation.com.br/api/v1/users/v1/ (CODE:401|SIZE:42)
==> DIRECTORY: http://api1.webapiexploitation.com.br/api/v1/users/users/ (CODE:401|SIZE:42)
==> DIRECTORY: http://api1.webapiexploitation.com.br/api/v1/users>List/ (CODE:200|SIZE:2895)
==> DIRECTORY: http://api1.webapiexploitation.com.br/api/v1/users/verify/ (CODE:401|SIZE:42)
==> DIRECTORY: http://api1.webapiexploitation.com.br/api/v1/users/profile/ (CODE:401|SIZE:42)
==> DIRECTORY: http://api1.webapiexploitation.com.br/api/v1/users/create/ (CODE:401|SIZE:42)
```

Acessando a URL obtivemos a listagem de todos os usuários da API com informações críticas.



The screenshot shows a browser window with the URL `api1.webapiexploitation.com.br/api/v1/users/list`. The page displays a JSON array of three user objects. Each user object contains fields such as id, name, email, phone, document, gender, birth_date, status, role, password, photo, and recovery_token. The 'password' field is notably highlighted in red, indicating it is a sensitive piece of information.

```
JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON
0:
  id: "809d7b44-d8c3-4be2-afad4-4672801b51c5"
  name: "Admin"
  email: "admin@webapiexploitation.com.br"
  phone: null
  document: null
  gender: null
  birth_date: null
  status: null
  role: "admin"
  password: "ec9c706e0557877777e462e2d5123fb"
  photo: null
  recovery_token: null
1:
  id: "0c8aae59-72ac-467d-8975-12ccb56de305"
  name: "Jhonatan Agostinho"
  email: "jagostinho@webapiexploitation.com.br"
  phone: null
  document: null
  gender: "male"
  birth_date: "2001-01-22T03:00:00.000Z"
  status: null
  role: "user"
  password: "949acda9df9c8a59a9a6a248253f9f16"
  photo: null
  recovery_token: null
2:
  id: "143fba53-8e79-41b6-a88a-ec6ce487b255"
  name: "Helvio Junior"
  email: "hjunior@webapiexploitation.com.br"
  phone: null
  document: null
  gender: "male"
  birth_date: "1981-07-08T03:00:00.000Z"
  status: null
  role: "user"
  password: "5b5f4ecbd5ae621ae7c2abe213efe0c"
  photo: null
```

Informações tradicionais web

Vale lembrar que API nada mais é do que um sistema Web, sendo assim todas as outras técnicas tradicionais podem nos ajudar. Isto posto, deixo outros questionamentos:

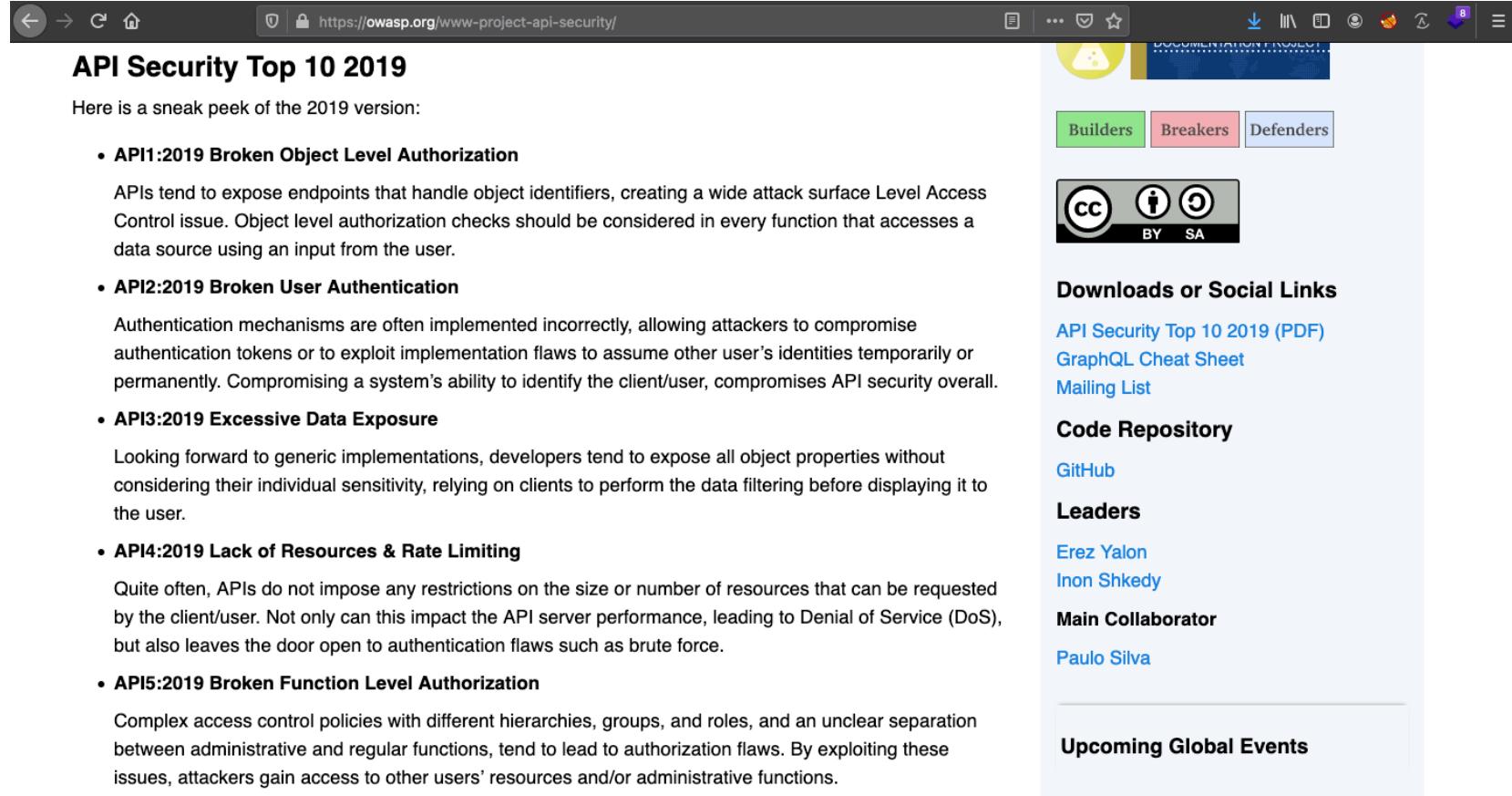
- Como o desenvolvedor versionou a API?
 - /v1/, /v2, api1.domínio.com.br, api2.domínio.com.br
- Qual a linguagem de programação utilizada?
 - Banner do servidor, erros, documentação e etc...
- Qual o DB utilizado?
 - Erros, documentação e etc...
- Como os clientes se autenticam na API?
 - Documentação e etc...

Para todos os questionamentos temos a possibilidade de:

- Busca em repositórios como o GitHub
- Busca por dúvidas em sites como o StackOverflow
- Informações públicas (Linkedin / vagas de emprego)
- Engenharia Social

- OWASP API Security
 - <https://owasp.org/www-project-api-security/>
 - https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html
- OWASP Top 10 Web
 - <https://owasp.org/www-project-top-ten/>
- OWASP Testing Guide
 - https://owasp.org/www-project-web-security-testing-guide/assets/archive/OWASP_Testing_Guide_v4.pdf
 - <https://github.com/tanprathan/OWASP-Testing-Checklist>

- API é um sistema Web, podendo sofrer os mesmos tipos de ataque vistos no OWASP Top 10 2017;
- Utilize os guias da OWASP para realização dos testes também em API;
- Muitas empresas tratam a segurança da API como segurança por obscuridade, em outras palavras, procure que você certamente achará algo não documentado;
- Pelo menos as 5 primeiras categorias do OWASP API Security Top 10 2019 se referem a algum tipo de exposição de dados, falha no controle de autorização ou autenticação exigindo do analista criatividade nos testes de verificação de furo nas regras de negócio, autenticação e autorização.



The screenshot shows a web browser window with the URL <https://owasp.org/www-project-api-security/>. The page displays the "API Security Top 10 2019" list. On the right side of the page, there is a sidebar with various links and icons.

API Security Top 10 2019

Here is a sneak peek of the 2019 version:

- API1:2019 Broken Object Level Authorization**

APIs tend to expose endpoints that handle object identifiers, creating a wide attack surface Level Access Control issue. Object level authorization checks should be considered in every function that accesses a data source using an input from the user.
- API2:2019 Broken User Authentication**

Authentication mechanisms are often implemented incorrectly, allowing attackers to compromise authentication tokens or to exploit implementation flaws to assume other user's identities temporarily or permanently. Compromising a system's ability to identify the client/user, compromises API security overall.
- API3:2019 Excessive Data Exposure**

Looking forward to generic implementations, developers tend to expose all object properties without considering their individual sensitivity, relying on clients to perform the data filtering before displaying it to the user.
- API4:2019 Lack of Resources & Rate Limiting**

Quite often, APIs do not impose any restrictions on the size or number of resources that can be requested by the client/user. Not only can this impact the API server performance, leading to Denial of Service (DoS), but also leaves the door open to authentication flaws such as brute force.
- API5:2019 Broken Function Level Authorization**

Complex access control policies with different hierarchies, groups, and roles, and an unclear separation between administrative and regular functions, tend to lead to authorization flaws. By exploiting these issues, attackers gain access to other users' resources and/or administrative functions.

DOCUMENTATION PROJECT

Builders **Breakers** **Defenders**

 CC BY SA

Downloads or Social Links

[API Security Top 10 2019 \(PDF\)](#)
[GraphQL Cheat Sheet](#)
[Mailing List](#)

Code Repository

[GitHub](#)

Leaders

[Erez Yalon](#)
[Inon Shkedy](#)

Main Collaborator

[Paulo Silva](#)

Upcoming Global Events



Web API Exploitation

<https://sec4us.com.br/web-api-exploitation/>

