Project Leaders

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

Inon Shkedy
- Head of Research @ Salt Security
- 7 Years of research and pentesting experience
- I’ve grown up with APIs
Today’s Agenda

● How APIs based apps are different? Deserve their own project?
● Roadmap
● Creation process
● API Security Top 10
● Acknowledgements
● Call for contributors
How API Based Apps are Different?

Client devices are becoming stronger

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

Traditional Application vs. Modern Application

Get HTML vs. API Get Raw
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?

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
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>Prepare</td>
<td></td>
</tr>
<tr>
<td>2019</td>
<td>Q3</td>
<td>RC</td>
<td>Kick-Off</td>
<td>Prepare</td>
</tr>
<tr>
<td>2019</td>
<td>Q4</td>
<td>V1.0</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 Access Control
- A2: Broken Authentication
- A3: Improper Data Filtering
- A4: Lack of Resources & Rate Limiting
- A5: Missing Function/Resource Level Access Control
- A6: Mass Assignment
- A7: Security Misconfiguration
- A8: Injection
- A9: Improper Assets Management
- A10: Insufficient Logging & Monitoring
A1: Broken Object Level Access Control

- APIs consume a lot of object IDs by design:
  - URL params (/api/users/717) / Query Params (/download_file?id=111)
  - Body params / HTTP Headers (user-id:717)
- Old “tricks” don’t work in APIs
  - Viewstate
  - The client-side maintain the user’s state
- Known also as:
  - IDOR
  - Forceful Browsing
  - Parameter Tampering
  - Broken Authorization
A2: Broken Authentication

- As in OWASP TOP 10 2017 - A2
A3: Improper Data Filtering

- **Client-side data filtering**
  APIs tend to return more data than required. This data is usually now shown to the user, but can be easily sniffed by a web proxy.

- **Filters manipulation**
  The FE usually maintains the user’s state. The client sends more filters to the BE in order to reflect the user’s state.
A4: Lack of Resources & Rate Limiting

- Might lead to DOS, Brute force attack
A5: Missing Function/Resource Level Access Control

- As in OWASP TOP 10 2013 - A7

- Popular in APIs because:
  - Easier to predict the entry points (GET → DELETE)
    (\texttt{/api/v1/users} → \texttt{api/v1/admins})
  - Complex user policies and roles

<table>
<thead>
<tr>
<th>Sensitive Resource</th>
<th>Sensitive Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET /\texttt{api/v1/financial_reports}</td>
<td>GET /\texttt{api/v1/users/export_all}</td>
</tr>
</tbody>
</table>
A6: Mass Assignment

- Modern frameworks encourage developers to use mass assignment techniques
  
- Easier to exploit in APIs
  
  - We can usually can find a GET request that returns all the properties of an object

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

Rails:
@user = User.new(params[:user])
```
A7: Security Misconfiguration

- Improper CORS
- Unnecessary HTTP methods
- Detailed Errors
A8: Injection

- The most common inject flow (SQLi) is becoming less and less common because of ORMs
- Same as A1 - OWASP TOP TEN 2017
A9: Improper Assets Management

- CI/CD → APIs change all the time:
  - Lack of documentation

- Cloud + Deployment automation (k8s) → super easy to deploy APIs
  - Shadow APIs
  - Application servers / full environments that have been forgotten
A10: Insufficient Logging & Monitoring

- Same as A10 - OWASP TOP 10 2017
Acknowledgements

Current Draft Creation
Checkmarx – Erez Yalon, David Sopas, Paulo Silva
SALT Security – Inon Shkedy, Chris Westphel

Reviewers
42Crunch - Matthieu Estrade
Imperva - Ziv Grinberg
Shay Chen    Philippe De Ryck
Stefan Mantel Sagar Popat

<YOUR NAME HERE>
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