# Introduction to hacking APIs

February 16, 2023

Alejandro Cruz Martínez @pwnedshell
Rubén Santos García @rsgbengi

HackOn 2023

# Agenda




About

Introduction
- What is an API
- CRUD
- API types
- API information
- Methodology

Tools

Recognition
- RESTful recognition
- GraphQL recognition

Vulnerabilities
- Information Disclosure
- Information Disclosure
- Broken Object Level Autorization
- Broken Function Level Autorization (BFLA)
- Mass Assigment
- Broken User Authentication

Clean Up

# About

# About
## $ Whoarewe



- Accenture Cyber Attack Simulator Analysts
- Cybersecurity engineers (URJC)
- Love to play around with offensive security tools and craft our owns
- Managed to get out of the League of Legends
- Lovers of food tiktoks
- One likes anime, the other one doesn't

Alejandro Cruz Martínez

@pwnedshell

Rubén Santos García

@rsgbengii

# About
## $ Where do we work



- Yes, we work at Accenture
- Yes, we actually pentest
  - Web pentesting
  - Wifi pentesting
  - Internal pentesting
  - Phishing campaings
  - OSINT
- We have an incredible team

ACCENTURE SECURITY

# About
## $ What do you need



- Workshop resources and cheatsheets
- `https://github.com/DeathChron/Talks`

# About
## $ What do you need



- Workshop resources and cheatsheets
- `https://github.com/DeathChron/Talks`

# Introduction

# Introduction

## $ What is an API

- **A**pplication **P**rogramming **I**nterface
- Mechanisms that enable two software components to communicate with each other
- CAMARERO!!!

# Introduction
## $ CRUD



- Primary actions and methods used to interact with APIs
  - Best practice **only**

# Introduction

## $ API –type=SOAP



- **S**imple **O**bject **A**ccess **P**rotocol
- XML
- Old and slow like PHP
- One endpoint
  - HTTP POST method
  - Not always HTTP

```
<!-- HTTP POST /soap -->

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope/">
<soap:Body>
  <getUserDetails xmlns:m="https://www.example.com/users">
    <userId>1234</userId>
  </getUserDetails>
</soap:Body>
</soap:Envelope>
```

# Introduction
## $ API –type=SOAP



- **S**imple **O**bject **A**ccess **P**rotocol
- XML
- Old and slow like PHP
- One endpoint
  - HTTP POST method
  - Not always HTTP

```
<!-- HTTP POST /soap -->

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope/">
<soap:Body>
  <getUserDetailsResponse xmlns:m="https://www.example.com/users">
    <user>
      <id>1234</id>
      <name>Rubén</name>
      <group>HackOn</group>
      <password>micontraseña22</password>
    </user>
  </getUserDetailsResponse>
</soap:Body>
</soap:Envelope>
```

# Introduction

## $ API –type=REST



- **RE**presentational **S**tate **T**ransfer
- JSON
- Most popular
- Multiple endpoints
  - Always HTTP (multiple methods)
  - CRUD implementation

{ REST }

```
HTTP GET /getUserDetails?id=1234
```

# Introduction

## $ API –type=REST



- **RE**presentational **S**tate **T**ransfer
- JSON
- Most popular
- Multiple endpoints
  - Always HTTP (multiple methods)
  - CRUD implementation

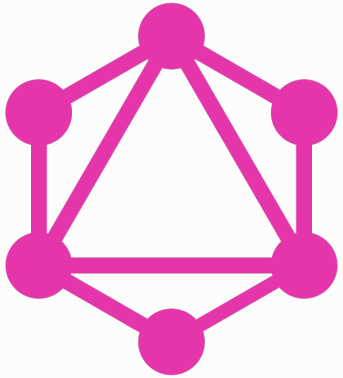
{ REST }

```
{
    "id": 1234,
    "name": "Rubén",
    "group": "HackOn",
    "password": "micontraseña22"
}
```

# Introduction
## $ API –type=GRAPHQL



- Own language
  - "Fresh new"
  - By Facebook
- Ask for exactly what you need
- One endpoint
  - Always HTTP POST method



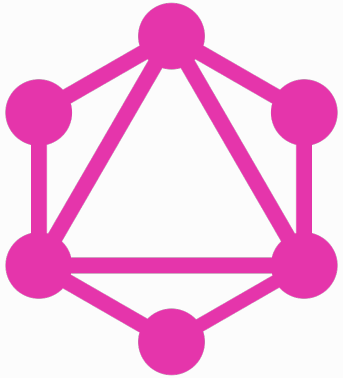
```
# HTTP POST /graphql

query {
  user(id: 1234){
    details {
      id
      name
    }
  }
}
```

# Introduction
## $ API –type=GRAPHQL



- Own language
  - “Fresh new”
  - By Facebook
- Ask for exactly what you need
- One endpoint
  - Always HTTP POST method

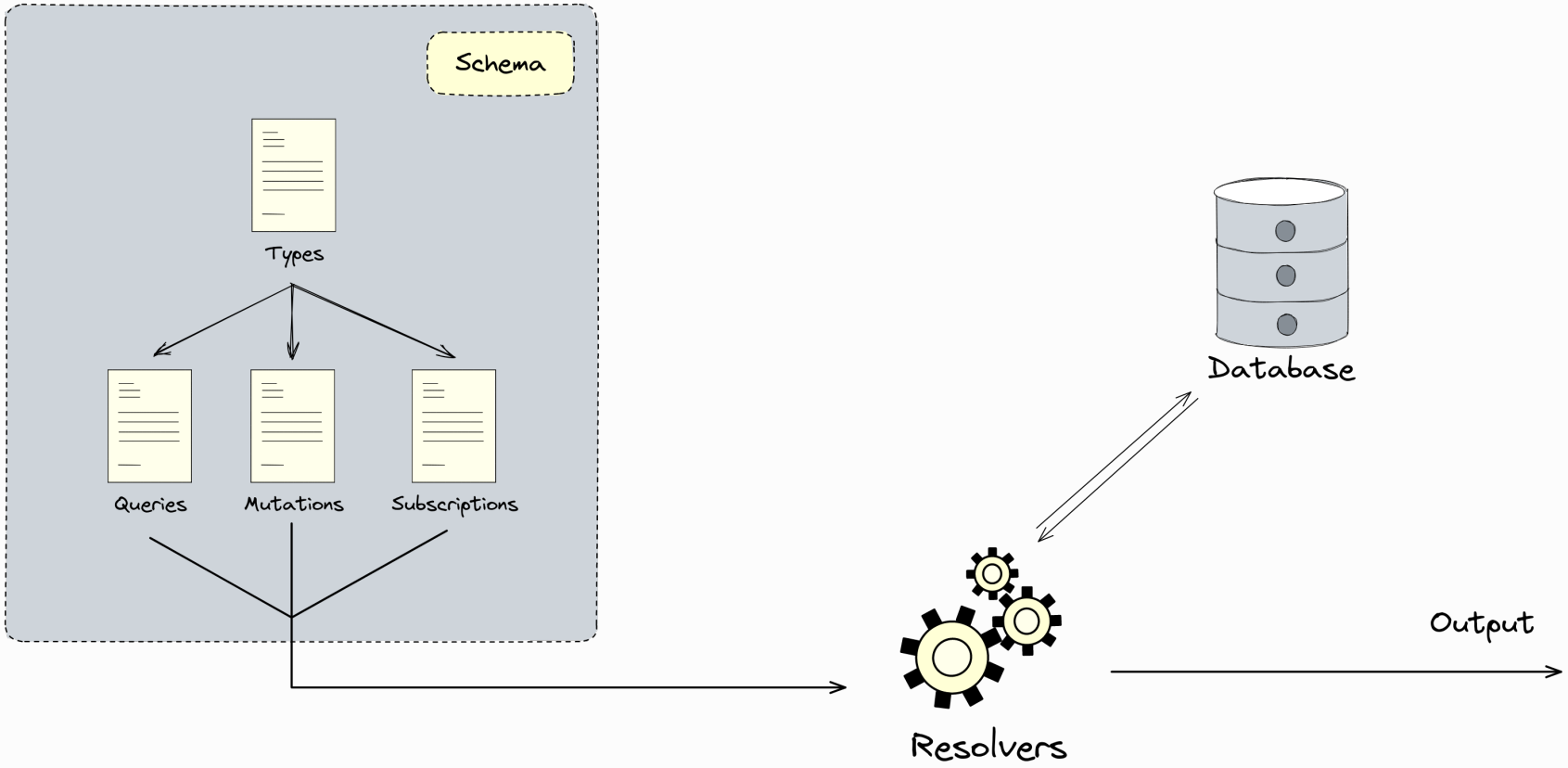


```
{
  "data": {
    "user": {
      "details": {
        "id": 1234,
        "name": "Rubén"
      }
    }
  }
}
```

# Introduction
## $ API –type=GRAPHQL



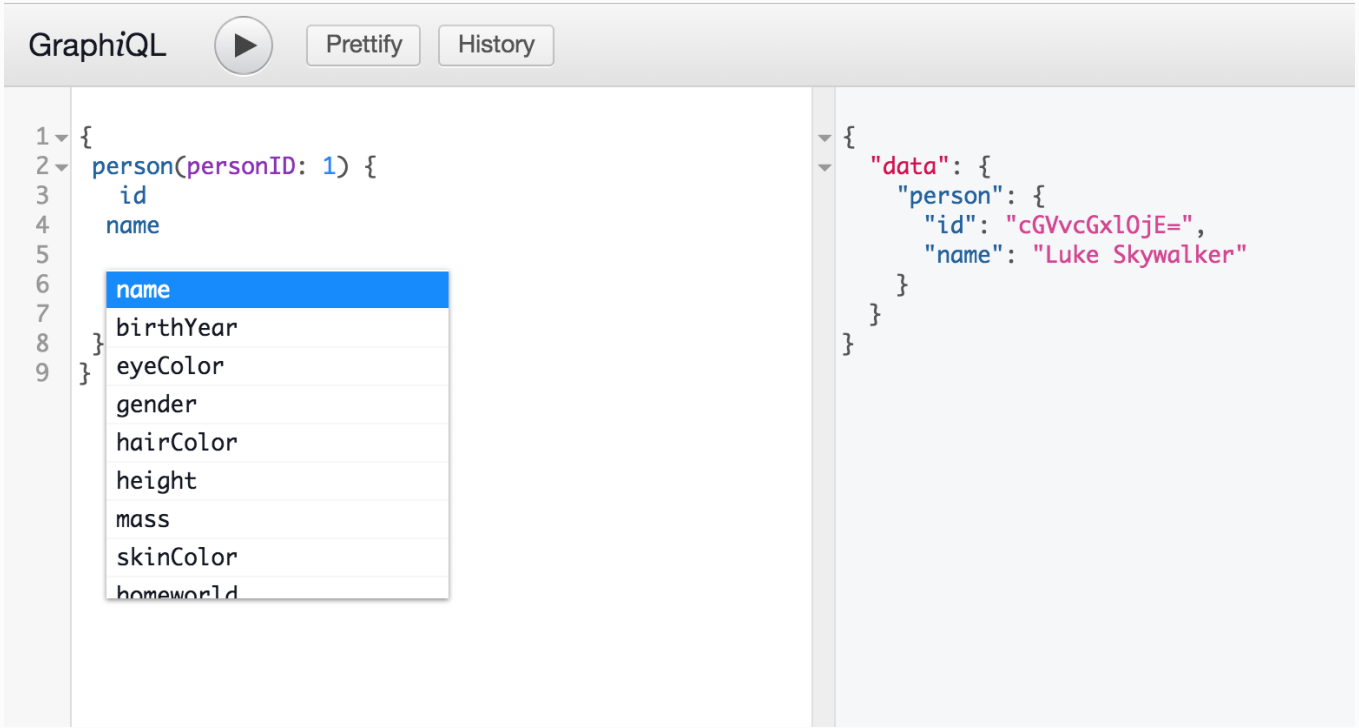
- How does GraphQL works?



# Introduction
## $ API –type=GRAPHQL



- Some GraphQL applications expose **graphical playgrounds**

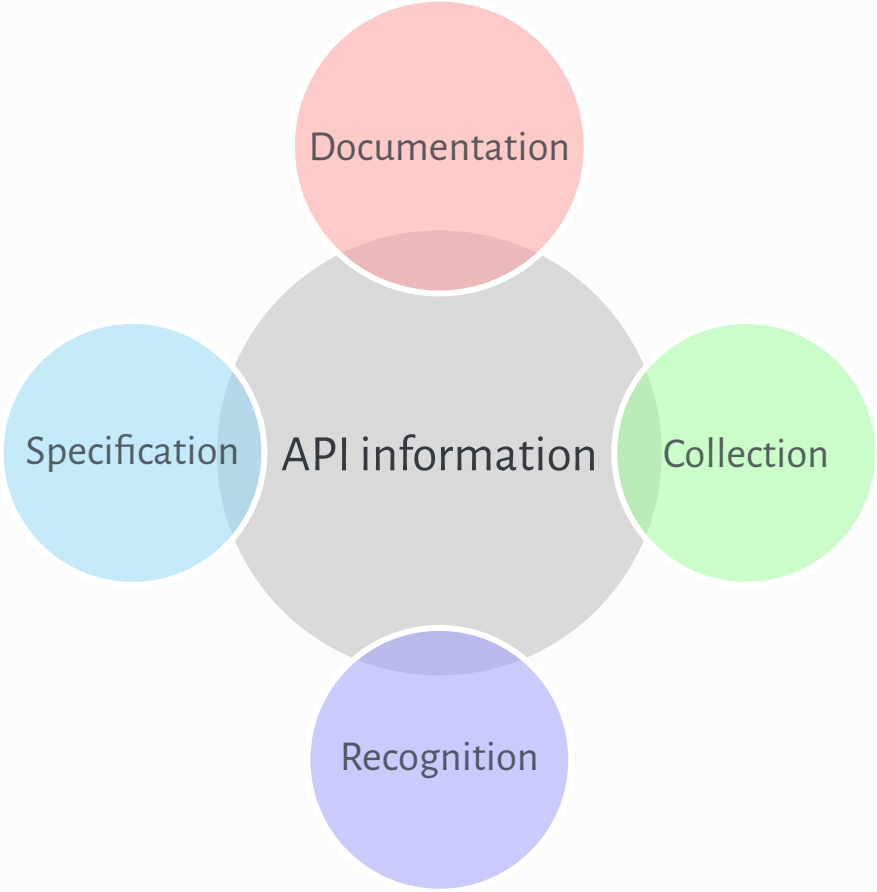


GraphQL demo time!

# Introduction
## $ API information



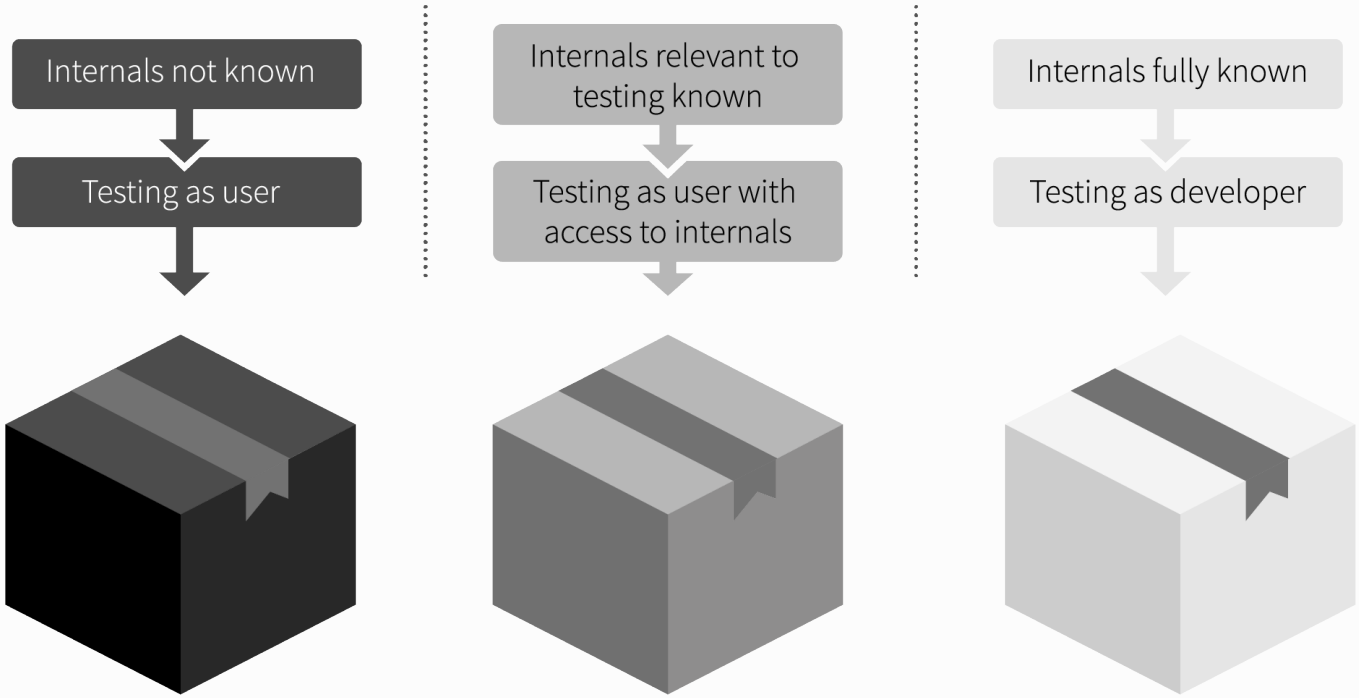
- OpenAPI (swagger) specification
- Application documentation
- Postman collection
- Recognition techniques



# Introduction
## $ Methodology



- **Black box** vs **Grey box** vs **White box**



# Tools

# Tools

## $ echo "The nightmare of installation and configuration"



- Testing tools we will need properly configured and installed:

- Burp Suite
- Ffuf
- Katana
- GraphW00f
- JWT_tool

https://github.com/DeathChron/Talks

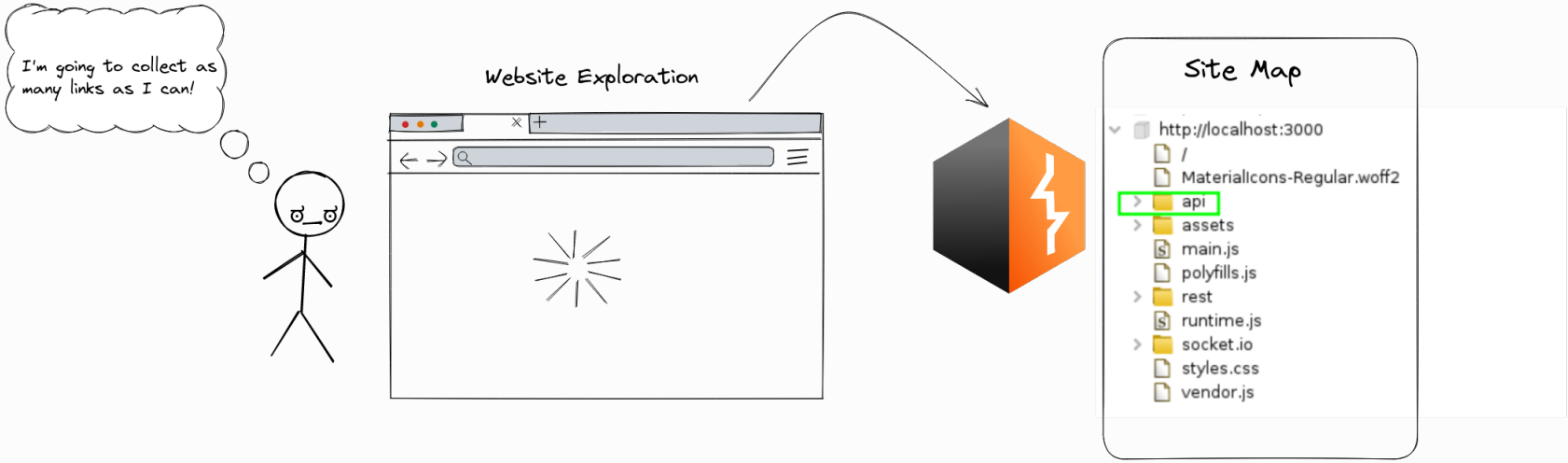
# Recognition

# Recognition

## $ RESTful API: Crawl the webpage



- The **crawling** technique consists on collecting all the possible webpage links
  - We can find endpoints related to API requests

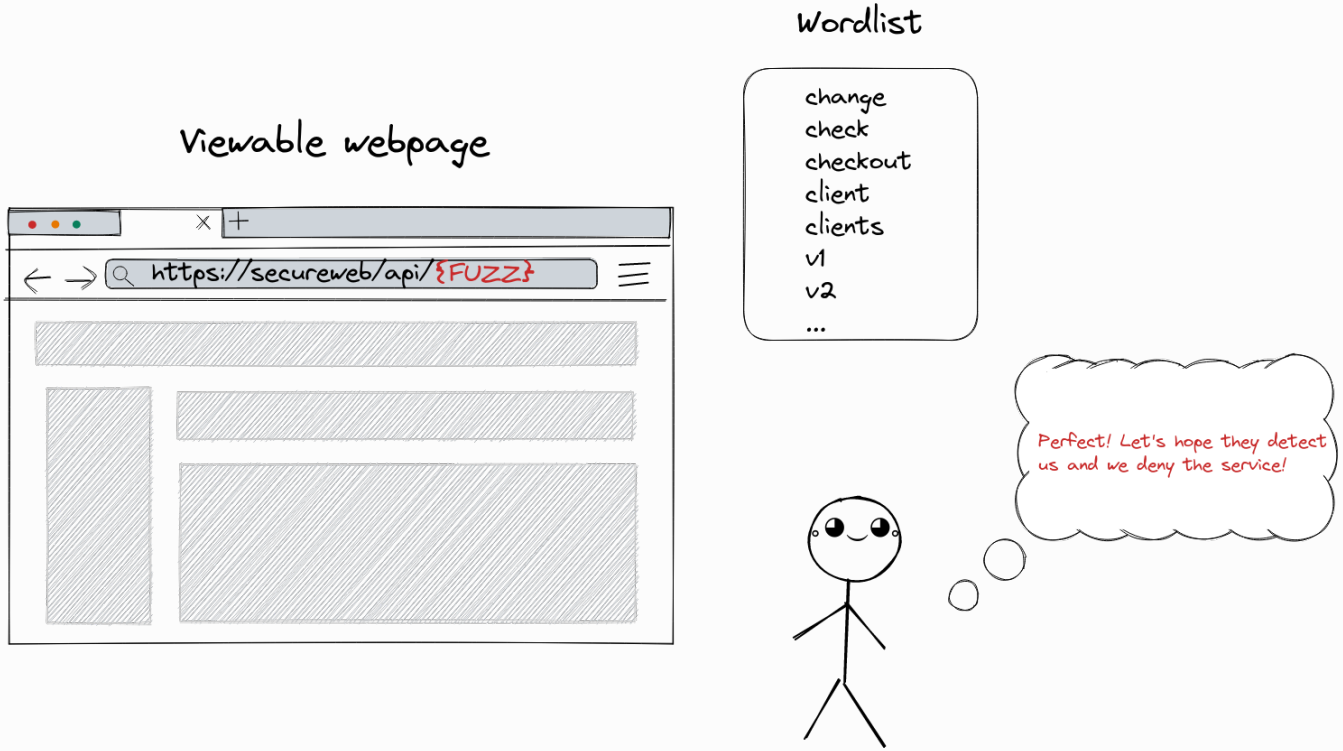


# Recognition

## $ RESTful API: Guess existing resources



- Search for endpoints that might be using the API
  - Usually done by **fuzzing** with a wordlist of commonly used API keywords



# Recognition

## $ RESTful API: Guess existing resources



- This technique can be easily applied with Burp Intruder

**Choose an attack type**

Attack type: Sniper

**Payload Positions**

Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target: http://localhost:3000

```
1 GET /api/§§ HTTP/1.1
2 Host: localhost:3000
3 Accept-Encoding: gzip, deflate
4 Accept: */*
5 Accept-Language: en-US;q=0.9,en;q=0.8
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.5249.62 Safari/537.36
7 Connection: close
8 Cache-Control: max-age=0
9
10
```

# Recognition

## $ RESTful API: Guess existing resources



- This technique can be easily applied with Burp Intruder

| Request | Payload | Status | Error | Timeout | Length | Comment |
|---|---|---|---|---|---|---|
| 0 | | 500 | | | 3346 | |
| 1 | Users | 401 | | | 1290 | |
| 2 | accelerate | 500 | | | 3366 | |
| 3 | acquire | 500 | | | 3360 | |
| 4 | activate | 500 | | | 3362 | |
| 5 | adapt | 500 | | | 3356 | |
| 6 | add | 500 | | | 3352 | |
| 7 | adjust | 500 | | | 3358 | |
| 8 | admin | 500 | | | 3356 | |
| 9 | alert | 500 | | | 3356 | |
| 10 | annotate | 500 | | | 3362 | |
| 11 | anticipate | 500 | | | 3366 | |
| 12 | apply | 500 | | | 3356 | |

Request Response

Pretty Raw Hex

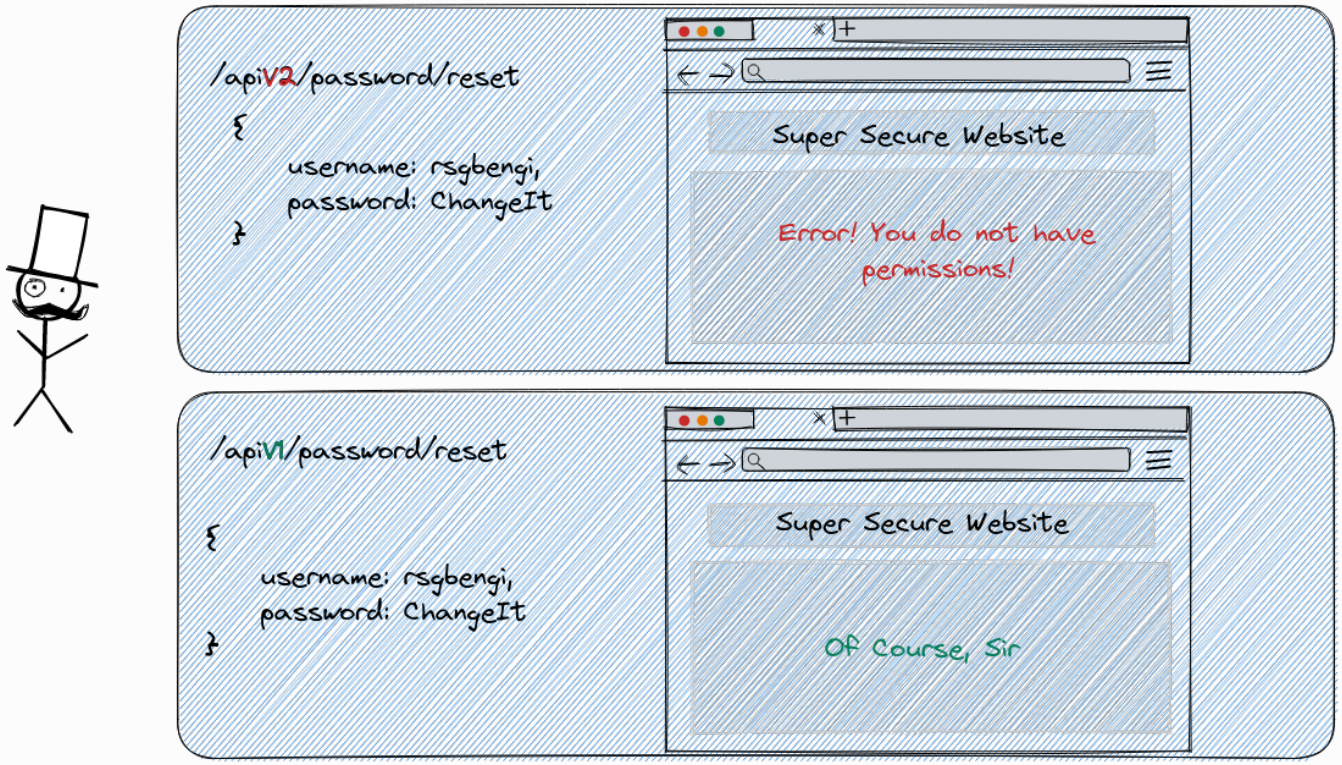
```
1 GET /api/Users HTTP/1.1
2 Host: localhost:3000
3 Accept-Encoding: gzip, deflate
4 Accept: */*
5 Accept-Language: en-US;q=0.9,en;q=0.8
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.5249.62 Safari/537.36
7 Connection: close
8 Cache-Control: max-age=0
9
10
```

# Recognition

## $ RESTful API: API versions



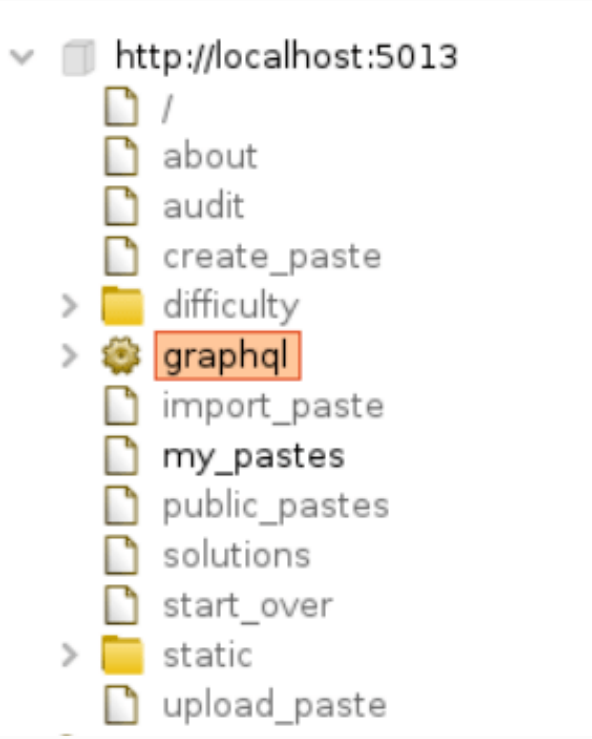
- Endpoints with newer versions may be patched for certain vulnerabilities, but not the same endpoints with older versions that are still public



# Recognition
## $ GraphQL: Detecting GraphQL



- To locate the graphql endpoint, we can use GraphW00f or Burp's crawler




```
┌──(rsgbengi㉿kali)-[~/tools/graphw00f]
└─$ python3 main.py -d -t http://127.0.0.1:5013/

                +-------------------+
                |     graphw00f     |
                +-------------------+
                  ***             ***
                **                  **
              **                      **
    +--------------+              +--------------+
    |    Node X    |              |    Node Y    |
    +--------------+              +--------------+
                  ***             ***
                     **         **
                       **     **
                    +------------+
                    |   Node Z   |
                    +------------+

                graphw00f - v1.1.8
          The fingerprinting tool for GraphQL
           Dolev Farhi <dolev@lethalbit.com>

[*] Checking http://127.0.0.1:5013//
[*] Checking http://127.0.0.1:5013//graphql
[!] Found GraphQL at http://127.0.0.1:5013//graphql
```

# Recognition
## $ GraphQL: Fingerprinting GraphQL



- It is interesting to know the technology used by the API we are attacking
  - For this, we can make use once again of GraphW00f

```
┌──(rsgbengi㉿kali)-[~/tools/graphw00f]
└─$ python3 main.py -f -t http://localhost:9000/graphql

                +-------------------+
                |     graphw00f     |
                +-------------------+
                  ***             ***
                **                  **
              **                      **
    +--------------+              +--------------+
    |    Node X    |              |    Node Y    |
    +--------------+              +--------------+
                  ***             ***
                     **         **
                       **     **
                    +------------+
                    |   Node Z   |
                    +------------+

                graphw00f - v1.1.8
           The fingerprinting tool for GraphQL
            Dolev Farhi <dolev@lethalbit.com>

[*] Checking if GraphQL is available at http://localhost:9000/graphql...
[!] Found GraphQL.
[*] Attempting to fingerprint...
[*] Discovered GraphQL Engine: (AWS AppSync)
[!] Attack Surface Matrix: https://github.com/nicholasaleks/graphql-threat-matrix/blob/maste
[!] Technologies:
[!] Homepage: https://aws.amazon.com/appsync
[*] Completed.
```

# Recognition

## $ How to



- **Black box** vs **Grey Box** vs **White box**

## Tools for recognition

1. Burp Suite's Crawler
2. Burp Suite's Intruder
3. GraphW00f
4. Ffuf
5. Katana

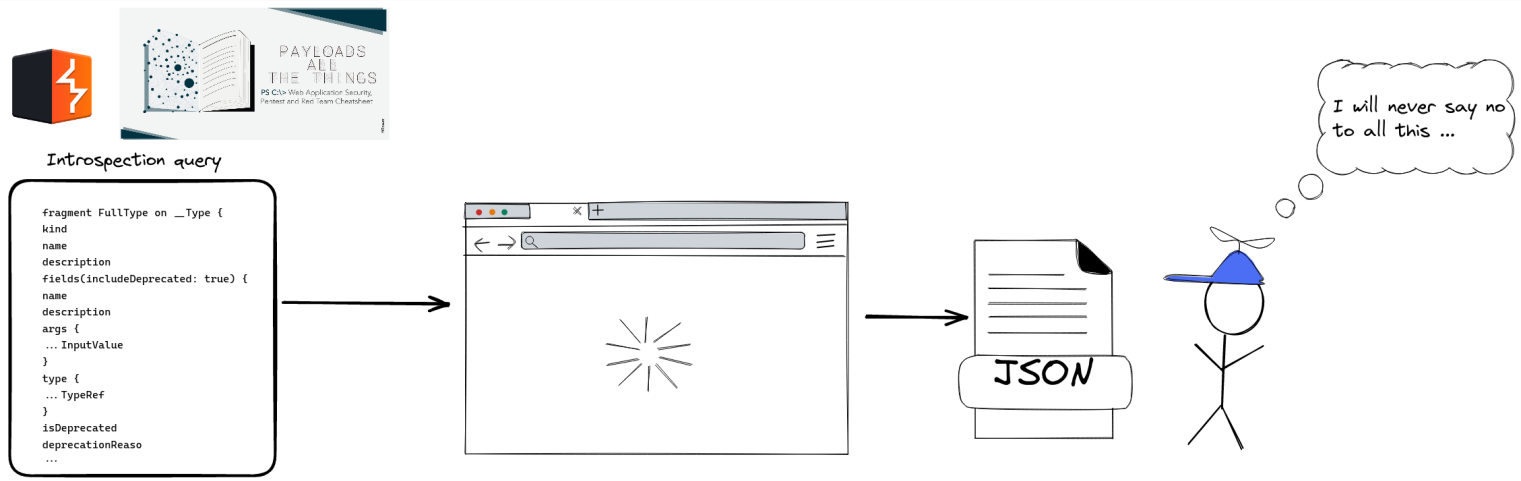
Crawling and fuzzing playground time!

# Recognition

## $ GraphQL: Introspection query



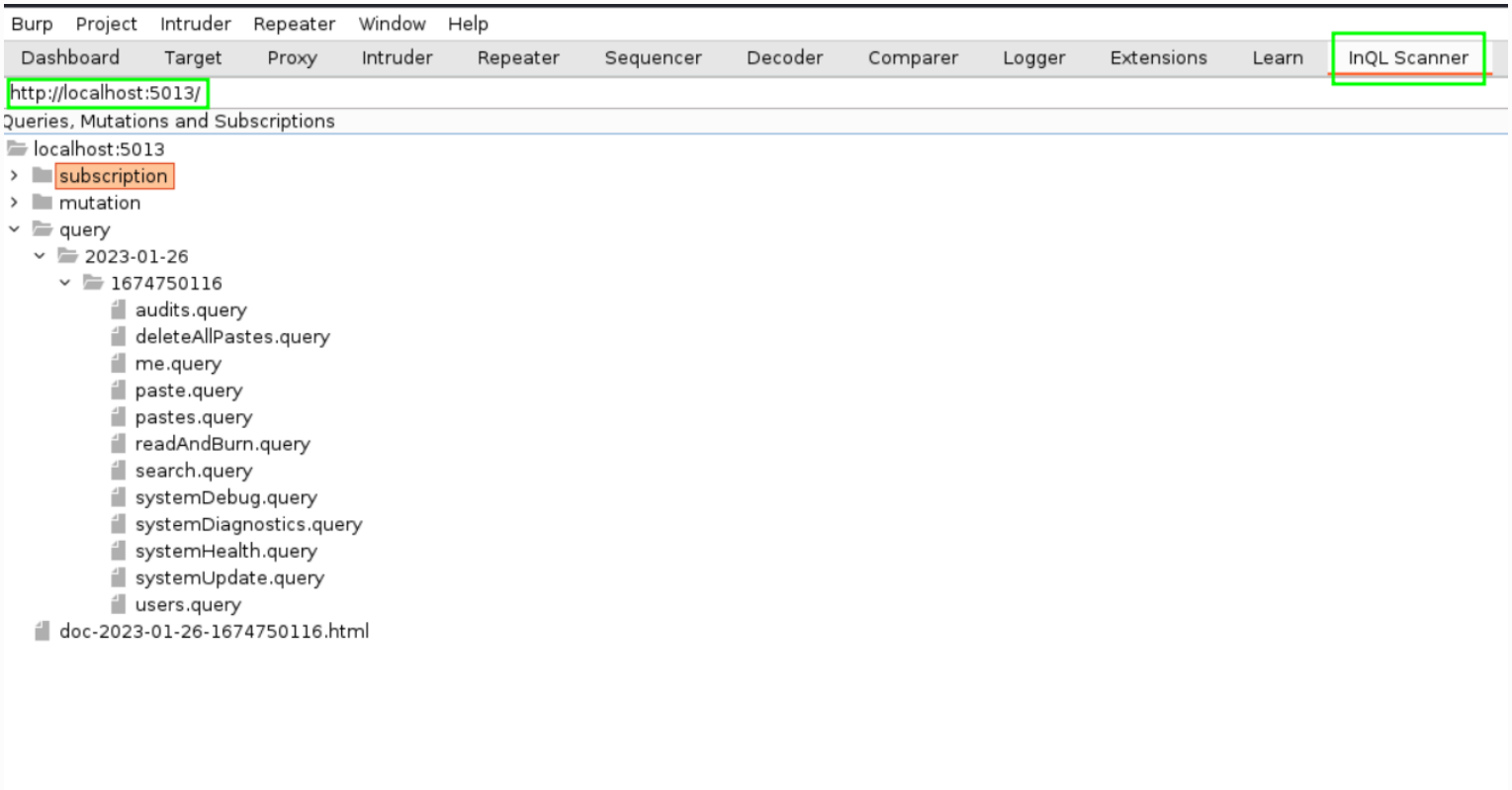
- **Introspection** is the ability to query which resources are available in the current API schema
- Given the API, introspection can help us know all the queries, types, fields, and directives the API supports



# Recognition
## $ GraphQL: Introspection query



- To obtain all the queries that can be made by GraphQL in an application, we can make use of the Burp extension InQL

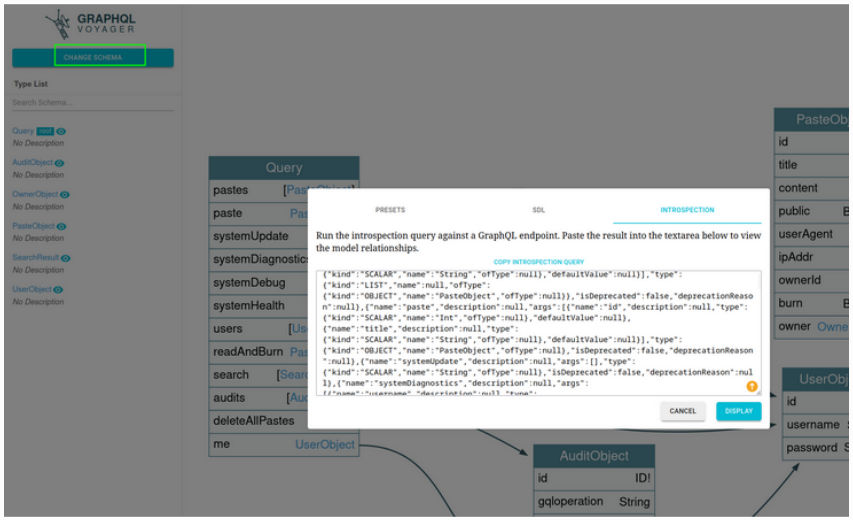


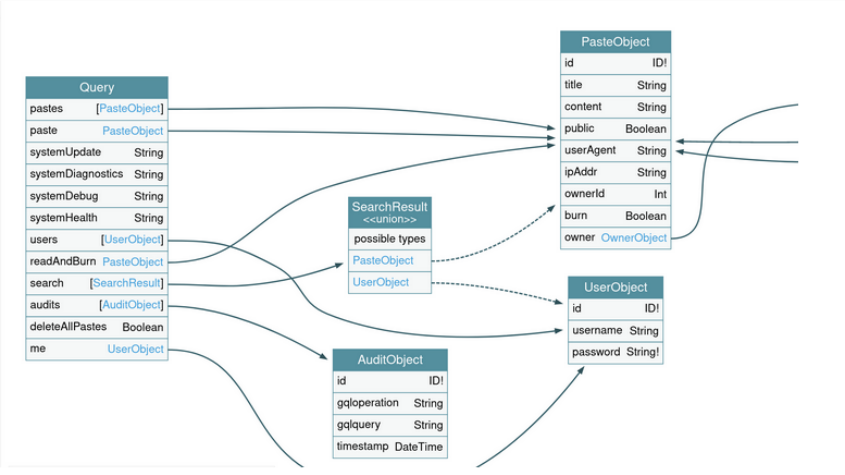
# Recognition

## $ GraHQL: Introspection query



- We can represent the data obtained by the introspection query using the GraphQL Voyage tool



GraphQL recognition and introspection playground time!

# Vulnerabilities

# Vulnerabilities

## $ Common ones



### OWASP Top 10 API - 2019

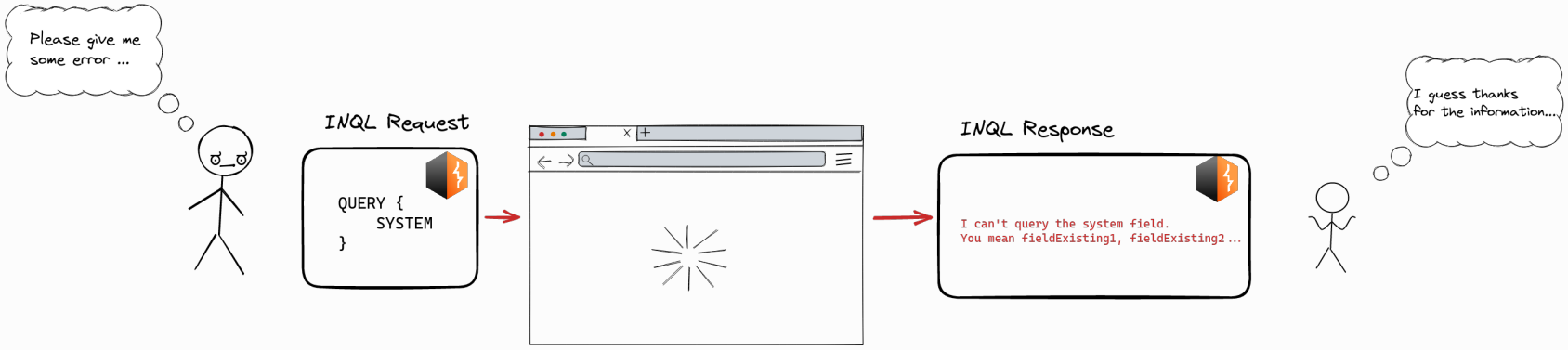
1. **Broken Object Level Authorization**
2. **Broken User Authentication**
3. **Excessive Data Exposure**
4. Lack of Resources & Rate Limiting
5. **Broken Function Level Authorization**
6. **Mass Assignment**
7. Security Misconfiguration
8. Injection
9. Improper Assets Management
10. Insufficient Logging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

# Vulnerabilities

## $ Information Disclosure :: GraphQL Field Suggestions



- So what if the application is forbidden to use introspection queries?
  - By using queries that do not exist, GraphQL can return information about existing queries



# Vulnerabilities

## $ Information Disclosure :: GraphQL Field Suggestions | How to



- **Black box** vs **Grey Box** vs **White box**

## Tools for testing GraphQL Field Suggestions

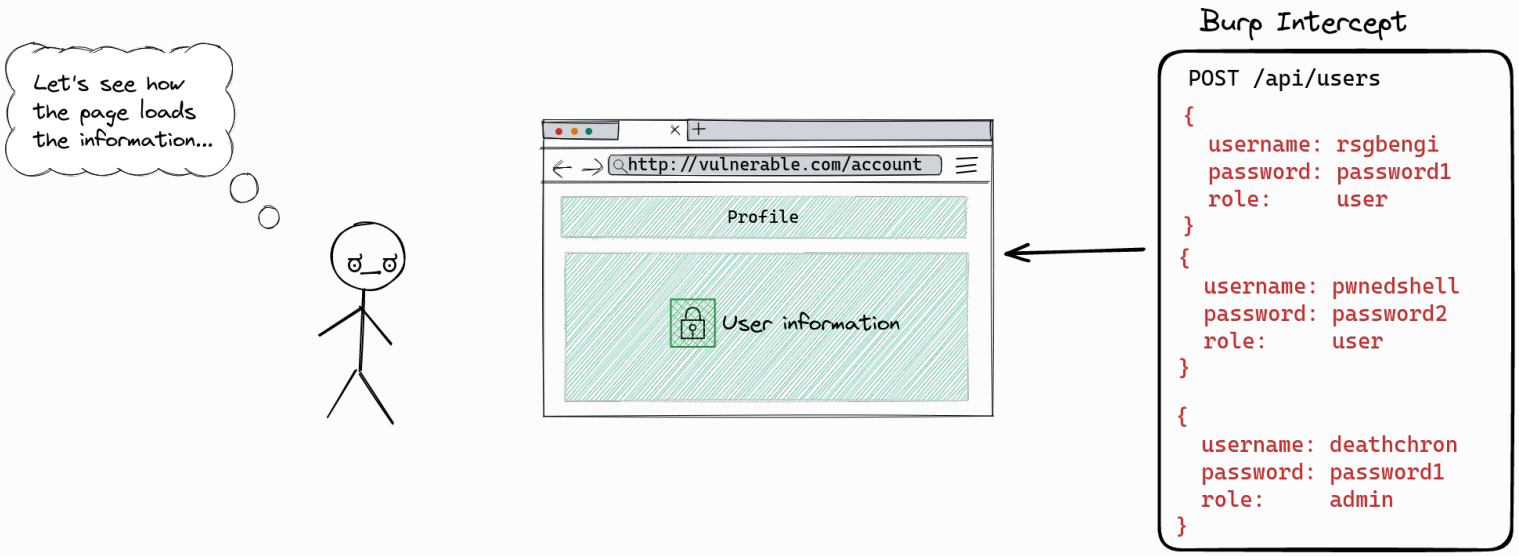
1. Burp Suite's InQL plugin
2. GraphQLMap
3. Clairvoyance

# Vulnerabilities

## $ Excessive data exposure



- APIs may return a lot of unnecessary data in response and filter it on client-side to display only required data
  - An attacker can easily bypass client-side filtering by using a proxy tool such as BurpSuite and access the excessive data returned to the response



# Vulnerabilities

## $ Excessive data exposure | How to



- **Black box** vs **Grey Box** vs **White box**

## Tools for testing Excessive data exposure

1. Burp Suite's AutoRepeater plugin
2. Firefox containers
3. Ffuf
4. Postman

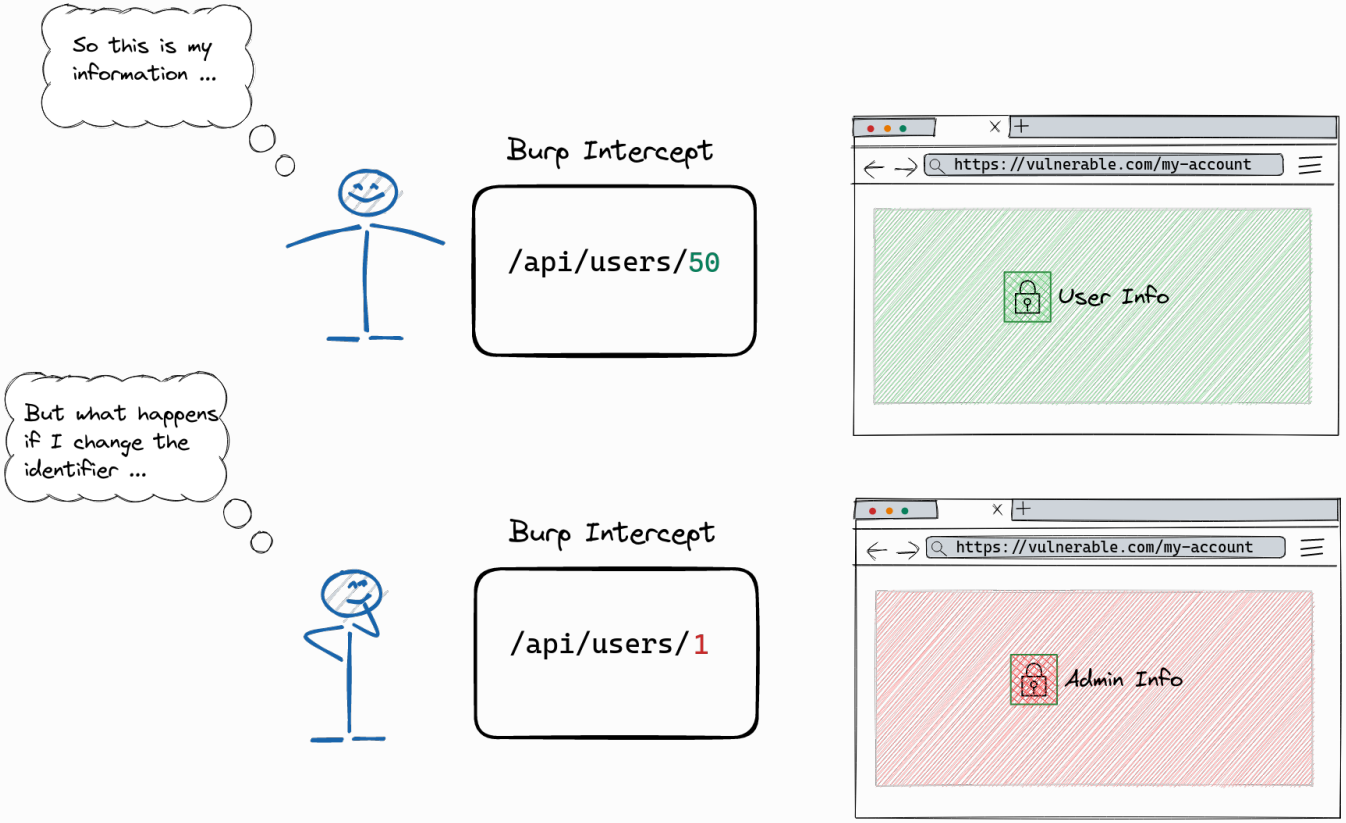
Information Disclosure playground time!

# Vulnerabilities

## $ Broken Object Level Autorization (BOLA)



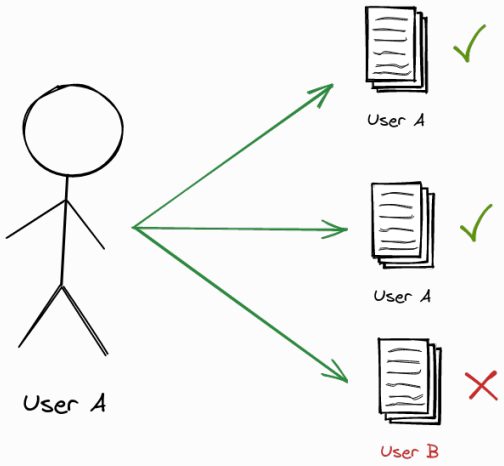
- When a consumer have access to resources which is not authorized to access

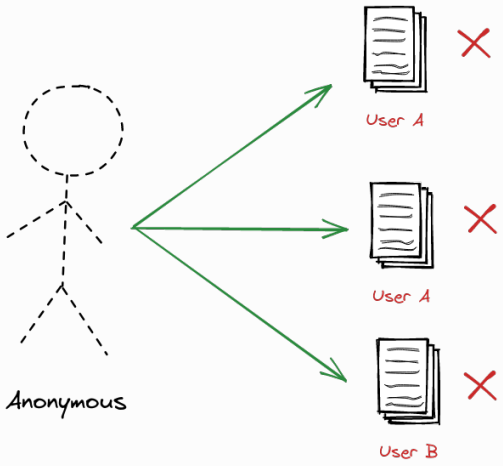


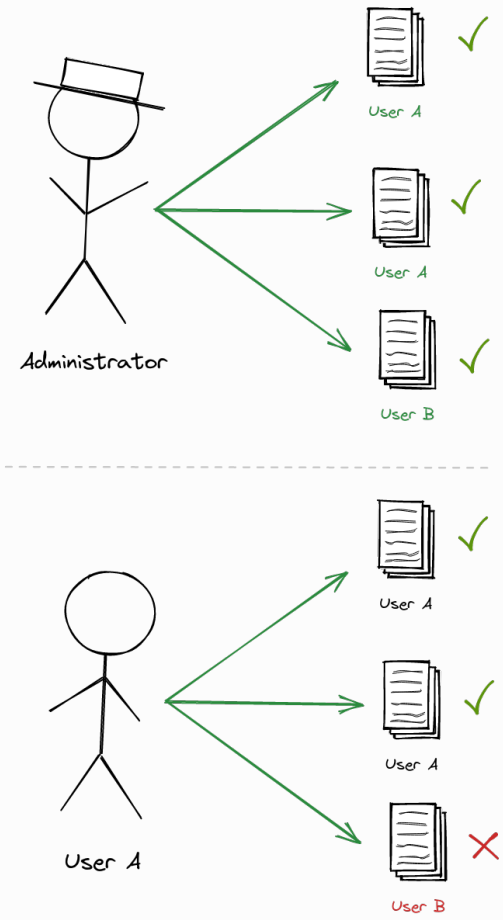
# Vulnerabilities

## $ Broken Object Level Autorization (BOLA) | Types





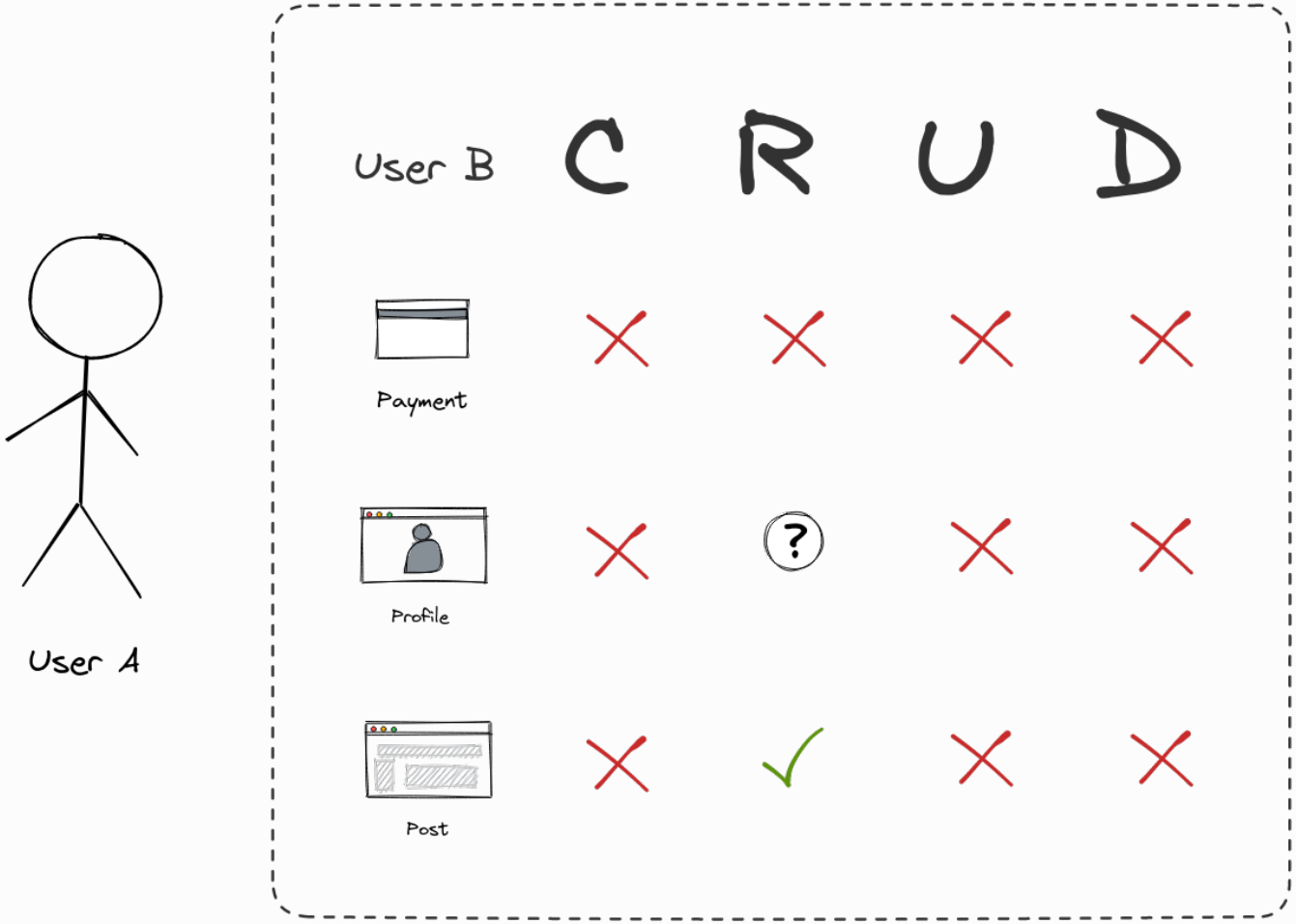

1. User
2. Unauthenticated
3. Permission

# Vulnerabilities

## $ Broken Object Level Autorization (BOLA) | CRUD map



- CRUD testing map

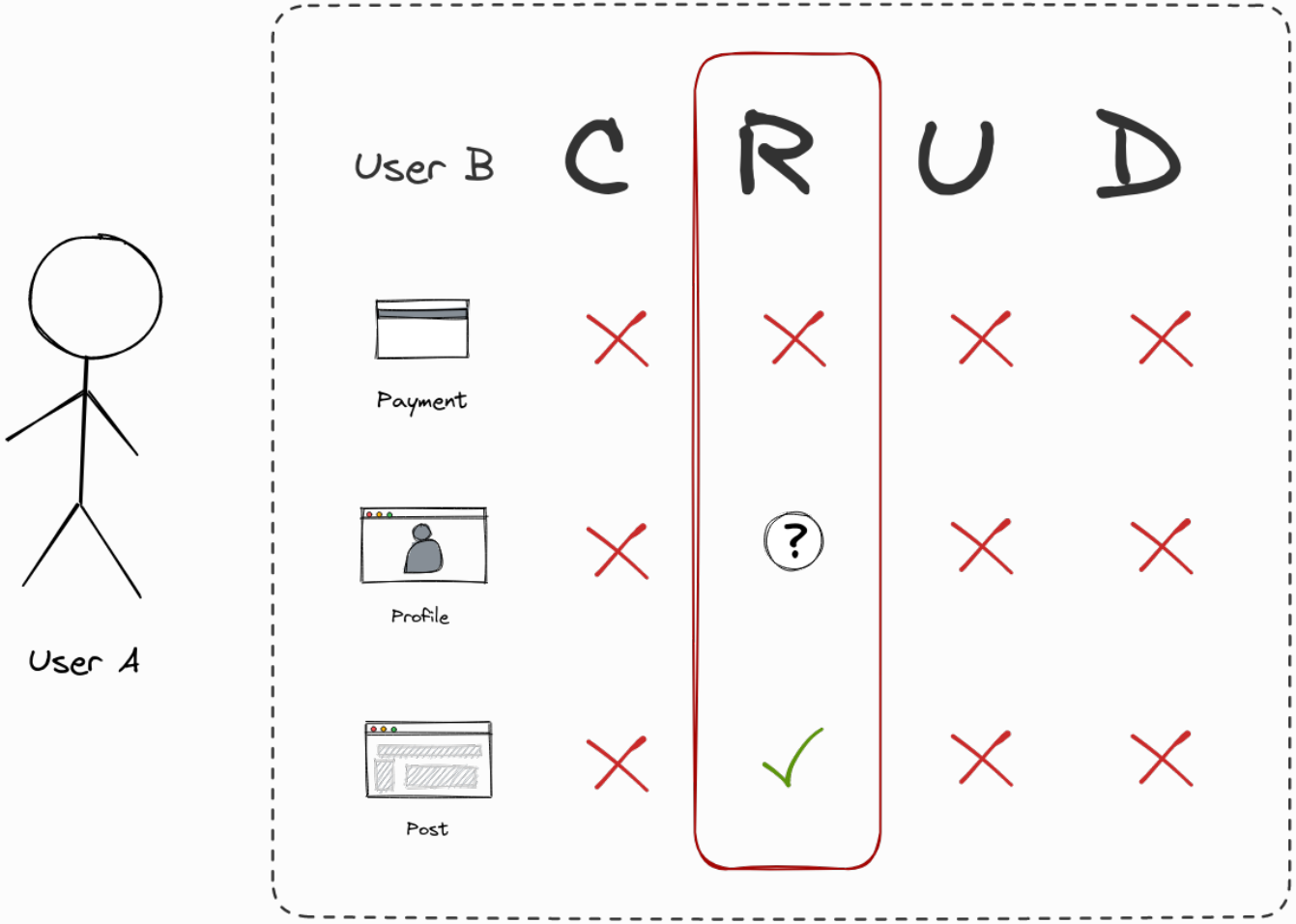


# Vulnerabilities

## $ Broken Object Level Autorization (BOLA) | CRUD map



- BOLA testing map



# Vulnerabilities

## $ Broken Object Level Autorization (BOLA) | Testing



| Type | Captured request | BOLA request |
|---|---|---|
| **Predictable ID** | GET /api/v1/account/**2** | GET /api/v1/account/**1** |
| **ID combo** | GET /api/v1/**UserA**/data/**2** | GET /api/v1/**UserB**/data/**3** |
| **Integer as ID** | POST /api/v1/account/<br>{"user": **2**} | POST /api/v1/account/<br>{"user": **1**} |
| **Email as user ID** | POST /api/v1/account/<br>{"email": "**userA**@email.com"} | POST /api/v1/account/<br>{"email": "**userB**@email.com"} |
| **Group ID** | GET /api/v1/group/**CompanyA** | GET /api/v1/group/**CompanyB** |
| **Group and user combo** | POST /api/v1/group/**CompanyA**<br>{"email": "**userA**@**CompanyA**.com"} | POST /api/v1/group/**CompanyB**<br>{"email": "**userB**@**CompanyB**.com"} |
| **Nested object** | POST /api/v1/user/checking<br>{"user": **2**} | POST /api/v1/user/checking<br>{"user": **{"user": 1}**} |
| **Multiple objects** | POST /api/v1/user/checking<br>{"user": **2**} | POST /api/v1/user/checking<br>{"user": 2, **"user": 1**} |

# Vulnerabilities

## $ Broken Object Level Autorization (BOLA) | How to



- **Black box** vs **Grey Box** vs **White box**

## Tools for testing BOLA

1. Burp Suite's Autorize plugin
2. Burp Suite's AutoRepeater plugin
3. Burp Suite's InQL plugin
4. Firefox containers
5. Ffuf
6. Postman

BOLA playground time!

# Vulnerabilities

## $ Broken Function Level Autorization (BFLA)



- When a consumer is able to access API functionality of a group or role other than their own

Burp Intercept

```
POST /api/{user}

{
  username: rsgbengi
  password: password1
  role:     user
}
```

Burp Intruder

```
POST /api/user                404
POST /api/profile             404
POST /api/credentials         404
                ...
POST /api/adminRemoveUser     200
```

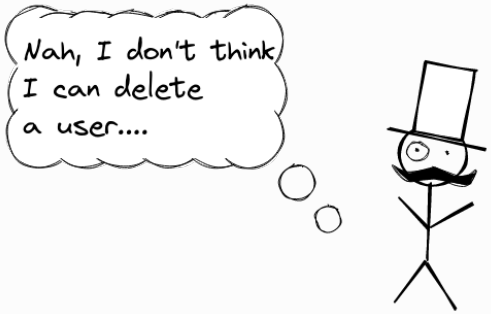
Burp Repeater

```
POST /api/adminRemoveUser

  {
    username: pwnedshell
    password: password1
    role:     admin
  }
```

```
200 OK

of course master
...
```

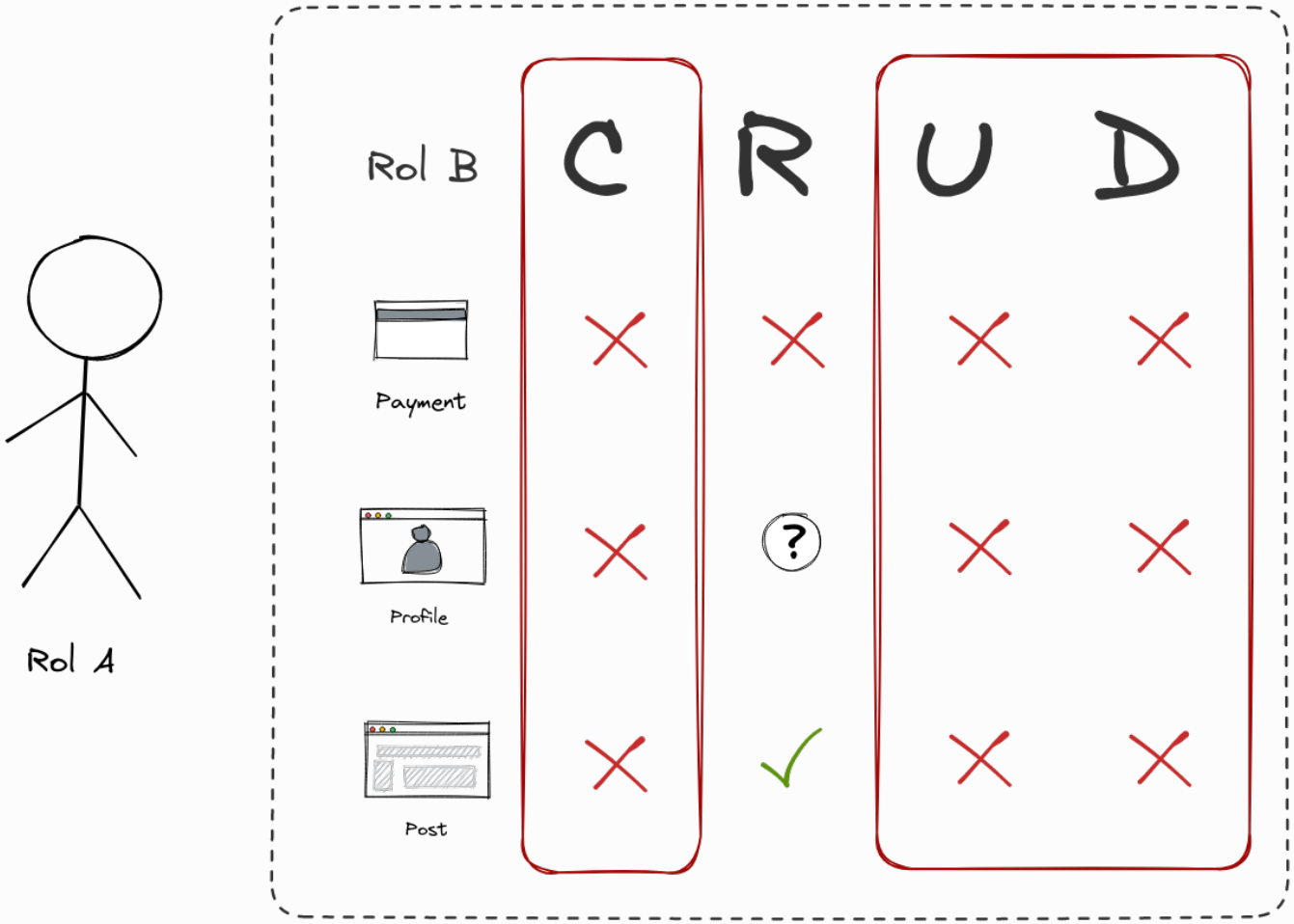


# Vulnerabilities

## Broken Function Level Autorization (BFLA) | CRUD map



- BFLA testing map



# Vulnerabilities

## $ Broken Function Level Autorization (BFLA) | How to



- **Black box** vs **Grey Box** vs **White box**

## Tools for testing BFLA

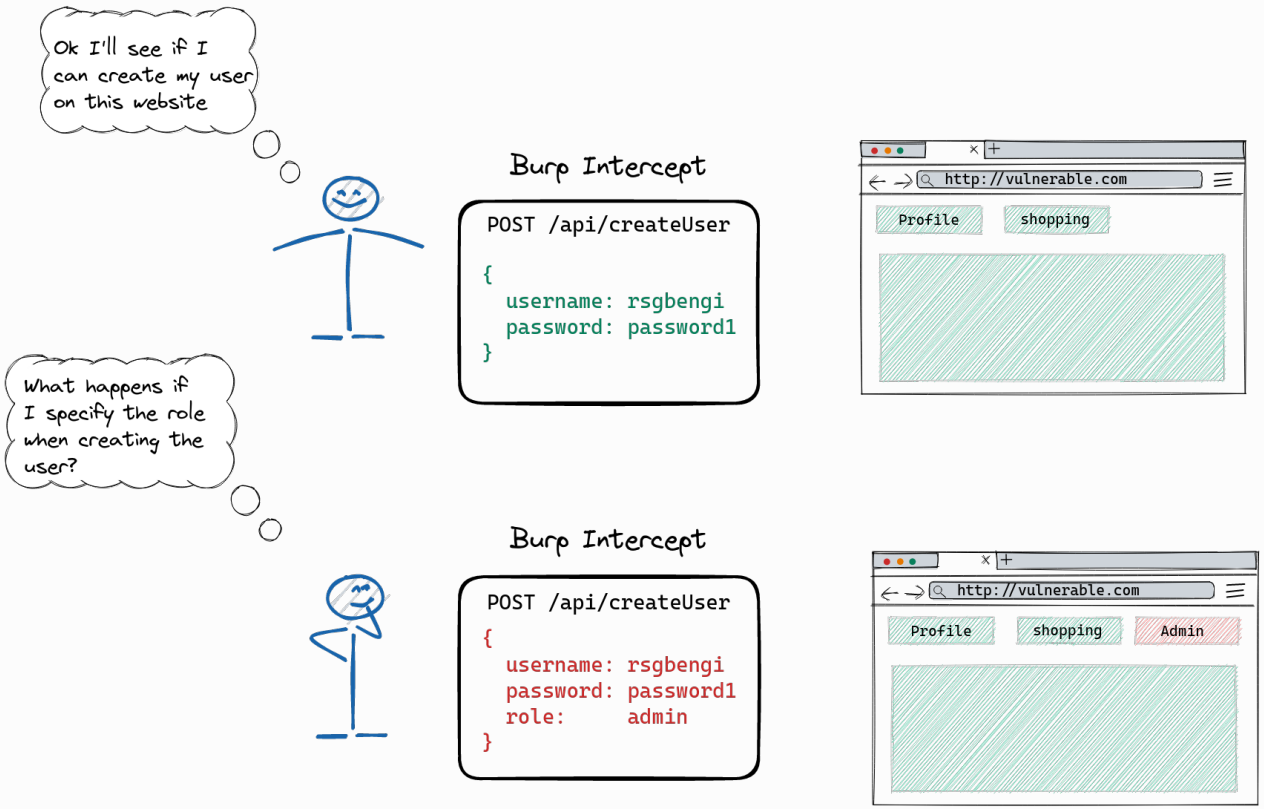
1. Burp Suite's Autorize plugin
2. Burp Suite's AutoRepeater plugin
3. Burp Suite's InQL plugin
4. Firefox containers
5. Ffuf
6. Postman

BFLA playground time!

# Vulnerabilities
## $ Mass Assigment



- When a consumer includes more parameters in their requests than the application intended and the application assign them to internal objects
  - Allows an attacker to modify data and/or increase privileges



# Vulnerabilities

## $ Mass Assigment | How to



- **Black box** vs **Grey Box** vs **White box**

## Tools for testing Mass Assigment

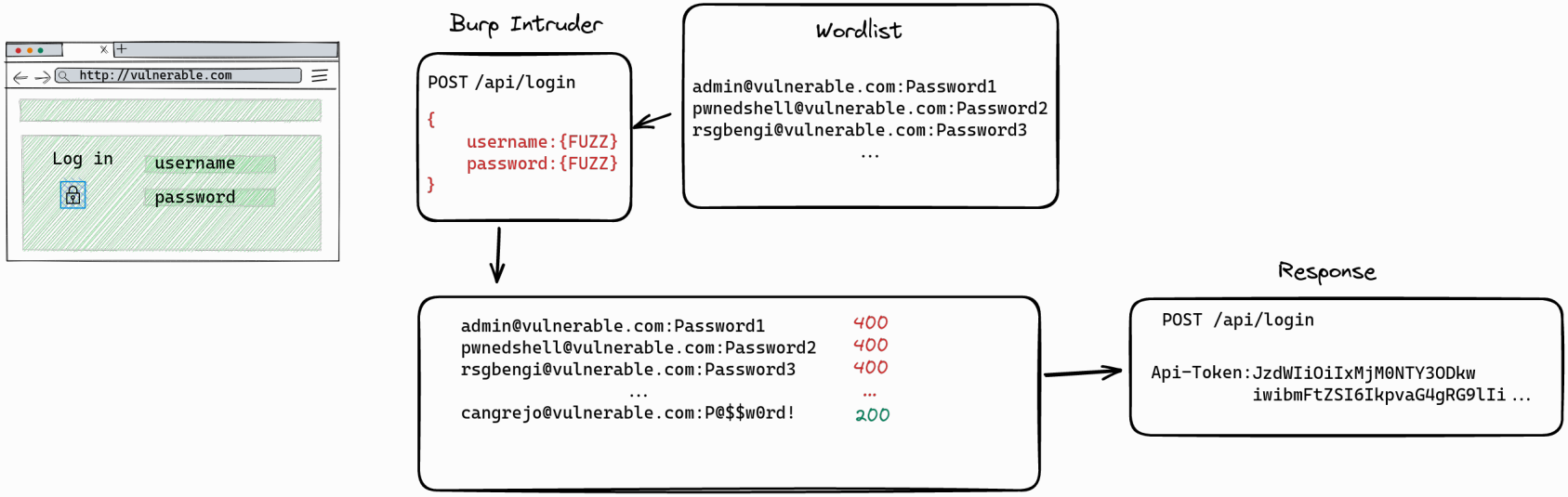
1. Arjun
2. Burp Suite's Intruder
3. Ffuf
4. Postman

Mass Assigment playground time!

# Vulnerabilities
## $ Broken User Authentication



- Authentication process could be compromissed in many different ways
  - Data breaches
  - Insecure mechanisms
  - Incorrect validation

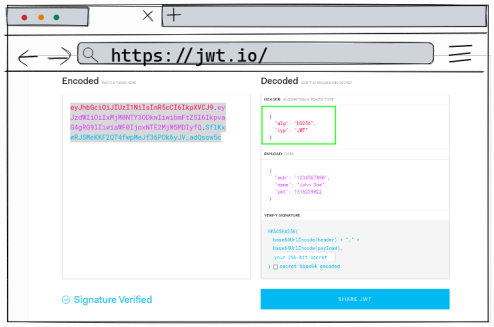


# Vulnerabilities
## $ Broken User Authentication

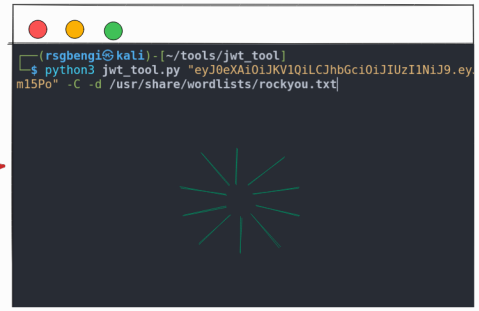


- How to abuse JWT authentication:
  1. Identify the encryption algorithm.
  2. Crack the JWT to obtain the password.
  3. Modify some parameter to escalate privileges.
  4. Create a new JWT with the modified parameter and the cracked password.

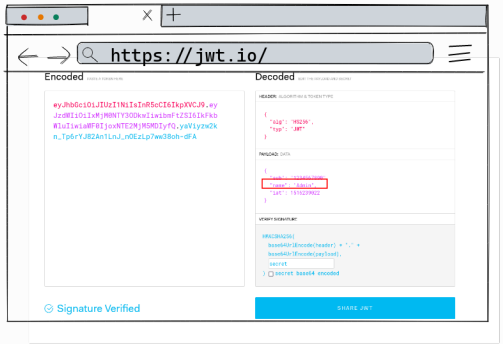
1 Identify encryption algorithm




2 We try to crack




2 Create new token



# Vulnerabilities

## $ Broken User Authentication | How to



- **Black box** vs **Grey Box** vs **White box**

## Tools for testing Broken User Authentication

1. Burp Suite's Intruder
2. Burp Suite's Secuencer
3. Burp Suite's Decoder
4. Burp Suite's JSON Web Tokens plugin
5. JWT_Tool.py
6. jwt.io
7. Ffuf
8. Postman

Broken User Authentication playground time!

# Clean Up

# Clean Up
## $ Resources



- Learn more about Hacking APIs:

- InsiderPhD's channel
- Hacking APIs book
- Payloads All The Things
- Awesome API security

# Clean Up

## $ Practice environments



- Test your learned skills against these practice environments:

- crAPI
- DVGA
- Generic University
- VAPI

Thank you! Questions?
EOF