

API Security Project Top-10 Release Candidate

OWASP Projects' Showcase

Sep 12, 2019

Founders and Sponsors



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

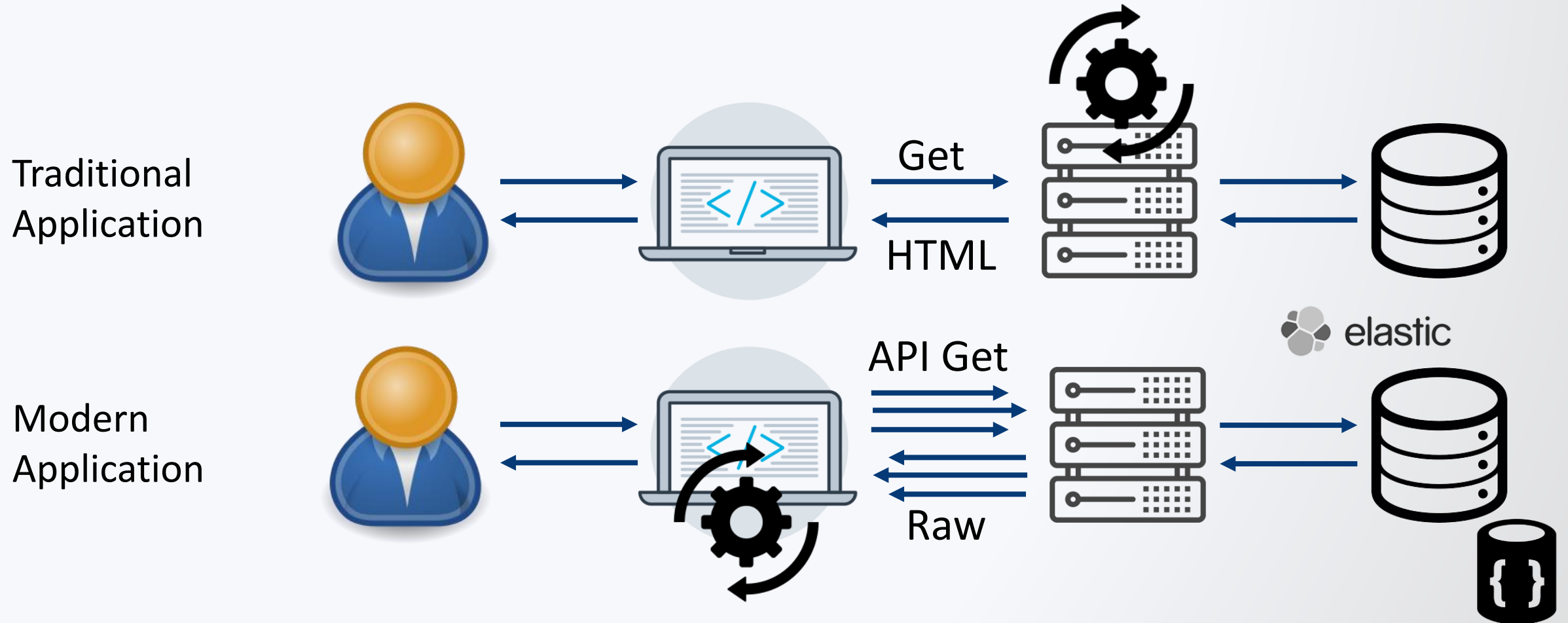
How API Based Apps are Different?

Client devices are becoming varied and stronger



Logic moves from Backend to Frontend
(together with some vulnerabilities)

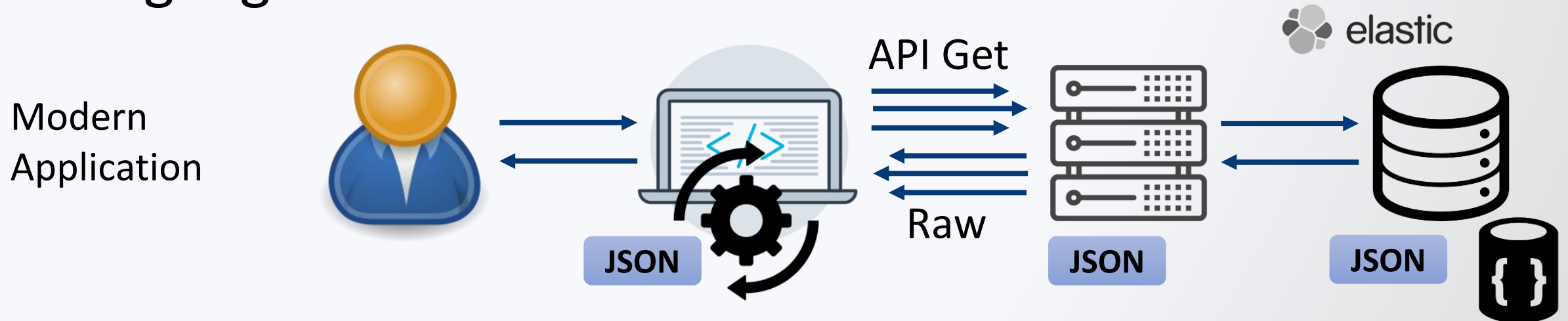
Traditional vs. Modern



Traditional vs. Modern

Less abstraction layers

Client and server (and DB) speak the same JSON language

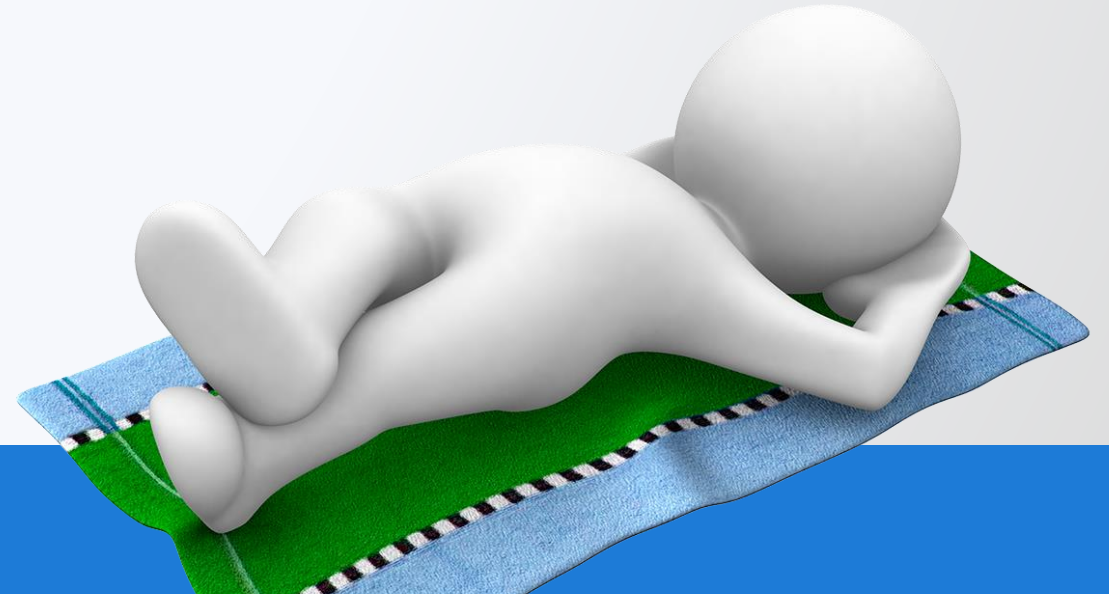


How API Based Apps are Different?

- 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?

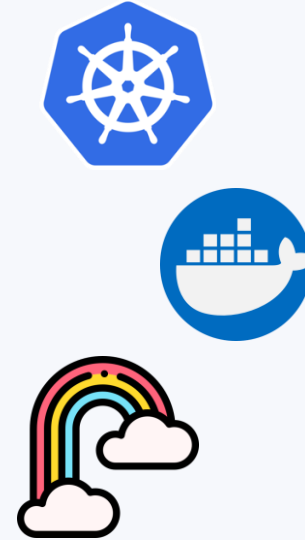
The good news

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 Secrity Top 10
- API Security Cheat Sheet
- crAPI (**C**ompletely **R**idiculous **API**
- an intentionally vulnerable API project)

Roadmap

| | Top 10 | Cheat Sheet | crAPI |
|---------|----------|-------------|-------------|
| 2019 Q1 | Prepare | | |
| 2019 Q2 | Kick-Off | | |
| 2019 Q3 | V1.0 | Kick-Off | Prepare |
| 2019 Q4 | | Collaborate | Kick-Off |
| 2020 Q1 | | V1.0 | Collaborate |
| 2020 Q2 | | | V1.0 |

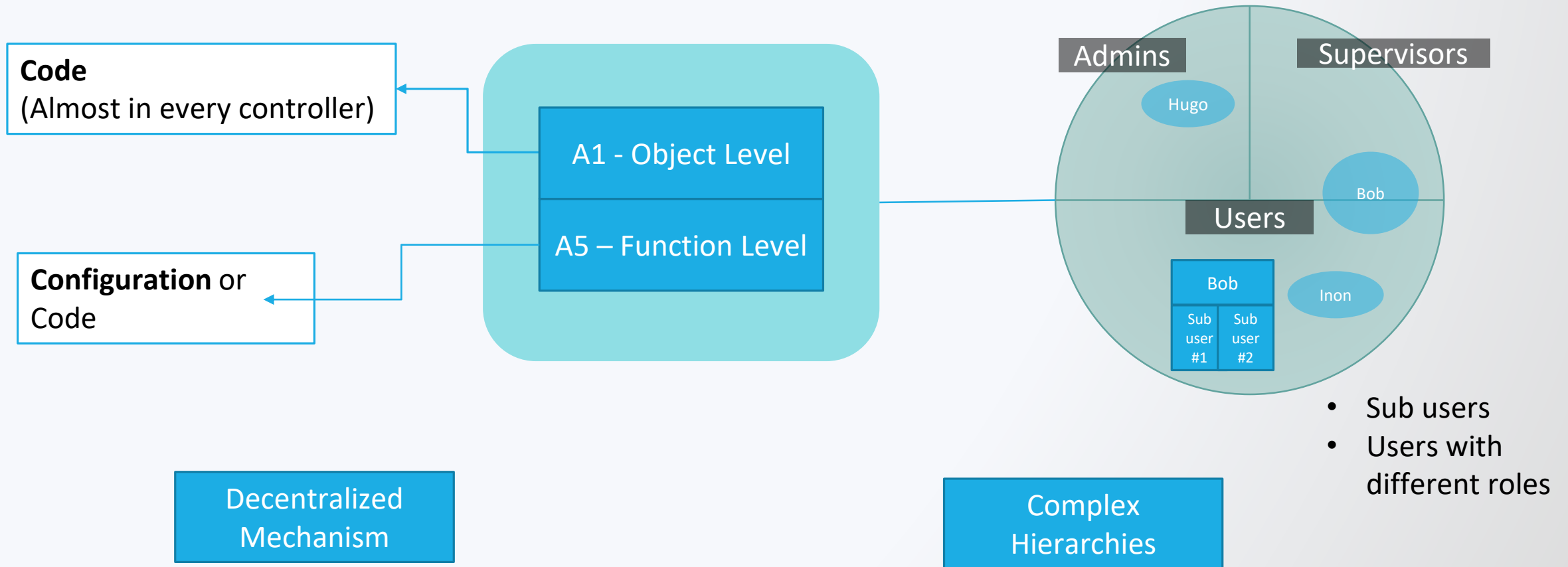
The creation process of the Top10

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

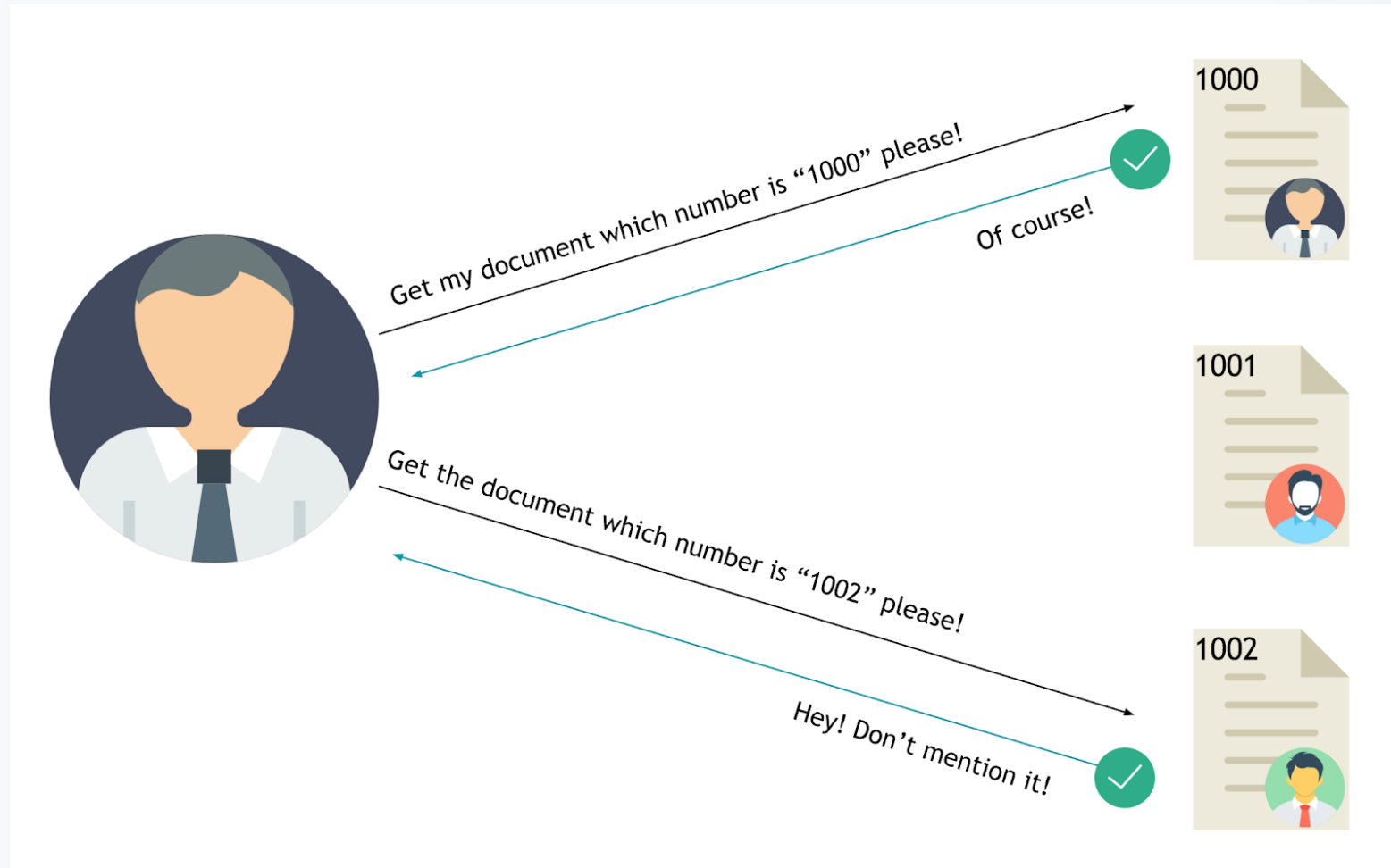
API Security Top 10

- **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



A1 – BOLA (Broken Object Level Authorization)



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



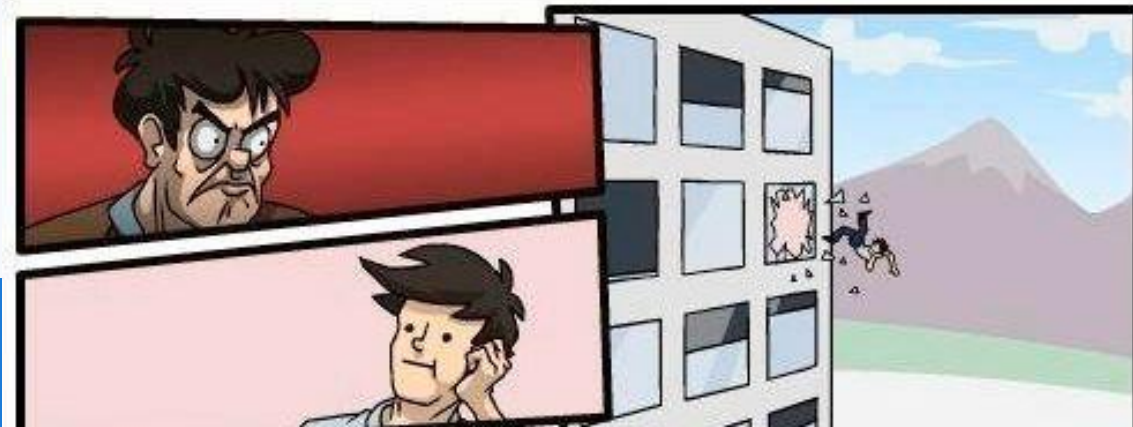
A1 – BOLA

Why not "IDOR"?

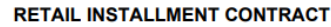
- "IDOR" - Insecure **D**irect **O**bject **R**eference is a **cool** name
- It's **not accurate** / indicative enough
- The name "**I****D****OR**" hints that the object reference (ID) should be indirect (e.g.: a salted hash map)
 - What would happen if you asked your developers to implement “Indirect” 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 -



How I could access the personal information of 2 million Verizon Wireless customers due to 1 very simple mistake



SELLER (CREDITOR): Verizon Wireless Services, LLC ("Verizon Wireless")
One Verizon Way, Basking Ridge, NJ 07920 (908) 559-7000

YOUR COMPANY, meaning the Buyer/Company named above, agree to pay US, the Seller/Creditor named above as Verizon Wireless, the Total Sale Price of the goods identified above according to the Terms of this Retail Installment Sale Agreement/ Retail Installment Obligation (referred to below as "Agreement").

Your Company's payment schedule will be:

Number of Payments:24; Payment 1:\$25.22; Payments 2-24:\$24.99

PAYMENTS RECEIVED 15 OR MORE DAYS AFTER YOUR COMPANY'S DUE DATE MAY INCUR A LATE PAYMENT FEE OF UP TO 5% OR \$5, WHICHEVER IS LESS. PLEASE SEE YOUR COMPANY'S AGREEMENT TERMS FOR ANY ADDITIONAL INFORMATION ABOUT NONPAYMENT, DEFAULT, ANY REQUIRED PAYMENT IN FULL BEFORE THE SCHEDULED PAYMENT DATES, AND PREPAYMENT TERMS.

| | |
|----------------------------------|----------|
| ITEMIZATION OF AMOUNT FINANCED | \$599.99 |
| (A) CASH PRICE (excluding tax) | \$599.99 |
| (B) DOWN PAYMENT (if applicable) | \$0.00 |
| (C) FINANCE CHARGE | \$0.00 |

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

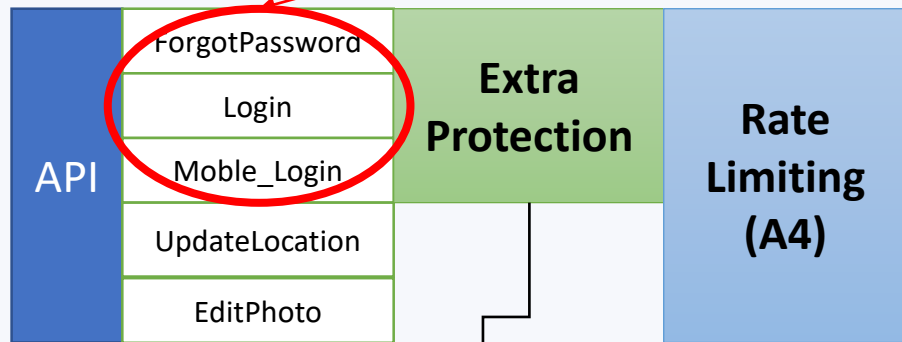
- **A2 – Broken Authentication**
Why is it so common?

- 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



- Account lockout mechanism
- Captcha
- Credentials Stuffing Protection

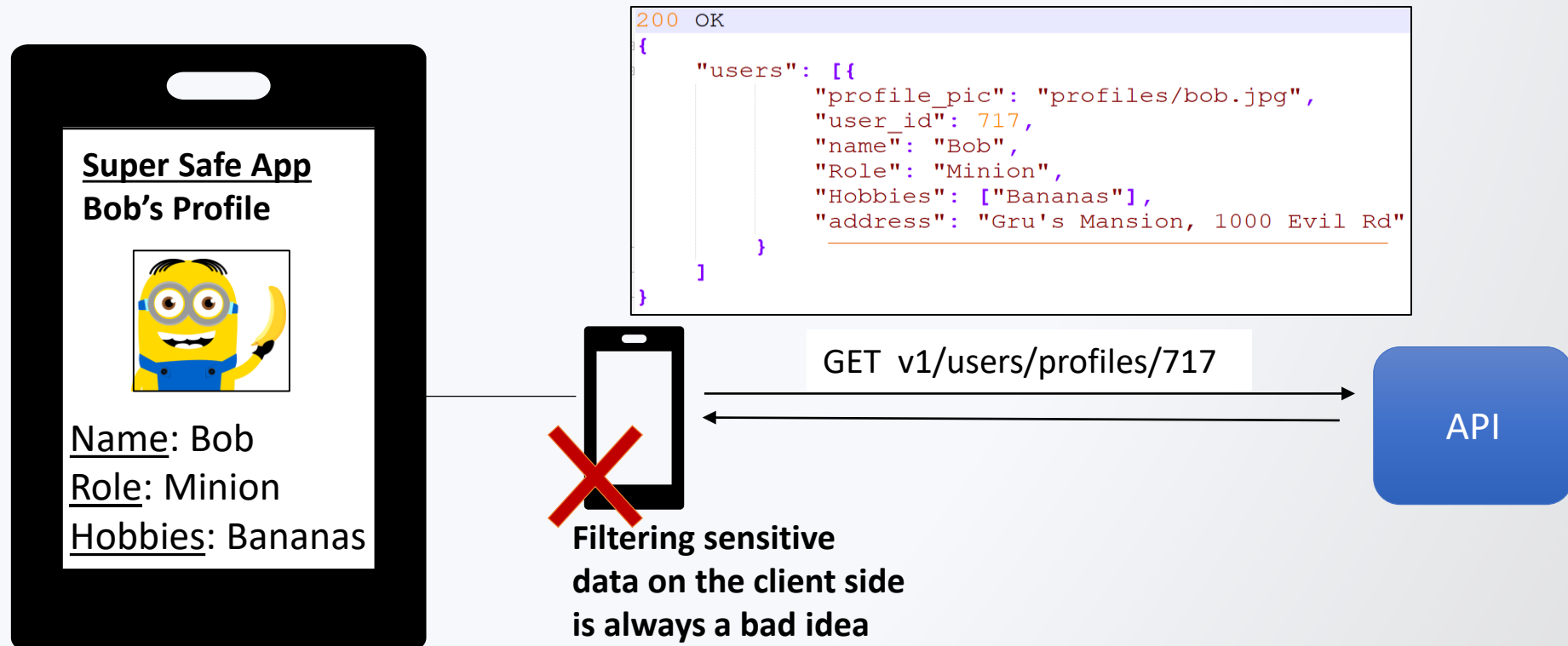
Misimplementation

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

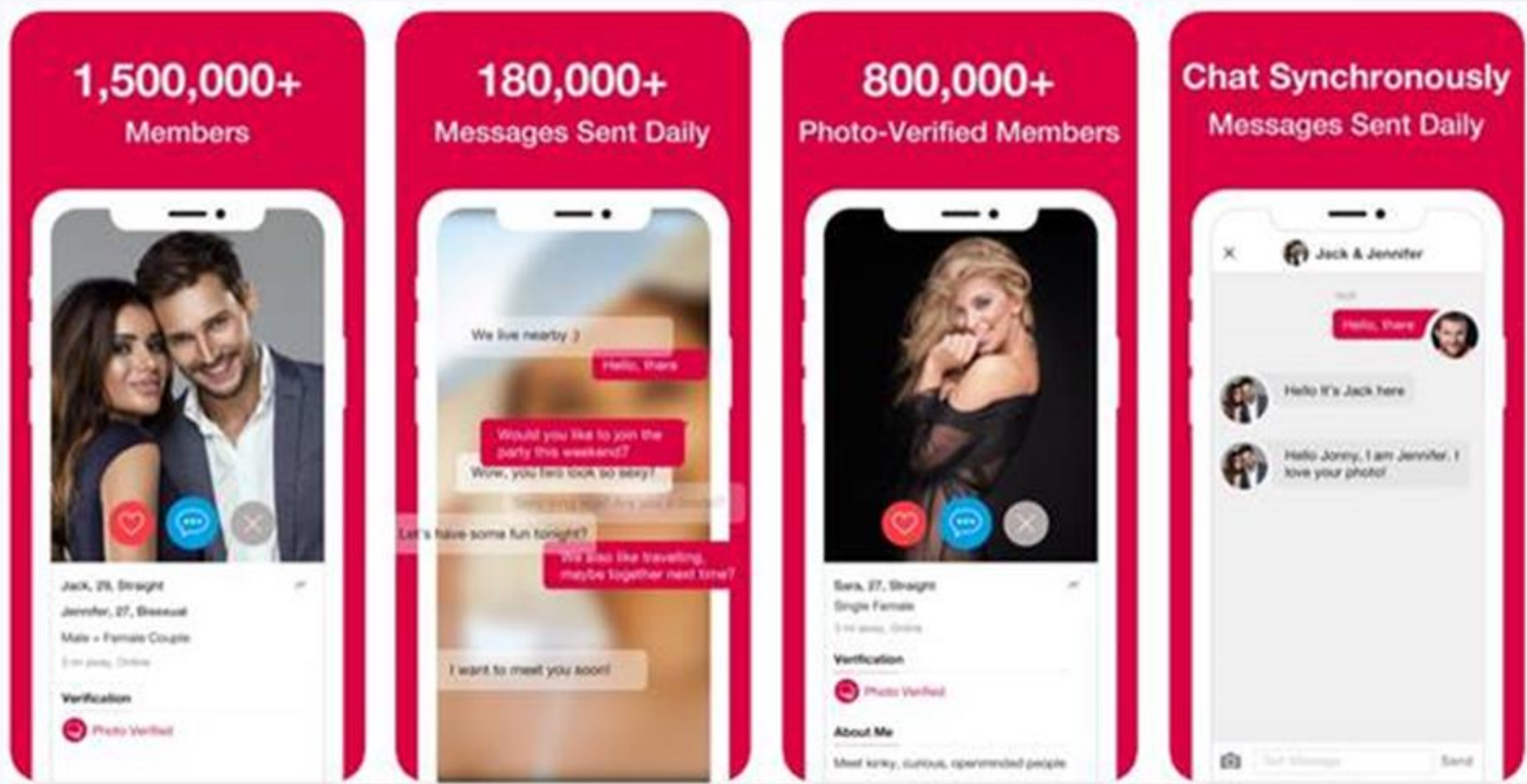
A3 – Excessive Data Exposure

- 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



A3 - Example from "3fun" app



- Found by Alex Lomas, [Pen Test Partners](#)

| # | Host | Method | URL | Params | Edited | Status | Length | MIME type |
|-----|-----------------------|--------|---|--------|--------|--------|--------|-----------|
| 322 | https://www.go3fun.co | POST | /account_kit_reg | ✓ | | 200 | 447 | JSON |
| 325 | https://www.go3fun.co | POST | /user/device_token | ✓ | | 200 | 198 | JSON |
| 326 | https://www.go3fun.co | POST | /user/update | ✓ | | 200 | 265 | JSON |
| 327 | https://www.go3fun.co | POST | /reset_push_badge | | | 200 | 198 | JSON |
| 329 | https://www.go3fun.co | GET | /match_users?from=0&latitude=51. [REDACTED] | ✓ | | 200 | 23807 | JSON |
| 331 | https://www.go3fun.co | GET | /user/refresh | | | 200 | 788 | JSON |

Request Response

Raw Headers Hex JSON Beautifier

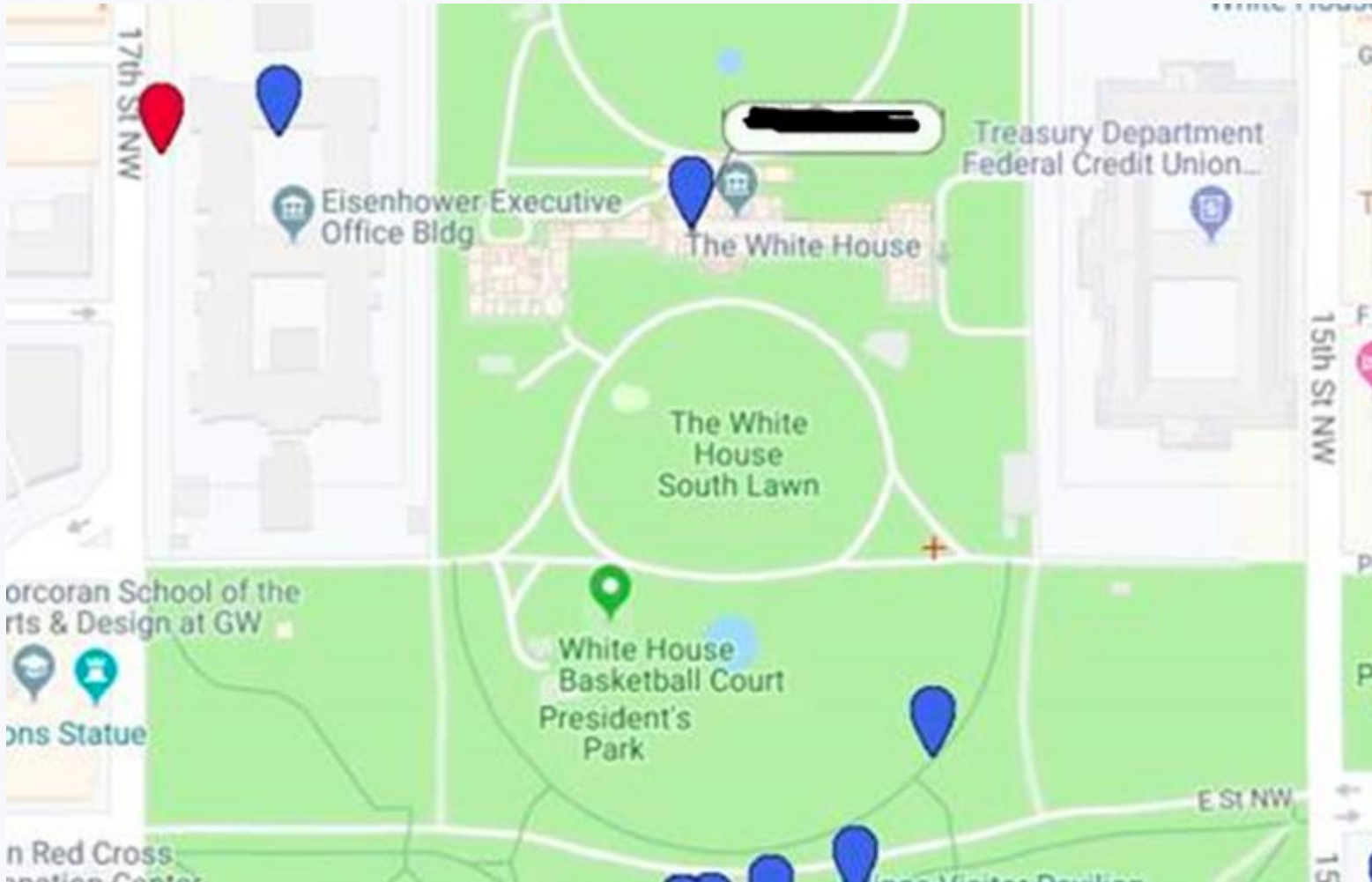
```

},
"latitude": "51. [REDACTED]",
"membership": "2",
"birthday": "1977-[REDACTED]",
"sex_orient": "4",
"gender": "1",
"longitude": "-0.1 [REDACTED]",
"photo_verified_status": "1",
"active": "0",
"partner_sex_orient": "0",
"liked_me": "0",
"settings": {
  "show_online_status": "1",
  "show_distance": "1"
},
"username": "[REDACTED]",
"usr_id": "17 [REDACTED]",
"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/[REDACTED]/[REDACTED]-small.jpg",
      "photo_id": "38 [REDACTED]",
      "py": "500",
      "px": "750",
      "photo": "https://s3.amazonaws.com/3fun/821/[REDACTED]/[REDACTED]-big.jpg",
      "descr": null
    }
  ]
}

```

- Found by Alex Lomas, [Pen Test Partners](#)

A3 - 3Fun Hack

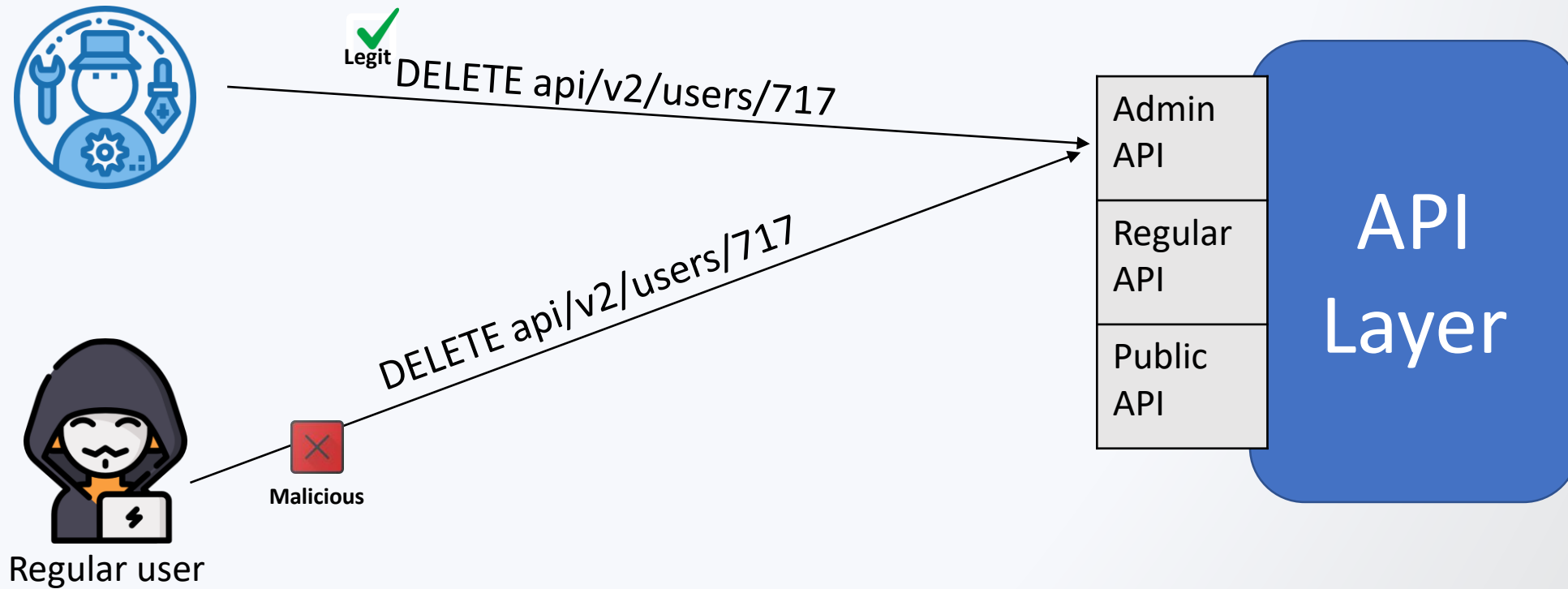
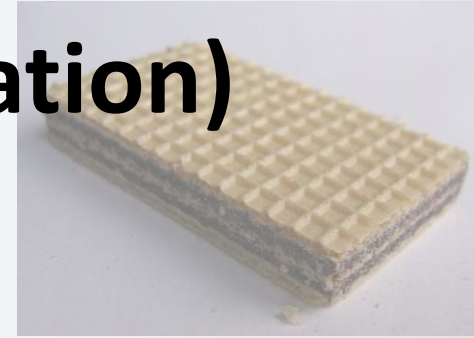


- Found by Alex Lomas, [Pen Test Partners](#)

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)



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

| Action | Get user's profile (Regular endpoint) | Delete user (Admin endpoint) |
|------------------|---------------------------------------|--|
| Traditional Apps | GET /app/users_view.aspx?user_id=1337 | POST app/admin_panel/users_mgmt.aspx action=delete&user_id=1337 |
| APIs | GET /api/v2/users/1337 | DELETE /api/v2/users/1337 |

Hard to Predict! ☹️

Very Predictable! 😊

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])
```



Might contain sensitive params that the user should not have access to

POST /api/users/new

{"username":"Inon", "pass":"123456"}

Legit Request

POST /api/users/new

{"username":"Inon", "pass":"123456", "role":"admin"}


Malicious Request

A6 – Mass Assignment

- Easier to exploit in APIs
- Instead of guessing object's properties, just find a GET method that returns them

```
GET /v1/user/video_files
-----
200 OK
{
  "id": 371,
  "name": "clip.mp4",
  "conversion_params": "-v codec h264"
}
```

```
PUT /v1/videos/371
{
  "name": "clip.mp4",
  "conversion_params": "-v codec h264 && format C:/"
}
```



A7 – Security Misconfiguration

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



A8 – Injection

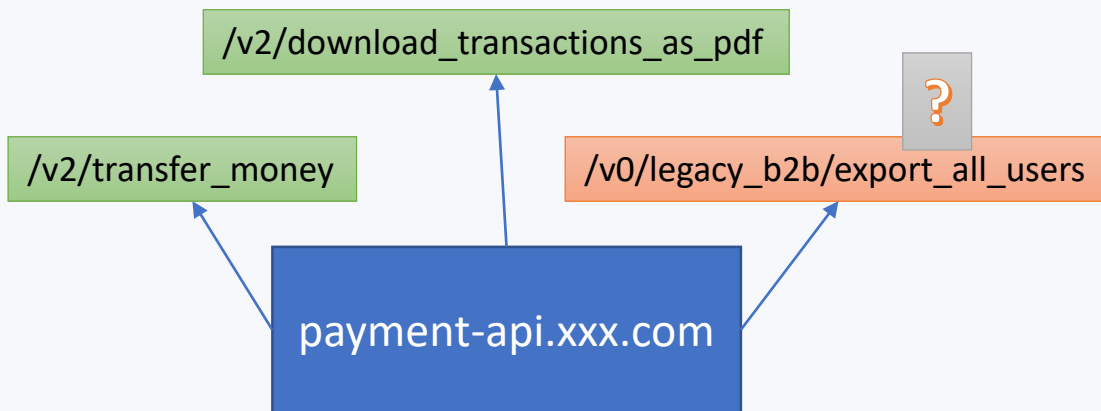
Why from **A1** to **A8**?

- 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

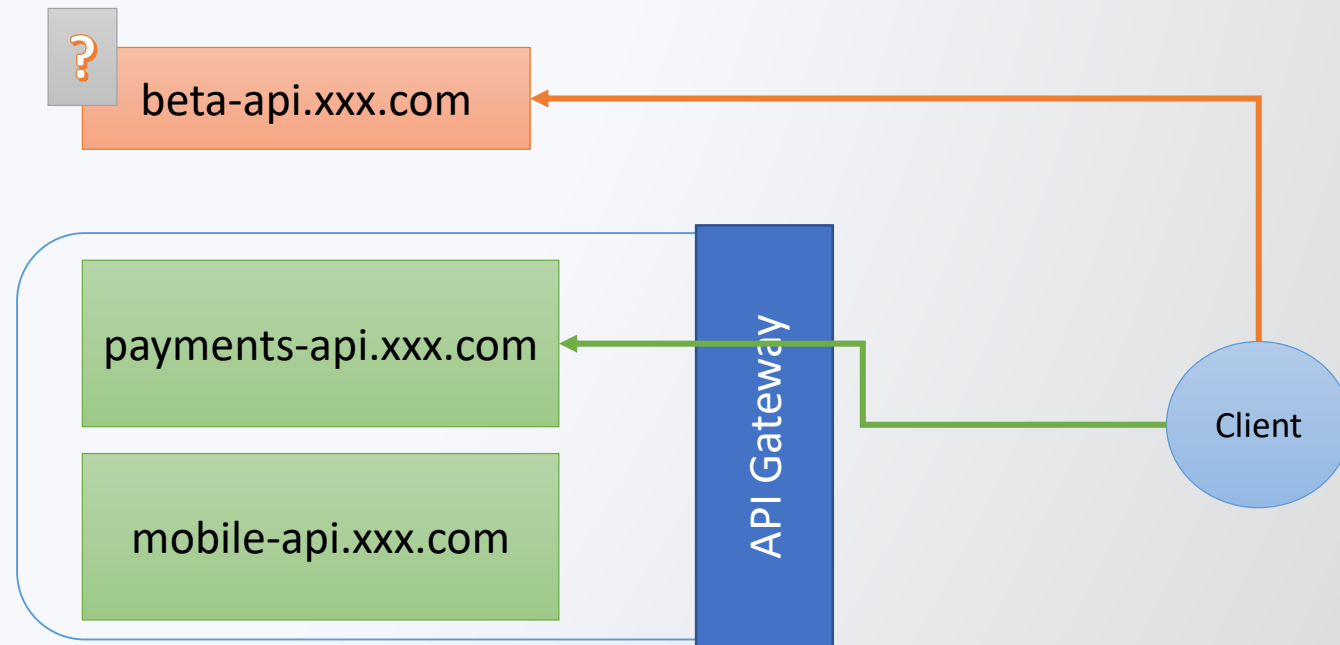
A9 – Improper Assets Management

Actually two different things

Lack of documentation



Exposed Risky Undocumented APIs



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://github.com/OWASP/API-Security/blob/develop/CONTRIBUTING.md)



[https://www.owasp.org/index.php/OWASP API Security Project](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!