API Security Project

OWASP Projects’ Showcase

Sep 12, 2019
Founders and Sponsors

CHECKMARX

SALT
Project Leaders

Erez Yalon
- Director of Security Research @ Checkmarx
- Focusing on Application Security
- Strong believer in spreading security awareness

Inon Shkedy
- Head of Research @ Traceable.ai
- 7 Years of research and pentesting experience
- I’ve grown up with APIs
What is API?

API stands for:

**Application Programming Interface**

"An Application Programming Interface (API) is an interface or communication protocol between a client and a server intended to simplify the building of client-side software. It has been described as a “contract” between the client and the server, such that if the client makes a request in a specific format, it will always get a response in a specific format or initiate a defined action."

Who Uses APIs?

Every Modern application:

- Mobile
- IoT
- B2B
- Serverless
- Cloud
- Single Page Application
API Security

==

API-Based Apps Security
Today’s Agenda

- How APIs-Based apps are different? Why deserve their own project?
- Roadmap
- API Security Top 10 RC
- Acknowledgements
- Call for contributors
How API Based Apps are Different?

Client devices are becoming varied and stronger

Logic moves from Backend to Frontend (together with some vulnerabilities)
Traditional vs. Modern

Traditional Application

Modern Application

Get

Get HTML

API Get

Raw

OWASP GLOBAL APPSEC - AMSTERDAM
Traditional vs. Modern

Less abstraction layers

Client and server (and DB) speak the same JSON language
How API Based Apps are Different?

- The server is used more as a proxy for data
- The rendering component is the client, not the server

- Clients consume raw data
- APIs expose the underlying implementation of the app
- The user’s state is usually maintained and monitored by the client
- More parameters are sent in each HTTP request (object ID’s, filters)
How API Based Apps are Different?

- The REST API standard
  - Standardized & generic
  - Predictable entry points
  - One entry point (URL) can be used for multiple purposes
How API Based Apps are Different?

The good news

Traditional vulnerabilities are less common in API-Based apps:

• SQLi – Increasing use of ORMs
• CSRF – Authorization headers instead of cookies
• Path Manipulations – Cloud-Based storage
• Classic IT Security Issues - SaaS
What About Dev(Sec)Ops?

APIs change all the time

It takes just a few clicks to spin up new APIs (hosts). Too easy!

APIs become hard to track:
- Shadow APIs
- Old Exposed APIs
Roadmap – Planned Projects

- API Security Top 10
- API Security Cheat Sheet
- crAPI (Completely Ridiculous API - an intentionally vulnerable API project)
## Roadmap

<table>
<thead>
<tr>
<th></th>
<th>Top 10</th>
<th>Cheat Sheet</th>
<th>crAPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019 Q1</td>
<td>Prepare</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2019 Q2</td>
<td>Kick-Off</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2019 Q3</td>
<td>V1.0</td>
<td>Kick-Off</td>
<td>Prepare</td>
</tr>
<tr>
<td>2019 Q4</td>
<td>Collaborate</td>
<td>Kick-Off</td>
<td></td>
</tr>
<tr>
<td>2020 Q1</td>
<td>V1.0</td>
<td>Collaborate</td>
<td></td>
</tr>
<tr>
<td>2020 Q2</td>
<td></td>
<td>V1.0</td>
<td></td>
</tr>
</tbody>
</table>
The creation process of the Top10

- Internal knowledge and experience
- Internal data collection (Bug bounties reports, published incidents, etc.)
- Call for Data
- Call for comments
API Security Top 10

- **A1**: Broken Object Level Authorization
- **A2**: Broken Authentication
- **A3**: Excessive Data Exposure
- **A4**: Lack of Resources & Rate Limiting
- **A5**: Broken Function Level Authorization
- **A6**: Mass Assignment
- **A7**: Security Misconfiguration
- **A8**: Injection
- **A9**: Improper Assets Management
- **A10**: Insufficient Logging & Monitoring
Authz in APIs - The Challenge

• Decentralized Mechanism

<table>
<thead>
<tr>
<th>Object Level</th>
<th>Function Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code (Almost every controller)</td>
<td>Code, Configuration, API-gateway</td>
</tr>
</tbody>
</table>

• Complex Users & Roles Hierarchies

<table>
<thead>
<tr>
<th>Riders</th>
<th>Drivers</th>
<th>Admins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hugo</td>
<td>Jon</td>
<td>Inon</td>
</tr>
<tr>
<td>Sub #1</td>
<td></td>
<td>Shredder</td>
</tr>
<tr>
<td>Sub #2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A1 - BOLA (Broken Object-Level Authorization)

POST api/trips/rate_trip
{"trip_id":718492, "rate":5}

UPDATE trips ...
WHERE ID = 718492
BOLA - Why Not IDOR

• **IDOR** - Insecure Direct Object Reference
• C00L name, not accurate
• The problem is not about the IDs!
BOLA - Solutions that **don’t** solve the problem

- GUIDs instead of numbers
- Indirect Object Reference
- Relying on IDs from JWT tokens

BOLA - Solutions that **solve** the problem

- Good authorization mechanism
- Make sure that developers actually use it in every controller
BOLA - Uber - Full Account Takeover

Request

POST /marketplace/__rpc?rpc=getConsentScreenDetails HTTP/1.1
Host: bonjour.uber.com
Connection: close
Content-Length: 67
Accept: application/json
Origin: [https://bonjour.uber.com](https://bonjour.uber.com)
x-csrf-token: xxxx
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit
DNT: 1
Content-Type: application/json
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: xxxx

{“language”:”en”,”userUuid”:”xxxx-776-4xxxx1bd-861a-837xxxx604ce”}

Response

{“status”:”success”,”data”:{“data”:{“language”:”en”,”userUuid”:”xxxxx1e”}},
”getUser”:{
“uuid”:”xxxxxxc5f7371e”,”firstname”:”Maxxxx”,”lastname”:”XXXX”,”role”:”PARTNER”,”languageId”:1,”countryId”:77,”mobile”:null,”mobileToken”:1234,”mobileCountryId”:77,”mobileCountryCode”:”+91”,”hasAmbiguousMobileCountry”:false,”lastConfirmedMobileCountryId”:77,”email”:”xxxx@gmail.com”,”emailToken”:”xxxxxxx”,”

Found by Anand Prakash,
AppSecure
A2 - Broken Authentication

**Lack of protection:**
- Account lockout
- Captcha
- Brute Force attacks

**Misconfiguration:**
- JWT allows \{"alg":"none"\}
- Tokens don’t expire
- etc..

**Rate Limiting (A4)**

<table>
<thead>
<tr>
<th>Function</th>
<th>EXTRA Protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>forgot_password</td>
<td></td>
</tr>
<tr>
<td>web_login</td>
<td></td>
</tr>
<tr>
<td>get_location</td>
<td></td>
</tr>
<tr>
<td>update_picture</td>
<td></td>
</tr>
</tbody>
</table>
A2 - Facebook - Full Account Takeover

Lockout mechanism (after 10 attempts)

Facebook.com

Beta.facebook.com

Mbasic.beta.facebook.com

Vulnerable request:

POST /recover/as/code/ HTTP/1.1
Host: beta.facebook.com

lsd=AVoywo13&n=XXXX

(5 Digits Reset Password Token)

100,000 options

Found by Anand Prakash, AppSecure
A3 - Excessive Data Exposure

• APIs expose sensitive data of other Users by design
A3 - Excessive Data Exposure

Filtering sensitive information on the client side == BAD IDEA!!
A3 - Why?

- API Economy + REST Standard == Generic Endpoints
- “tojson” functions from ORM / Model
- Developers don’t think who’s the consumer
Recent Example - “3fun” app

Found by Alex Lomas, [Pen Test Partners](http://www.pentestpartners.com)
<table>
<thead>
<tr>
<th>#</th>
<th>Host</th>
<th>Method</th>
<th>URL</th>
<th>Params</th>
<th>Edited</th>
<th>Status</th>
<th>Length</th>
<th>MIME type</th>
</tr>
</thead>
<tbody>
<tr>
<td>322</td>
<td><a href="https://www.go3fun.co">https://www.go3fun.co</a></td>
<td>POST</td>
<td>/account_kit_reg</td>
<td>✓</td>
<td></td>
<td>200</td>
<td>447</td>
<td>JSON</td>
</tr>
<tr>
<td>325</td>
<td><a href="https://www.go3fun.co">https://www.go3fun.co</a></td>
<td>POST</td>
<td>/user/device_token</td>
<td>✓</td>
<td></td>
<td>200</td>
<td>198</td>
<td>JSON</td>
</tr>
<tr>
<td>326</td>
<td><a href="https://www.go3fun.co">https://www.go3fun.co</a></td>
<td>POST</td>
<td>/user/update</td>
<td>✓</td>
<td></td>
<td>200</td>
<td>265</td>
<td>JSON</td>
</tr>
<tr>
<td>327</td>
<td><a href="https://www.go3fun.co">https://www.go3fun.co</a></td>
<td>POST</td>
<td>/reset_push_badge</td>
<td>✓</td>
<td></td>
<td>200</td>
<td>198</td>
<td>JSON</td>
</tr>
<tr>
<td>329</td>
<td><a href="https://www.go3fun.co">https://www.go3fun.co</a></td>
<td>GET</td>
<td>/match_users?id=0&amp;latitude=51</td>
<td>✓</td>
<td></td>
<td>200</td>
<td>23807</td>
<td>JSON</td>
</tr>
<tr>
<td>331</td>
<td><a href="https://www.go3fun.co">https://www.go3fun.co</a></td>
<td>POST</td>
<td>/user/refresh</td>
<td>✓</td>
<td></td>
<td>200</td>
<td>788</td>
<td>JSON</td>
</tr>
<tr>
<td>334</td>
<td><a href="https://www.go3fun.co">https://www.go3fun.co</a></td>
<td>POST</td>
<td>/user/update_location</td>
<td>✓</td>
<td></td>
<td>200</td>
<td>479</td>
<td>JSON</td>
</tr>
<tr>
<td>338</td>
<td><a href="https://www.go3fun.co">https://www.go3fun.co</a></td>
<td>POST</td>
<td>/upload_photo</td>
<td>✓</td>
<td></td>
<td>200</td>
<td>201</td>
<td>JSON</td>
</tr>
<tr>
<td>339</td>
<td><a href="https://www.go3fun.co">https://www.go3fun.co</a></td>
<td>GET</td>
<td>/i_like_list?id=0&amp;offset=30</td>
<td>✓</td>
<td></td>
<td>200</td>
<td>201</td>
<td>JSON</td>
</tr>
<tr>
<td>340</td>
<td><a href="https://www.go3fun.co">https://www.go3fun.co</a></td>
<td>GET</td>
<td>/chatted_list</td>
<td>✓</td>
<td></td>
<td>200</td>
<td>201</td>
<td>JSON</td>
</tr>
<tr>
<td>341</td>
<td><a href="https://www.go3fun.co">https://www.go3fun.co</a></td>
<td>GET</td>
<td>/reset_push_badge</td>
<td>✓</td>
<td></td>
<td>200</td>
<td>992</td>
<td>JSON</td>
</tr>
<tr>
<td>344</td>
<td><a href="https://www.go3fun.co">https://www.go3fun.co</a></td>
<td>GET</td>
<td>/user/refresh</td>
<td>✓</td>
<td></td>
<td>200</td>
<td>198</td>
<td>JSON</td>
</tr>
<tr>
<td>348</td>
<td><a href="https://www.go3fun.co">https://www.go3fun.co</a></td>
<td>GET</td>
<td>/matchedList?id=0&amp;offset=30</td>
<td>✓</td>
<td></td>
<td>200</td>
<td>201</td>
<td>JSON</td>
</tr>
</tbody>
</table>

Request vs Response:

```
},
  "latitude": "51",
  "membership": "2",
  "birthday": "1997-",
  "sex_orient": "4",
  "gender": "1",
  "longitude": "-0.1",
  "photo_verified_status": "1",
  "active": "0",
  "partner_sex_orient": "0",
  "liked_me": "70",
  "settings": {
    "show_online_status": "1",
    "show_distance": "1"
  },
  "username":",
  "user_id": 417,
  "about_me": "Kinky and attractive french financier open to many things ..."
},

"last_login": "2019-06-24 20:21:12",
"private_photos": {
  "icon": "https://s3.amazonaws.com/3fun/821/821_small.jpg",
  "photo_id": 38
```
A4 - Lack of Resources & Rate Limiting

• Might lead to DOS
• www.socialnetwork.com/users/list?limit=99999999
A5 - BFLA
(Broken Function Level Authorization)

Admin

DELETE /users/717

Attacker w/ Driver User

DELETE /users/717

Admin API
Riders API
Drivers API

API
## Why in APIs

<table>
<thead>
<tr>
<th></th>
<th>Fetch User’s Profile (not sensitive function)</th>
<th>Delete user (admin function)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Traditional App</strong></td>
<td>GET /app/users_view.aspx?user_id=1337</td>
<td>POST app/admin_panel/users_mgmt.aspx action=delete&amp;user_id=1337</td>
</tr>
<tr>
<td><strong>API</strong></td>
<td>GET /api/users/1337</td>
<td>DELETE /api/users/1337</td>
</tr>
</tbody>
</table>

**HARD to predict :(

**Very Predictable**
Function Level Authorization

- Can be implemented in different components:
  - Code
  - Configuration
  - API Gateway

- Different Roles:
  - Admins / Super-admins / supervisors / riders / drivers
@uzsunny reported that by creating two partner accounts sharing the same business email, it was possible to be granted "collaborator" access to any store without any merchant interaction.

“The code did not properly check what type the existing account was”

Found by uzsunny
$20,000 bounty on Hackerone
A6 - Mass Assignment
“Create_user” flow in traditional apps

```java
User new_user = User();
new_user.firstName = Request.Query["fname"];  
new_user.lastName = Request.Query["lname"];  
new_user.pass = Request.Query["pass"];  
new_user.Save();
```

ORM
```
{first_name=Inon
last_name=shkedy
pass=123456}
```

Create_user
fname=inon&
lname=shkedy&
pass=123456
A6 - Mass Assignment

POST /users/create
{“user”:{“lname”:”Inon”,“fname”:”shkedy”,“pass”:”123456”}}
A6 - Mass Assignment

POST /api/users/new
{"username":"Inon", "pass":"123456"}

POST /api/users/new
{"username":"Inon", "pass":"123456", "role":"admin"}
A6 - Why in APIs

• Mass Assignment isn’t a new vulnerability.
• Easier to exploit in APIs though
• Don’t guess object properties, just find a GET method that returns them :)

GET /v1/user/video_files

200 OK

```json
{
"id": 371,
"name": "clip.mp4",
"conversion_params": "-v codec h264 &\& format C:"
}
```

PUT /v1/videos/371

```json
{
"name": "clip.mp4",
"conversion_params": "-v codec h264 &\& format C:"
}
```
A6 - Example

Users can enable API access for free via mass assignment

POST /accounts/<account_id>.json

account[first_name]="Evil" &
account[allow_api_access]=true

Found by
James Kettle,
Port Swigger
A7 - Security Misconfiguration

- Lack of CSRF / CORS protection
- Lack of security related HTTP headers
- Unnecessary exposed HTTP methods
- Weak encryption
- Etc...
A8 - Injection
Why from A1 to A8?

• First of all, ask yourself - why injection was A1?

• SQLi much less common:
  • ORMs
  • Gazillion of security products that solve them
  • Use of NoSQL

• NoSQL Injection are a thing, but are usually not as severe / common
A9 - Improper Asset Management

API endpoints with no documentation

API

/v1/get_user
/v2/update_location
/v0/b2b_old/export_all_users

Unknown API hosts

payment-api.acme.com
mobile-api.acme.com
qa-3-old.acme.com

Developers

DevOps
A9 - Why in APIs?

• APIs change all the time because of CI/CD

• Cloud + deployment automation (K8S) ==
  Too easy to spin up a new API host
A10 - Insufficient Logging & Monitoring

• Same as A10 (2017)
Call for Discussions

Mailing List
https://groups.google.com/a/owasp.org/d/forum/api-security-project
Call for Contributions

GitHub Project
https://github.com/OWASP/API-Security/blob/develop/CONTRIBUTING.md
https://www.owasp.org/index.php/OWASP_API_Security_Project

https://github.com/OWASP/API-Security

QUESTIONS?