API Security Project
Top-10 Release Candidate

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
Today’s Agenda

- How APIs-Based apps are different? Why deserve their own project?
- Roadmap
- Call for contributors
- **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 HTML

Get

API Get

Raw

elastic
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

APIs become hard to track:
• Shadow APIs
• Old Exposed APIs

It takes just a few clicks to spin up new APIs (hosts). Too easy!
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>Year</th>
<th>Quarter</th>
<th>Top 10</th>
<th>Cheat Sheet</th>
<th>crAPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019</td>
<td>Q1</td>
<td>Prepare</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2019</td>
<td>Q2</td>
<td>Kick-Off</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2019</td>
<td>Q3</td>
<td>V1.0</td>
<td>Kick-Off</td>
<td>Prepare</td>
</tr>
<tr>
<td>2019</td>
<td>Q4</td>
<td></td>
<td>Collaborate</td>
<td>Kick-Off</td>
</tr>
<tr>
<td>2020</td>
<td>Q1</td>
<td></td>
<td>V1.0</td>
<td>Collaborate</td>
</tr>
<tr>
<td>2020</td>
<td>Q2</td>
<td></td>
<td></td>
<td>V1.0</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
Authorization in APIs - The Challenge

- Code (Almost in every controller)
- Configuration or Code

A1 - Object Level
A5 – Function Level

Decentralized Mechanism
Complex Hierarchies

- Admins
- Supervisors
- Users
- Hugo
- Bob
- Inon

- Bob
- Sub user #1
- Sub user #2

• Sub users
• Users with different roles
A1 – BOLA (Broken Object Level Authorization)

Get my document which number is “1000” please!
Of course!

Get the document which number is “1002” please!
Hey! Don’t mention it!

From Sam Houston, Bugcrowd
A1 – BOLA (Broken Object Level Authorization)
Why is it so common?

• The **attack surface** is much **wider**
  • APIs receive more IDs, because clients maintain the user's state

• No security solution that solves the problem
Why not "IDOR"?

- "IDOR" - Insecure Direct Object Reference is a cool name
- It's not accurate / indicative enough
- The name "IDOR" hints that the object reference (ID) should be indirect (e.g.: a salted hash map)
  - What would happen if you asked your developers to implement “Indirect” mechanism in every place that receives ID?
Illustration – you asked your developers to implement an “Indirect Object Reference Mechanism” to solve IDORs in the code.

- The problem is not the Object Reference, but a lack of authorization -
Accessing 2 million Verizon Pay Monthly contracts
How I could access the personal information of 2 million Verizon Wireless customers due to 1 very simple mistake

After a quick check, I learnt that 1310000000 was the lowest contract number that could be viewed and 1311999999 was the highest. That means that there was information of around 2 million Verizon Pay Monthly customers exposed.

Found by Daley Bee
• A2 – Broken Authentication

Why is it so common?

• Authentication endpoints are exposed to anyone by design.

• Software/security engineers have misconceptions.
  • OAuth isn't authentication
  • API keys should not be used for user's authentication

• Multiple authentication flows in modern apps
  • IoT / Mobile / Legacy / Deep links with credentials, etc..
A2 – Broken Authentication

Lack of protection

Assets that need to be protected

- Extra Protection
  - ForgotPassword
  - Login
  - MobileLogin

- Rate Limiting (A4)
  - UpdateLocation
  - EditPhoto

Misimplementation

- JWT Supports {“alg”:”none”}
- Service doesn’t validate the Oauth Provider
- Passwords stored without salt
- Etc...

- Account lockout mechanism
- Captcha
- Credentials Stuffing Protection
A3 – Excessive Data Exposure

• APIs expose sensitive data of other users by design

• Why it is so common?
  • REST Standard & API economy encourage developers to implement APIs in a generic way
  • Use of generic functions as "to_json" from the Model / ORM, without thinking about who's the consumer
A3 – Excessive Data Exposure

Filtering sensitive data on the client side is always a bad idea.

Super Safe App
Bob’s Profile
Name: Bob
Role: Minion
Hobbies: Bananas

GET v1/users/profiles/717

API

Super Safe App
Bob’s Profile
Name: Bob
Role: Minion
Hobbies: Bananas

GET v1/users/profiles/717

API

Super Safe App
Bob’s Profile
Name: Bob
Role: Minion
Hobbies: Bananas

GET v1/users/profiles/717

API

Super Safe App
Bob’s Profile
Name: Bob
Role: Minion
Hobbies: Bananas

GET v1/users/profiles/717

API

Super Safe App
Bob’s Profile
Name: Bob
Role: Minion
Hobbies: Bananas

GET v1/users/profiles/717

API
A3 - Example from "3fun" app

• Found by Alex Lomas, Pen Test Partners
<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?from=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>GET</td>
<td>/user/refresh</td>
<td>✓</td>
<td></td>
<td>200</td>
<td>788</td>
<td>JSON</td>
</tr>
</tbody>
</table>

- Found by Alex Lomas, Pen Test Partners
A3 - 3Fun Hack

- Found by Alex Lomas, Pen Test Partners
A4 - Lack of Resources & Rate Limiting

• Might lead to DoS, Brute force attacks

• http://socialnetwork.com/api/v1/users?limit=999999999
A5 – BFLA (Broken Function Level Authorization)

Regular user

Legitimate API Request:
DELETE api/v2/users/717

Malicious API Request:
DELETE api/v2/users/717

API Layer

Admin API
Regular API
Public API
A5 – BFLA
Why it is common in APIs?

- Function Level Authorization can be implemented in different ways:
  - Code
  - Configuration
  - API Gateway

- Easier to detect and exploit in APIs – Endpoints are more predictable

<table>
<thead>
<tr>
<th>Action</th>
<th>Get user’s profile (Regular endpoint)</th>
<th>Delete user (Admin endpoint)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional Apps</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>APIs</td>
<td>GET /api/v2/users/1337</td>
<td>DELETE /api/v2/users/1337</td>
</tr>
</tbody>
</table>
A6 - Mass Assignment

• Modern frameworks encourage developers to use “Mass Assignment” functions

NodeJS:
var user = new User(req.body);
user.save();

Rails:
@user = User.new(params[:user])

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

POST /api/users/new
{"username": "Inon", "pass": "123456", "role": "admin"}  
Malicious Request

Might contain sensitive params that the user should not have access to
A6 – Mass Assignment

• Easier to exploit in APIs

• Instead of guessing object’s properties, just find a GET method that returns them
A7 – Security Misconfiguration

• Weak encryption
• Unnecessary exposed HTTP methods
• No CSRF protection
• Detailed errors
• Improper CORS

Bad Things
A8 – Injection
Why from A1 to A8?

• The main reason that “Injection” is currently #1 (2017), is because of SQL Injections.

• SQL Injection are not very common in modern APIs, because:
  • Use of ORMs
  • Increasing use of NoSQL

• NoSQL injection are a thing, but are usually not as common / severe
A9 – Improper Assets Management

Actually two different things

Lack of documentation

/v2/download_transactions_as_pdf
/v2/transfer_money
/v0/legacy_b2b/export_all_users

Exposed Risky Undocumented APIs

beta-api.xxx.com
payments-api.xxx.com
mobile-api.xxx.com

API Gateway

Client
A9 – Improper Assets Management

Why now?

- APIs change all the time because of CI/CD, developers are focused on delivering and not documenting

- Cloud + deployment automation (k8s) == Way too easy to spin up new APIs and machines
  - API hosts that have been forgotten
  - Complete environments that have been forgotten
    (excuse me mister, but what the heck is “qa-3-old.app.com”?)
A10 - Insufficient Logging & Monitoring

Same as 2017 A10
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?
Rate this Session

SCAN THE QR CODE TO COMPLETE THE SURVEY

API Security Project
Top-10 Release Candidate

Erez Yalon
Inon Shkedy

Thank You!