API Security

Erez Yalon

OWASP Singapore | 13 July 2021
Question: What do all this logos have in common?
Who am I?

Erez Yalon, Head of Security Research, Checkmarx
- Previous independent security researcher and developer
- Better at breaking than building
- Responsible for maintaining Checkmarx’s top notch vulnerability detection technology
- Lead several OWASP projects including the API Security and CN Projects
- Founder of AppSec Village in DEF CON
What is an API?

“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.”

Wikipedia

But what is API Security?

Diagram of inter-connected components (Comp 1, Comp 2, Comp 3, Comp 4).
What Uses APIs?

• Microservices
• Mobile
• IoT
• B2B
• Serverless
• Cloud
• Single Page Application

Every Modern Application
API Security
Traditional vs. Modern Applications

Traditional Application

Modern Application

Get

API Get

HTML

Raw
Traditional vs. Modern Applications

- Less abstraction layers
- Client and server (and DB) speak the same JSON language
Traditional vs. Modern Applications

The differences we see in Modern Apps

- The server is used more as a proxy for data
- The rendering component is the client, not the server
- The user’s state is usually maintained and monitored by the client
- Clients consume raw data
- More parameters are sent in each HTTP request (object ID’s, values, filters)
- APIs expose the underlying implementation of the app
What Makes APIs Vulnerable?

1. The abundancy of API endpoints makes the attack surface bigger
What Makes APIs Vulnerable?

2. Clients consume raw data
   More parameters are sent in each HTTP request (object ID’s, values, filters)
What Makes APIs Vulnerable?

3. The flexibility of CI/CD processes today, and the effortless deployment of new microservices, containers, and cloud infrastructure. It takes just a few clicks to spin up new APIs (hosts).

The rate of updates and changes in APIs may be too fast to handle.

**APIs Become hard to track:**
- Shadow APIs
- Old Exposed APIs
It’s Not All Bad News

• Traditional vulnerabilities are less common in API-based apps:
  - SQLi – due to increasing use of frameworks/ORMs
  - CSRF – due to authorization headers instead of cookies
  - Path Manipulations – due to cloud-based storage
  - Classic IT security issues - SaaS
Bridging The Gap
Bridge The Gap

**T10 - OWASP Top 10 Application Security Risks – 2017**

<table>
<thead>
<tr>
<th>Risk</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1: 2017 - Injection</td>
<td>Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker’s hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.</td>
</tr>
<tr>
<td>A2: 2017 - Broken Authentication</td>
<td>Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users’ identities temporarily or permanently.</td>
</tr>
<tr>
<td>A3: 2017 - Sensitive Data Exposure</td>
<td>Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PI. Attacks may steal or modify such weakly protected data, such as credit card numbers, social security numbers, account balances, or proprietary information.</td>
</tr>
<tr>
<td>A4: 2017 - XML External Entities (XXE)</td>
<td>Many older or poorly configured XML processors evaluate external entity references within XML documents. External entity references can be used to disclose internal files using the file URI handler, internal file names, and XML entity references.</td>
</tr>
<tr>
<td>A5: 2017 - Broken Access Control</td>
<td>Many older or poorly configured XML processors evaluate external entity references within XML documents. External entity references can be used to disclose internal files using the file URI handler, internal file names, and XML entity references.</td>
</tr>
<tr>
<td>A6: 2017 - Security Misconfiguration</td>
<td>Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched and upgraded in a timely fashion.</td>
</tr>
<tr>
<td>A7: 2017 - Cross-Site Scripting (XSS)</td>
<td>XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim’s browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.</td>
</tr>
<tr>
<td>A8: 2017 - Insecure Deserialization</td>
<td>Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.</td>
</tr>
<tr>
<td>A9: 2017 - Using Components with Known Vulnerabilities</td>
<td>Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.</td>
</tr>
<tr>
<td>A10: 2017 - Insufficient Logging &amp; Monitoring</td>
<td>Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 203 days, typically detected by external parties rather than internal processes or monitoring.</td>
</tr>
</tbody>
</table>
OWASP API Security Top 10
OWASP API Security Top 10

• API1: Broken Object Level Authorization
• API2: Broken Authentication
• API3: Excessive Data Exposure
• API4: Lack of Resources & Rate Limiting
• API5: Broken Function Level Authorization
• API6: Mass Assignment
• API7: Security Misconfiguration
• API8: Injection
• API9: Improper Assets Management
• API10: Insufficient Logging & Monitoring
API8 – Injection

Why drop from A1 to A8?

- “Injection” is #1 because of SQL Injections.
- SQL Injection are not very common in modern APIs, because:
  - Use of ORMs
  - Increasing use of NoSQL

- NoSQL injections are a thing, but are usually not as common / severe

[Image of OWASP Top 10 - 2017 table where A1:2017-Injection is highlighted]
API10 - Insufficient Logging & Monitoring

Same as OWASP Top 10
Exploitation of insufficient logging and monitoring is the bedrock of nearly every major incident.

Attackers rely on the lack of monitoring and timely response to achieve their goals without being detected.
OWASP API Security Top 10

- API1: Broken Object Level Authorization
- API2: Broken Authentication
- API3: Excessive Data Exposure
- API4: Lack of Resources & Rate Limiting
- API5: Broken Function Level Authorization
- API6: Mass Assignment
- API7: Security Misconfiguration
- API8: Injection
- API9: Improper Assets Management
- API10: Insufficient Logging & Monitoring
Access Control
API Security’s Biggest Challenge
Access Control

• API1: Broken Object Level Authorization
• API2: Broken Authentication
• API5: Broken Function Level Authorization
API2: Broken Authentication

Lack of protection

- Login
- Mobile Login
- Forgot Password
- Update Location
- Edit Photo

Protection

Extra Protection

Misimplementation

- JWT Supports {“alg”:”none”}
- No validation of authentication provider
- Passwords stored without salt
- Etc...

- Captcha
- Account lockout mechanism
- Credentials Stuffing Protection
API2: Broken Authentication

Why is it so common in APIs?

• Authentication endpoints are exposed to anyone by design.

• Software/security engineers have misconceptions.
  
    API keys should not be used for user’s authentication

    Authorization != Authentication

• Multiple authentication flows in modern apps
  
    IoT, Mobile, Legacy, Deep links with credentials

    etc...
Access Control

• API1: Broken Object Level Authorization
• API2: Broken Authentication
• API5: Broken Function Level Authorization
API1: Broken Object Level Authorization (BOLA)
API1: Broken Object Level Authorization (BOLA)

Why is it so common in APIs?

• The attack surface is much wider

• No security solution exists that solves the problem
API1: Broken Object Level Authorization (BOLA)

Why not “IDOR”? It’s not accurate / indicative enough

- "IDOR" - Insecure Direct Object Reference
- "IDOR" implies that object reference should be indirect (salted hash map / random string added to every ID)
- The problem is not the Object Reference, but a lack of authorization

What would happen if you asked your developers to implement “Indirect” mechanism in every place that receives ID?
Access Control

- API1: Broken Object Level Authorization
- API2: Broken Authentication
- API5: Broken Function Level Authorization
API5: Broken Function Level Authorization (BFLA)
**API5: Broken Function Level Authorization (BFLA)**

Why is it so common in APIs?

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

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

<table>
<thead>
<tr>
<th></th>
<th>Get user’s profile (Regular endpoint)</th>
<th>Delete user (Admin endpoint)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Traditional</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>Modern</strong></td>
<td>GET /api/v2/users/1337</td>
<td>DELETE /api/v2/users/1337</td>
</tr>
</tbody>
</table>
Coursera API vulnerabilities disclosed by researchers

Coursera took “prompt ownership” of the bugs, once reported.

Researchers have disclosed a set of API vulnerabilities in the Coursera platform.

On Thursday, Checkmarx security researcher Paulo Silva revealed the discovery of multiple security failings in the Coursera online learning platform.
Access Control Vulnerability in Real Life

Smart vacuum flaws could give hackers access to camera feed, say security researchers

Researchers at Checkmarx detail security issues discovered with a robot vacuum cleaner.
OWASP API Security Top 10

- API1: Broken Object Level Authorization
- API2: Broken Authentication
- API3: Excessive Data Exposure
- API4: Lack of Resources & Rate Limiting
- API5: Broken Function Level Authorization
- API6: Mass Assignment
- API7: Security Misconfiguration
- API8: Injection
- API9: Improper Assets Management
- API10: Insufficient Logging & Monitoring
API3 – Excessive Data Exposure

Super Safe App
Bob’s Profile

- Name: Bob
- Role: Minion
- Hobby: Bananas

Filtering sensitive data on the client side is always a bad idea.
API3 – Excessive Data Exposure

Why it is so common?

- REST Standards 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
API3 - 3Fun Hack

Found by Alex Lomas, Pen Test Partners
Found by Alex Lomas, Pen Test Partners
API3 - 3Fun Hack

Found by Alex Lomas, Pen Test Partners
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":"Bob", "pass":"123456"}

POST /api/users/new
{"username":"Bob", "pass":"123456", "role":"admin"}

Might contain sensitive params that the user should not have access to
# OWASP API Security Top 10

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API1</td>
<td>Broken Object Level Authorization</td>
</tr>
<tr>
<td>API2</td>
<td>Broken Authentication</td>
</tr>
<tr>
<td>API3</td>
<td>Excessive Data Exposure</td>
</tr>
<tr>
<td>API4</td>
<td>Lack of Resources &amp; Rate Limiting</td>
</tr>
<tr>
<td>API5</td>
<td>Broken Function Level Authorization</td>
</tr>
<tr>
<td>API6</td>
<td>Mass Assignment</td>
</tr>
<tr>
<td>API7</td>
<td>Security Misconfiguration</td>
</tr>
<tr>
<td>API8</td>
<td>Injection</td>
</tr>
<tr>
<td>API9</td>
<td>Improper Assets Management</td>
</tr>
<tr>
<td>API10</td>
<td>Insufficient Logging &amp; Monitoring</td>
</tr>
</tbody>
</table>

- **API1: Broken Object Level Authorization**
- **API2: Broken Authentication**
- **API3: Excessive Data Exposure**
- **API4: Lack of Resources & Rate Limiting**
- **API5: Broken Function Level Authorization**
- **API6: Mass Assignment**
- **API7: Security Misconfiguration**
- **API8: Injection**
- **API9: Improper Assets Management**
- **API10: Insufficient Logging & Monitoring**
API4 - Lack of Resources & Rate Limiting

When a resource (memory, CPU, DB, file, etc.) is exposed to the web, there should be defined use limits

- Requests
  - Number, Frequency

- Files
  - Size

- Strings
  - Length
API4 - Lack of Resources & Rate Limiting

Several consequences for not having a limit:

- DoS – Denial of Service
- Brute-force attacks
  - Credential Stuffing
API7 – Security Misconfiguration

- Weak encryption
- Unnecessary exposed HTTP methods
- CSRF protection turned off
- Detailed errors
- Improper CORS
API9 – Improper Assets Management

Two similar housekeeping Issues

Lack of documentation  |  Exposed Risky APIs
API9 – Improper Assets Management

Two similar housekeeping issues

- Lack of documentation
  - /v2/download_transactions_as_pdf
  - /v2/transfer_money
  - /v0/legacy_b2b/export_all_users

- Exposed Risky APIs
  - beta-api.xxx.com
  - payments-api.xxx.com
  - mobile-api.xxx.com

API Gateway

Client

Checkmarx
API9 – Improper Assets Management

Why is it such a big issue?

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

- With cloud & deployment automation it is way too easy to spin up new APIs and machines:
  - API hosts that have been forgotten
  - Complete environments that have been forgotten (what the heck is qa-3-old.app.com ?)
OWASP API Security Top 10

- API1: Broken Object Level Authorization
- API2: Broken Authentication
- API3: Excessive Data Exposure
- API4: Lack of Resources & Rate Limiting
- API5: Broken Function Level Authorization
- API6: Mass Assignment
- API7: Security Misconfiguration
- API8: Injection
- API9: Improper Assets Management
- API10: Insufficient Logging & Monitoring
Summary
What You Need to Remember

• Modern API-based apps are different
• Being different, they have their own security issues
• The attack surface is much wider
• There is more data moving between components
• Access Control is a real challenge
Thank you