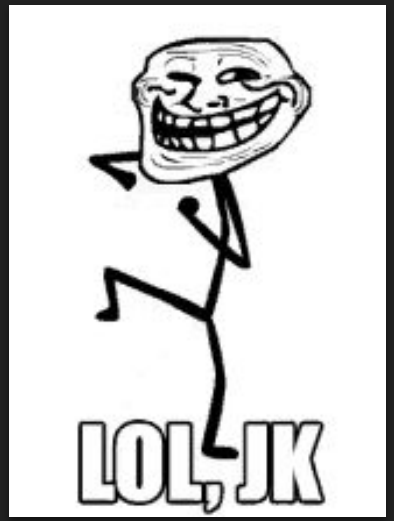
Agenda

- Modern AppSec:
  - SQLi
  - XSS
  - CSRF
  - More boring stuff

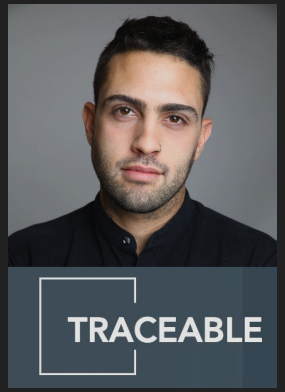


Real Agenda

- Modern AppSec Issues
- Relevant (and cool) Examples
- What has changed

Whoami

- Head of Security Research @ Traceable.ai
- 8 Years of experience in AppSec
- I've grown up with APIs



Government, Military, Financial

Java | Microsoft ASP.net | SAP

Multi Page Apps, On Prem, Waterfall,
Less APIs

Startups, Tier 1 Companies

RAILS | node.js

Single Page Apps, Cloud, CI/CD,
Mostly APIs

Traditional

Web Images Video Local Shopping More

YAHOO!

My Yahoo! Sign In

MY FAVORITES Add

- View Yahoo! Sites
- Yahoo! Mail
- Autos
- Facebook
- Finance (Dow Jones)
- Games
- HotJobs
- Messenger
- Movies

TODAY - March 26, 2010

How to tell if you're middle-class

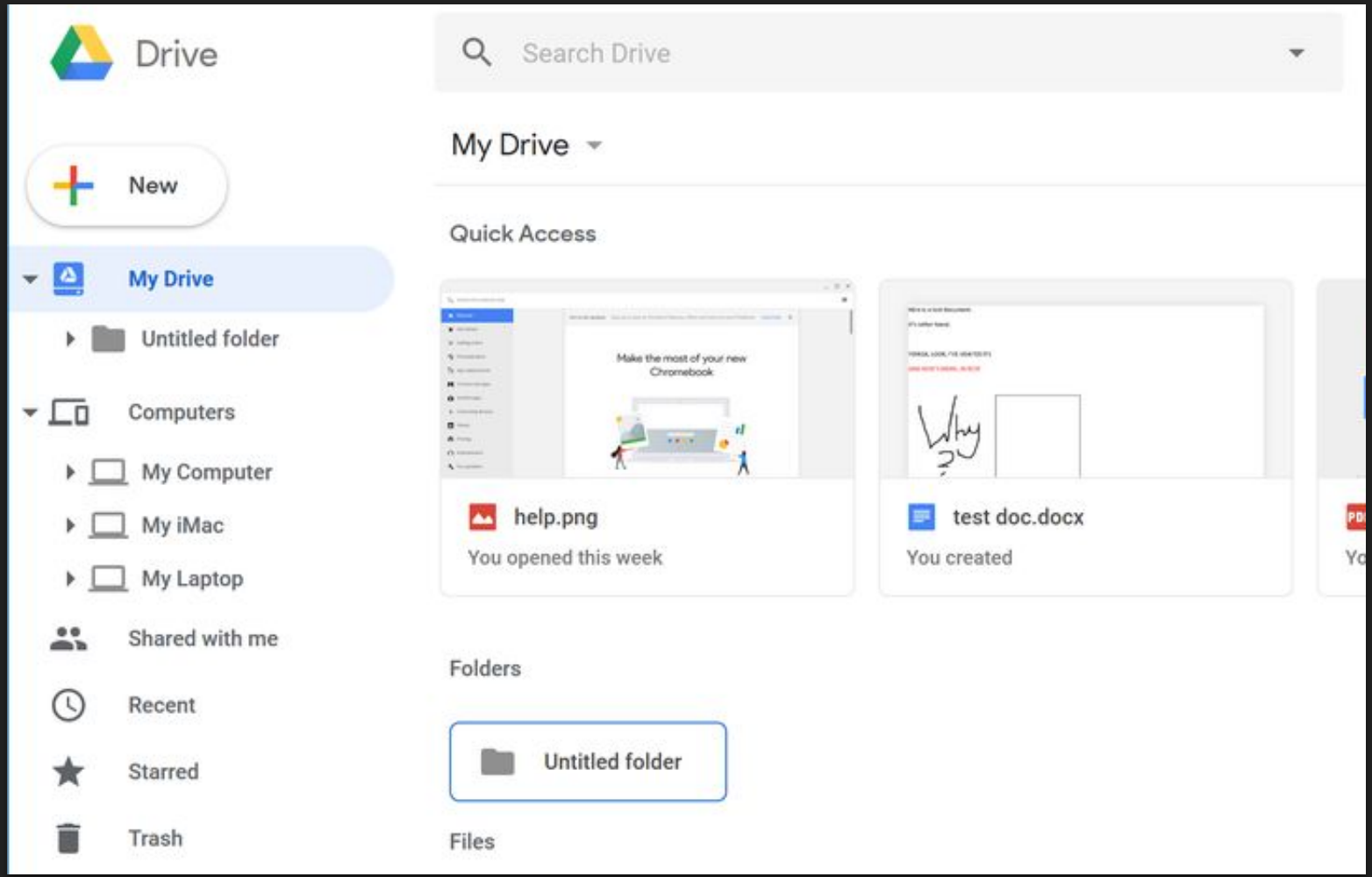
Find out what typical median-income families earn, and how much they spend on homes and cars. » Are you close?

- 7 middle-class worries
- How to avoid marriage tax
- Find top-paying jobs

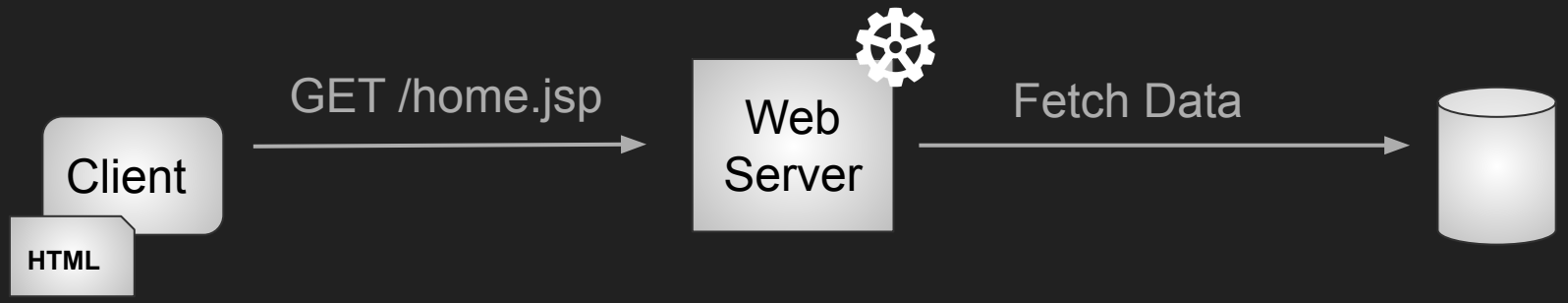
TRI

1.
2.
3.
4.
5.

Modern

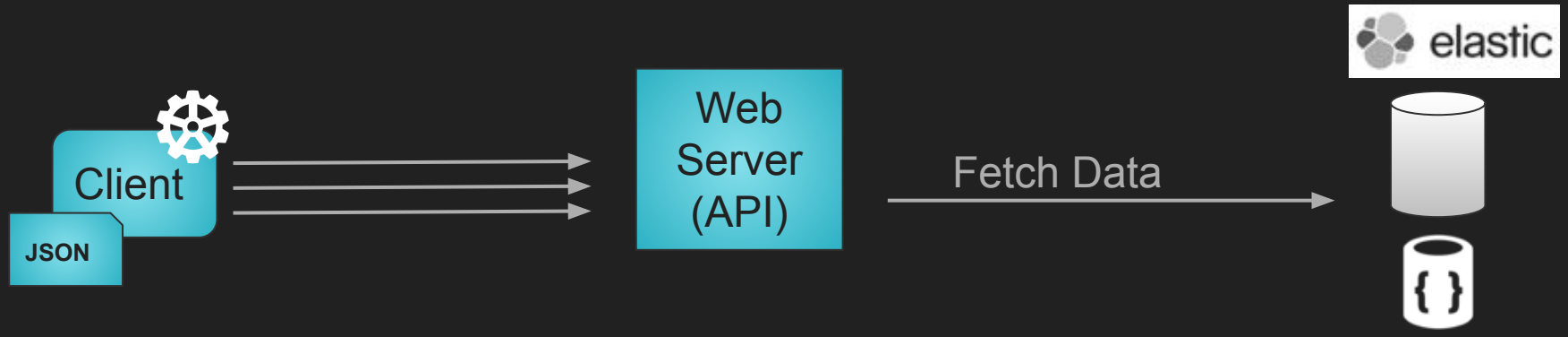

Drive
Search Drive
My Drive
New
Quick Access
My Drive
Untitled folder
Computers
My Computer
My iMac
My Laptop
Shared with me
Recent
Starred
Trash
Make the most of your new Chromebook
help.png
You opened this week
test doc.docx
You created
Folders
Untitled folder
Files

What's changed?



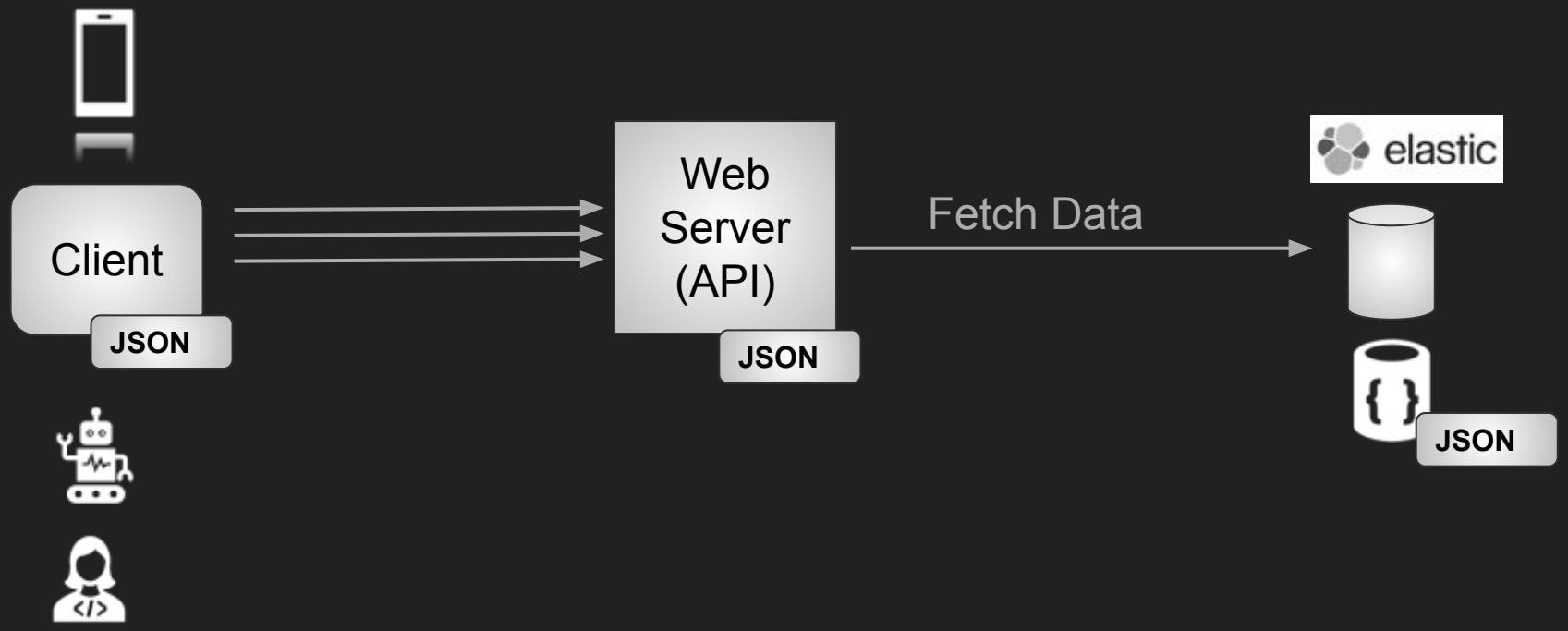
Traditional

Modern

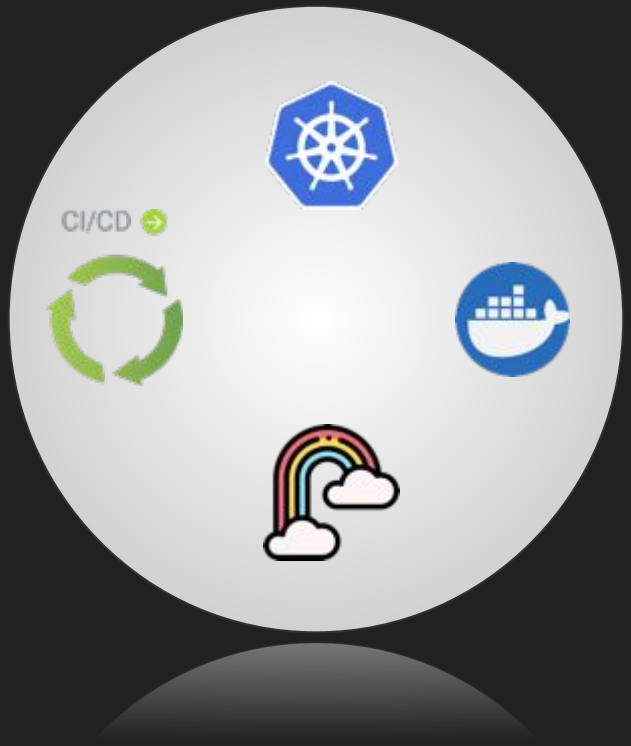


What's new?

- **Clients**:
  More types, more powerful
- **Less Abstraction Layers:**
  Shared language - JSON



Dev(Sec)Ops



Bad :

Hard too keep on track
(Shadow APIs)

Good :

Classic IT issues
(open ports, old version)
barley exist

The Good News

Issue	Solved By
SQLi	ORMs
CSRF	Use of Authorization Header
XSS	Clients are responsible
Path Manipulation	Cloud Based Storage
XXE	JSON

The Bad News

Less Abstraction

- Predictable endpoints
- More exposed PII

Client does rendering

- Wider Attack Surface
  - More entry points
  - More params

OWASP API Project

- Defining the top 10 threats
- Leaders

Erez Yalon
(Checkmarx)

Inon Shkedy
(Traceable.ai)

- Join us!

https://www.owasp.org/index.php/OWASP_API_Security_Project

OWASP TOP 10 For APIs

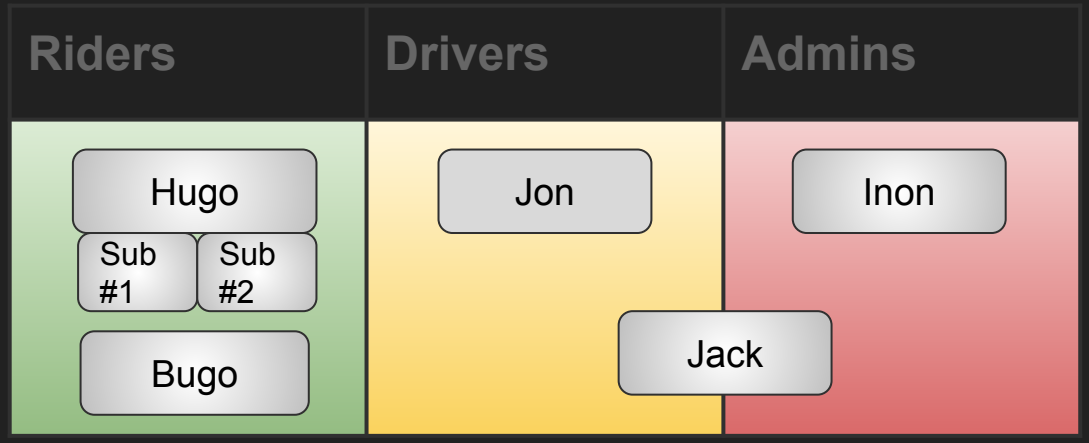
#1	BOLA (Broken Object Level Authorization)
#2	Broken Authentication
#3	Excessive Data Exposure
#4	Lack of Resources & Rate Limiting
#5	BFLA (Broken Function Level Authorization)
#6	Mass Assignment
#7	Security Misconfiguration
#8	Injection
#9	Improper Assets Management
#10	Insufficient Logging & Monitoring

Authz in APIs - The **Challenge**

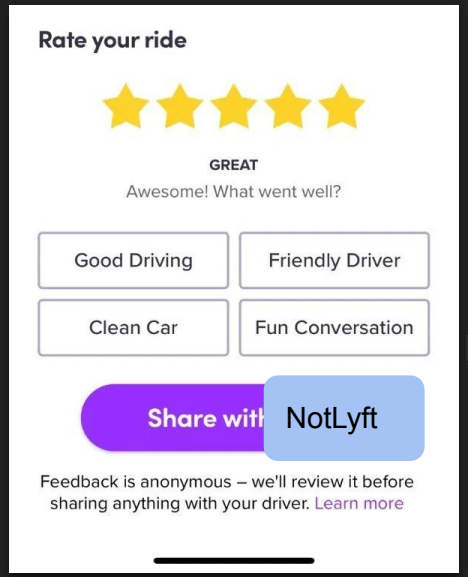
- Decentralized Mechanism

Object Level	Function Level
Code (Almost every controller)	Code, Configuration, API-gateway

- Complex Users & Roles Hierarchies



A1 - BOLA (Broken Object-Level Authorization)



Trip.find_by_id(**718492**).update

POST api/trips/rate_trip
{“**trip_id**”:718492, “rate”:5}

API

UPDATE trips …
WHERE ID = 718492

DB

BOLA - 2 types

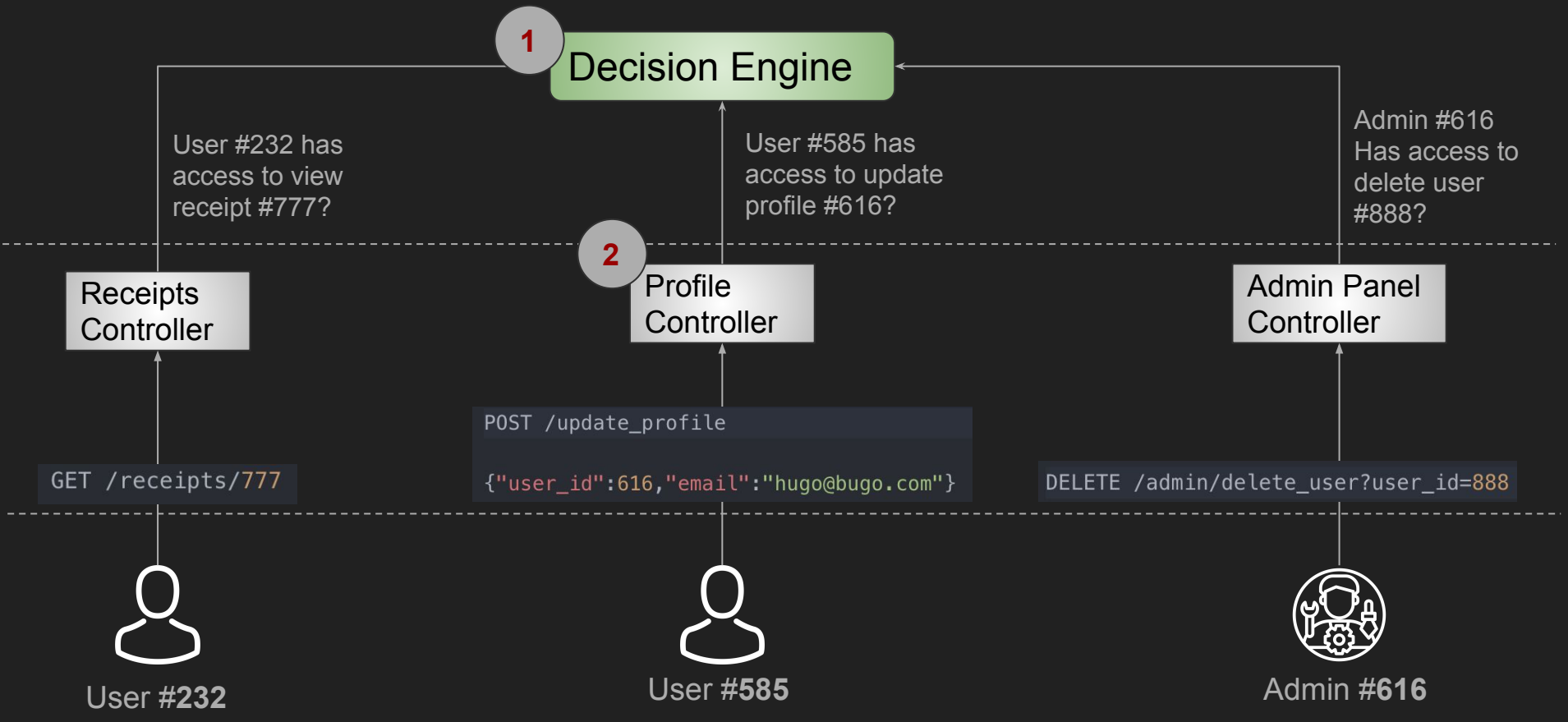
Based on user_id

- Easy to protect
  - If (params[:user_id] == current_user.id)

Based on object_id

- Challenging
  - A trip with a co-rider

What is a good authorization mechanism?



BOLA - Why So Common?

- More IDs are sent from the clients to the APIs

- REST standard encourages developers to send IDs in URLs
  - /users/**717**/details

- Even though there’s an authz mechanism, developers just don’t use it

BOLA - Why Not IDOR

- **IDOR** - **I**nsecure **D**irect **O**bject **R**eference
- C00L name, not accurate
- The problem is not about the IDs !

BOLA - Solutions that <u>don’t</u> solve the problem

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

BOLA - Solutions that <u>solve</u> the problem

- Good authorization mechanism
- Make sure that developers actually use it in every controller

BOLA - Uber - Full Account Takeover

Request

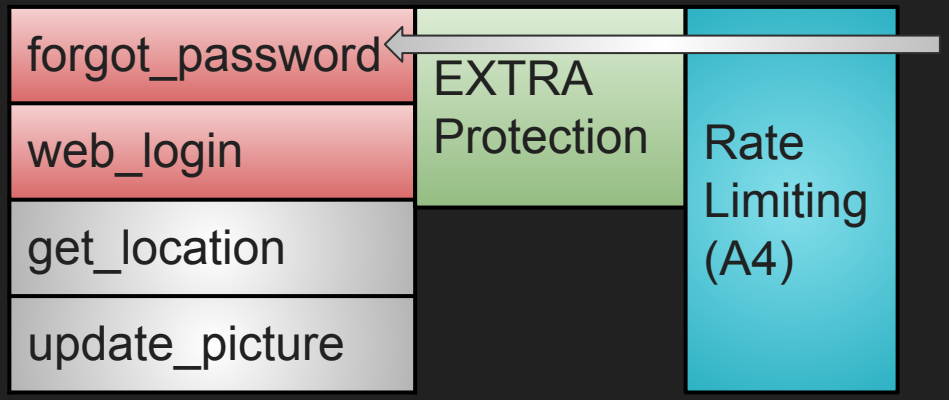
```
POST /marketplace/\_rpc?rpc=getConsentScreenDetails HTTP/1.1
Host: bonjour.uber.com
Connection: close
Content-Length: 67
Accept: application/json
Origin: [https://bonjour.uber.com](https://bonjour.uber.com)
x-csrf-token: xxxx
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleW
DNT: 1
Content-Type: application/json
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: xxxxx
{"language":"en","userUuid":"xxxx-776-4xxxx1bd-861a-837xxx604ce"}
```

Response


```
{
    "status":"success",
    "data":{
        "data":{
            "language":"en",
            "userUuid":"xxxxxx1e"
        },
        "getUser":{
            "uuid":"cxxxxxc5f7371e",
            "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":"xxxxxxxx",
```


Found by Anand Prakash, AppSecure

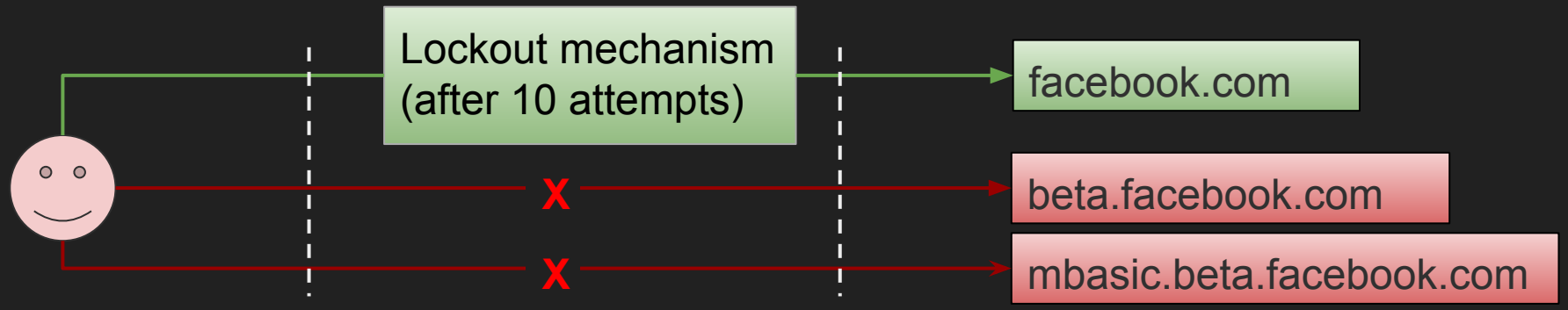
A2 - Broken User Authentication



Extra protection:
- Account lockout
- Captcha
- Brute Force attacks

Misconfiguration:
- JWT allows {“alg”:”none”}
- Tokens don’t expire
- etc..

A2 - Facebook - Full Account Takeover



Vulnerable request:

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

lsd=AVoywo13&n=XXXXX
```

(5 Digits Reset Password Token)
100,000 options

Brute forcing the "n" successfully allowed me to set new password for any Facebook user.

Found by Anand Prakash, AppSecure

A3 - Excessive Data Exposure

- APIs expose sensitive data of other Users by design

A3 - Excessive Data Exposure

GET /users/717

```
200 OK
{
  "name":"Bob",
  "hobbies":["Banana"],
  "profile_pic":"/bob.jpg",
  "address":"Gru's Mansion, 1000 Evil Rd"
}
```

X

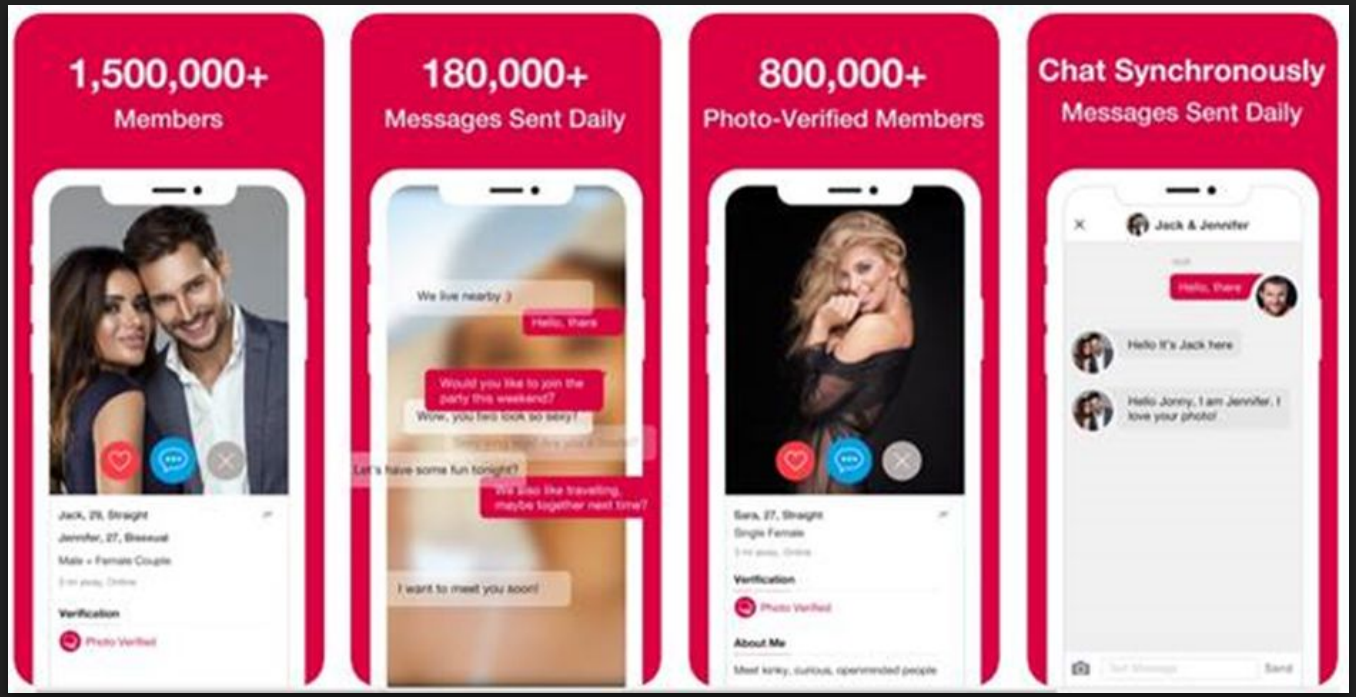
API

Filtering sensitive information on the client side == **BAD IDEA!!**

A3 - Why ?

- API Economy + REST Standard == Generic Endpoints :(
- “to_ json” / “to_string” functions from ORM / Model
- Developers don’t think who’s the consumer

Recent Example - “3fun” app



Found
by Alex Lomas, Pen Test Partners

Found by Alex Lomas, Pen Test Partners

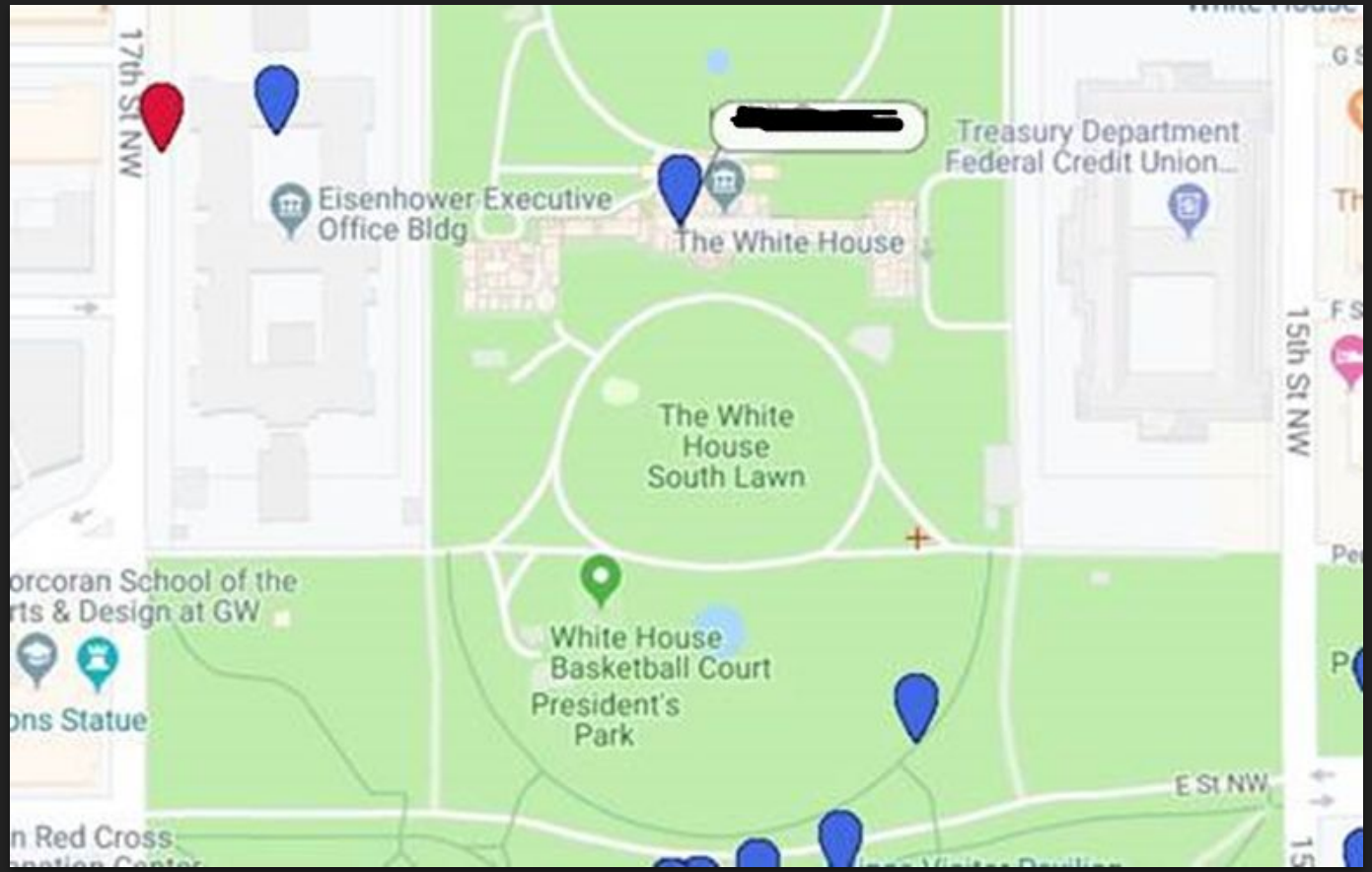
#	Host	Method	URL	Params	Edited	Status	Length	MIME type
322	https://www.go3fun.co	POST	/account_kit_reg	✓		200	447	JSON
325	https://www.go3fun.co	POST	/user/device_token	✓		200	198	JSON
326	https://www.go3fun.co	POST	/user/update	✓		200	265	JSON
327	https://www.go3fun.co	POST	/reset_push_badge			200	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
334	https://www.go3fun.co	POST	/user/update_location	✓		200	198	JSON
338	https://www.go3fun.co	POST	/upload_photo	✓		200	479	JSON
339	https://www.go3fun.co	GET	/i_like_list?from=0&offset=30	✓		200	201	JSON
340	https://www.go3fun.co	GET	/chatted_list			200	201	JSON
341	https://www.go3fun.co	POST	/reset_push_badge			200	198	JSON
344	https://www.go3fun.co	GET	/user/refresh			200	992	JSON
348	https://www.go3fun.co	GET	/matched_list?from=0&offset=30	✓		200	201	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]",
```

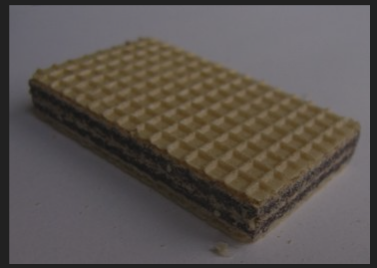
Found by Alex Lomas, Pen Test Partners

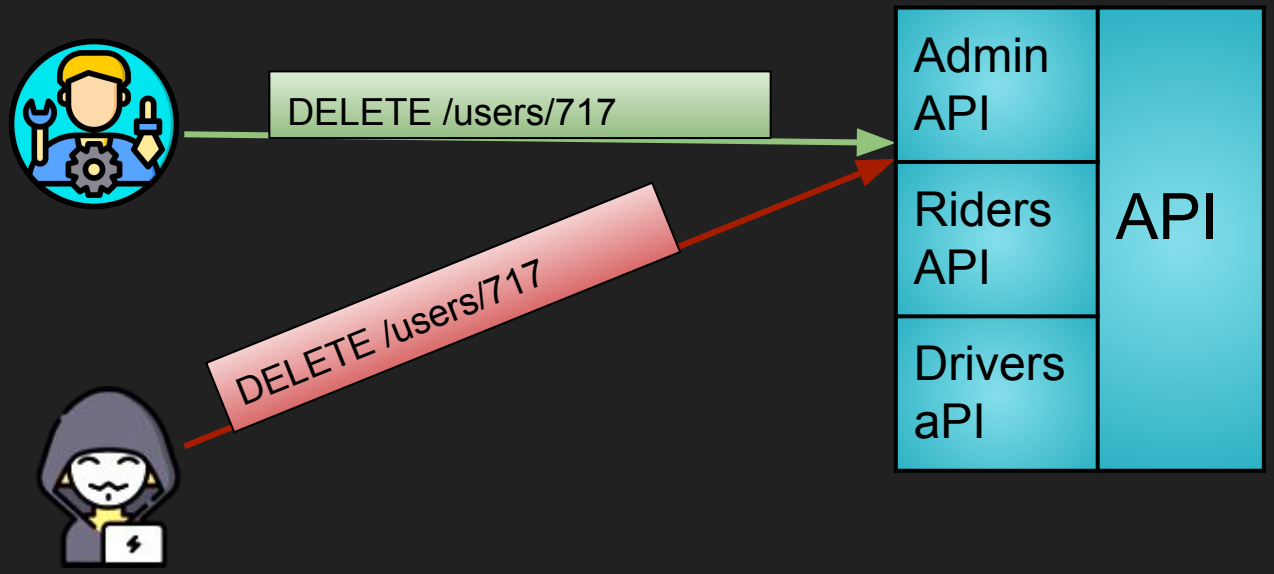


A4 - Lack of Resources & Rate Limiting

- Might lead to DOS
- www.socialnetwork.com/users/list?limit=**99999999**

A5 - BFLA
(Broken Function Level Authorization)





Why in APIs

- Easier to detect in APIs

	Fetch User's Profile (not sensitive function)	**Delete user** (admin function)
Traditional App	GET /app/users_view.aspx?user_id=1337	POST app/admin_panel/users action=delete&user_id=1337
API	**GET** /api/users/1337	**DELETE** /api/users/1337

HARD to predict :(

Very Predictable

Function Level Authorization

- Various ways to implement:
  - Code
  - Configuration
  - API Gateway

- Comlex Roles:
  - Admins / Super-admins / supervisors / riders / drivers

A5 - BFLA - Example - Shopify

@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”

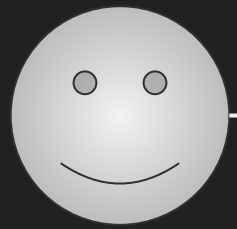
Found by uzsunny
$20,000 bounty on Hackerone

A6 - **Mass Assignment**

“Create_user” flow in traditional apps

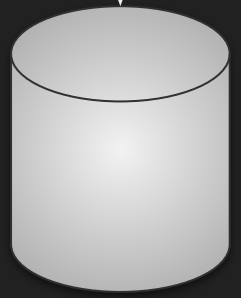
```
User new_user = User();
new_user.first_name = Request.Query["fname"];
new_user.last_name = Request.Query["lname"];
new_user.pass = Request.Query["pass"];
new_user.Save();
```

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



create_user
fname=**inon**&
lname=**shkedy**&
pass=**123456**

APP
Server



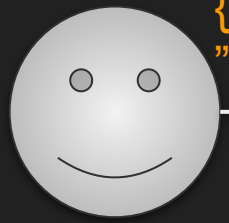
A6 - Mass Assignment

(ORM Black Magic)

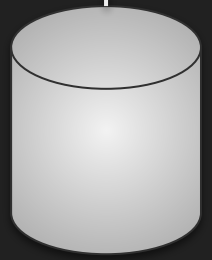
```
@user = User.new(params[:user])
```

ORM
{JSON
AS IS}

POST /users/create
{“user”:{“lname”:”lnon”,”fname”:
”shkedy”,”pass”:”123456”}}



APP
Server



A6 - Mass Assignment

Legit

POST /api/users/new

{“username”:”lnon”, ”pass”:”123456”}

Malicious

POST /api/users/new
{“username”:”lnon”, ”pass”:”123456”, ”role”:”admin”}

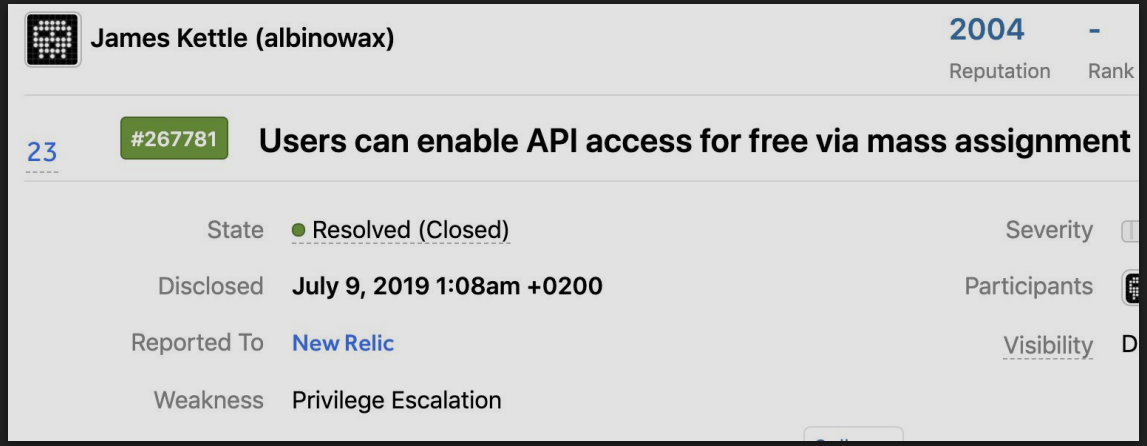
A6 - Why in APIs

- Mass Assignment Ain’t New
- Easier to exploit in APIs
  - Always try with POST, PUT & PATCH
- Don’t guess object properties
  - Just find a GET method which returns them
- Use Mass Assignment to bypass other security controls

```
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:/"
}
```

A6 - Example




```
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** ?

- 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

Developers

Unknown API hosts

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

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

- Excuse me mister , but what the heck is “qa-3-old.app.com” ?

A10 - Insufficient Logging & Monitoring

- Same as A10 (2017)

How to hack APIs?

Pentesters Methodology - **API Mindfulness**

- Beginner's mind (Shoshin) -
  Always **be curious** about APIs
  Understand the business logic by asking meaningful questions
- Wean yourself off GUI
  Don't let fear stop you from generating API calls from scratch
- Use the evaluation phases

F

High-Level Evaluation

Learn:

- REST based ride sharing app
- Has a carpooling feature

Ask:

- What is “VIN”??

```
GET /api/v2/trips/5337

------

200 OK

{
  "id": 5337,
  "payed_by": "10093",
  "price": "10$",
  "src_loc": {
    "lon": 13.36671,
    "lan": 52.54344
  },
  "dest_loc": {
    "lon": 13.31337,
    "lan": 52.5337
  },
  "driver": "e52cdc4b-3d1a-435f-9740-d5329c11d2d4",
  "co_riders": [
    {
      "id": 1001,
      "name": "Hugo Bugo"
    }
  ],
  "payment_options": [
    "01adf0d0-f5e0-4e2f-b8d4-27c4281f27c4"
  ],
  "VIN": "JM1BG2260M0230936"
}
```

Drill Down Evaluation

Learn:

- Trips & users - Numeric ID
- Drivers & payment - GUID

Ask:

- More than one version?
- Payment splitting?!
- Maybe soap?

Do:

- Cause an error:
  /v2/trips/**aaa555**
- Find the payment splitting feature

```
GET /api/v2/trips/5337

------

200 OK

{
  "id": 5337,
  "payed_by": "10093",
  "price": "10$",
  "src_loc": {
    "lon": 13.36671,
    "lan": 52.54344
  },
  "dest_loc": {
    "lon": 13.31337,
    "lan": 52.5337
  },
  "driver": "e52cdc4b-3d1a-435f-9740-d5329c11d2d4",
  "co_riders": [
    {
      "id": 1001,
      "name": "Hugo Bugo"
    }
  ],
  "payment_options": [
    "01adf0d0-f5e0-4e2f-b8d4-27c4281f27c4"
  ],
  "VIN": "JM1BG2260M0230936"
```

Access Control Evaluation

Learn:

- Different user types

Ask:

- Should the last name be exposed?
- Can I be a driver & a rider?
- Support for cookies authz?

Do:

- Identify the session label

```
GET /api/v2/trips/5337

Content-type: application/json
Authorization: Bearer <TOKEN>

------

200 OK

{
  "id": 5337,
  "payed_by": "10093",
  "price": "10$",
  "src_loc": {
    "lon": 13.36671,
    "lan": 52.54344
  },
  "dest_loc": {
    "lon": 13.31337,
    "lan": 52.5337
  },
  "driver": "e52cdc4b-3d1a-435f-9740-d5329c11d2d4",
  "co_riders": [
    {
      "id": 1001,
      "name": "Hugo Bugo"
    }
  ],
  "payment_options": [
    "01adf0d0-f5e0-4e2f-b8d4-27c4281f27c4"
  ]
}
```

Real Attack #1: Food Delivery App

- Background:
  - Food delivery app
  - API seemed to be pretty secured

- Attack Steps:
  - Downloaded an old version of the app
  - Found a niche feature hidden in the GUI – update user's phone number
  - Includes 2 steps

Step	API endpoint	BOLA
1. Send an SMS with a token	`POST /api/v3/<user_phone_num>/update_num` `{"old":"0501113434","new":"050666666"}`	Vulnerable
2. Verify Code	POST /api/v/api/v3/<user_phone_num>/verify_update_number	Not Vulnerable

Attack steps

	How I felt	Step
#1	😈	Found that the token could be used for the login mechanism as well:
#2	😞	“login_with_code” verifies also the device GUID
#3	🤔	Double verification? Sounds like a feature that might have been added recently
#4	🤓	Scanned for old versions of endpoint (v0.0 - v5.0 in the URL)
#5	😈	Found that V2.7 was exposed and didn’t verify the device GUID
#6		**| Full Account Takeover |**

Real Attack #2: Social Network

- Background:
  - Large social network
  - Haven't found interesting in the Web App
  - Endpoint that exposes the user resource from the evaluation phase:

GET /V3/users/<USER_GUID>

Different EP structure. Potentially trigger different code

- Attack Steps:
  - Expanded the attack surface
    - old android version from apkpure.com
  - Found an older implementation of "user" resource
  - Tried to change the method from "POST" to "PUT"
  - Created a request to update my own user
  - **Received 403 ==**
    They implemented "**function level authorization**"

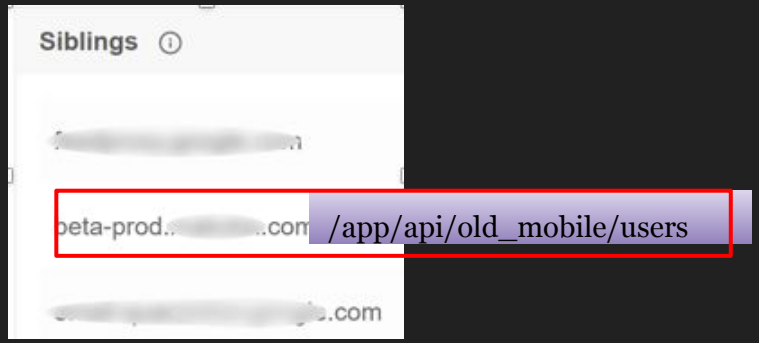
PUT /app/api/old_mobile/users

{"user": <USER_GUID>}

PUT /app/api/old_mobile/users
{"user":<MY_GUID>,
"email":"inon@traceable.ai",

Real Attack #2: Social Network

- Attack Steps #2:
  - Expanded the attack surface
    - Used VirusTotal to find sub domains



  - "beta-prod" exposes the same API endpoint from previous steps
  - The API behaves differently (different headers & error handling)
    - Different behavior == different build / deployment / network flow
  - The "funcion-level authorization" **isn't active** on "beta-prod"
    - API Is **vulnerable** to **A5 (BFLA)**
  - Used the API call from previous step to update any user's email
  - Used the "forgot password" feature to reset the password ==

| FULL ACCOUNT TAKEOVER |

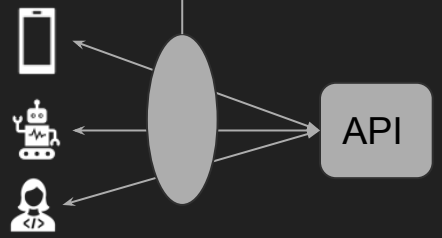
PUT /app/api/old_mobile/users
{"user":"ANY_USER_ID",
"email":"inon@traceable.ai"}

Got Stuck?

Expanding the attack surface

- Find more API endpoints!

	Wet Endpoints	Dry Endpoints
Source	Active traffic from active clients	Full / Partial documentation
Pros	Easier to work with	Easier to find a large amount of them
Cons	You're limited to the endpoints your clients have access to	Hard to work with them



OPENAPI INITIATIVE

swagger

Find more endpoints

Wet Endpoints

- ★ Use multiple clients (mobile/web/web for mobile)
- ★ Use older versions of each client
  - APK Pure (Android)
  - Archive.com (Web Apps)
- ★ Use different hosts that run the same API
  - Use VirusTotal and Censys to find more hosts
- ★ Use different environments (QA/Staging/beta)
- ★ Use Burp “Site View” feature

Dry Endpoints

- ★ Scan client files for strings that look like URLs
  - .js (JS-Scan tool) /.ipa / .apk files
- ★ Look for swagger / WADL files: /swagger.json; /api-docs; /application.wadl; etc..
- ★ Look for exposed API documentation for developers

Bypass Security Controls

- ★ Sometimes non-prod environments (QA, Staging, etc) don’t implement basic security mechanisms

- ★ Different protocols == different implementations.
  - An app can expose REST, SOAP, ElasticSearch, GraphQL and websockets at the same time.
  - Don’t assume they implement the same security mechanisms.

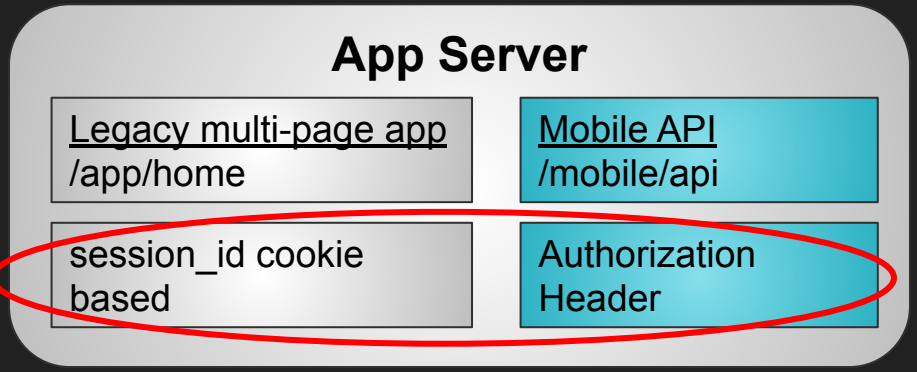
- ★ Find different hosts that expose the same API; They might be deployed with different configurations / versions

Find vulnerable endpoints

- ★ Always look for the most niche features
- ★ Interesting features that tend to be vulnerable:
  - Export mechanism (look for Export Injection)
  - User management, sub-users
  - Custom views of a dashboard
  - Object sharing among users (like sharing a post / receipt)

Mass Assignment + CSRF - Reset User's Password

Auth settings are shared ==
API supports cookies ==
Potential CSRF



★ Let's exploit it to change user's email!

★ "POST /api/update_email" endpoint requires password 😞

★ Anyhow, "PUT /mobile/api/users/me" is vulnerable to **Mass Assignment**

We can update every user's property, including email! 😈

Exploitation

- ★ Target a victim who uses the old app (cookies are stored in his browser)
- ★ Create a malicious HTML page to exploit the CSRF and call

```
PUT /Mobile/users/me
{
"email":"inon@traceable.ai"
}
```

- ★ Send it to the victim, change his email address and reset his password ☺

Questions ?

Follow me on

@InonShkedy