Application Security Project
Cybernetic

Group: Joel (Leader), William, Dylan, Kent

Introduction

Cybernetic

- Web APIs used for e-commerce web application
- It is first implemented as insecure version and further improved as secure version

Insecure Version	Secure Version
Insecure with OWASP API Security Top 10 Vulnerability	Secure with Security Fixes Implemented
No input validation	(The Top 10 Proactive Control) C5: All input validation
Not All Errors and Exceptions Handled	C10: Handle All Errors and Exceptions
Security Frameworks and Libraries partially Leveraged.	C2: Leverage Security Frameworks and Libraries
	C3: Secure Database Access

Platform Used

Development Framework

Flask

Flask-RESTX

Programming Language

Python

Database Used

MySQL

Security Testing tool

Bandit (SAST)

OWASP ZAP (DAST)

Cloud Service Provider

Amazon web service (AWS)

DevSecOps

Github

SIEM

AlienVault OSSIM

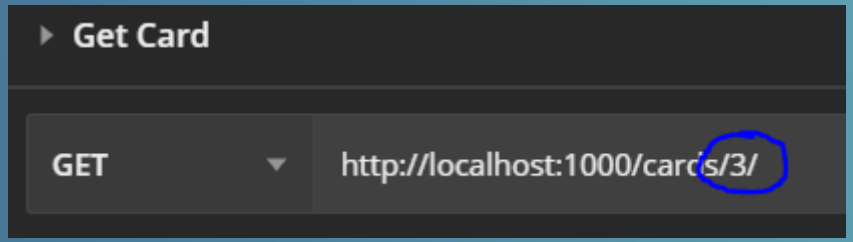
API 1

Broken Object level Authorization

By: Joel

Vulnerability

APIs which are vulnerable to this can be exploited by manipulating the ID of an object that is sent within the request which may lead to unauthorized access to sensitive data.



Scenario :
The attacker can change the id to access other users object

Fixes

JWT Token Authorization with Object's Foreign Key :
User ID references User(User_ID)

API 2

Broken User Authentication

By: Joel

Vulnerability

This vulnerability is when authentication is not implemented properly which can be used by an attacker to compromise authentication tokens or assume the identity of another user.

```
{
    "rating": 5,
    "comments": "Very Good",
    "user_id": 1
}
```

Scenario :
The attacker can impersonate another user by changing user ID

Fixes

JWT Token Authentication

Email Verification

Optional Multi Factor Authentication

API 3

Excessive Data Exposure

By : Dylan

Vulnerability

Exploitation of Excessive Data Exposure is performed by sniffing the traffic to analyze the API responses, looking for sensitive data that should not be returned to the user.


```
{
    "success": true,
    "data": {
        "reviews": [
            {
                "comments": "nil",
                "user": {
                    "admin": false,
                    "active": true,
                    "id": 2,
                    "email": "dylanliew0503@gmail.com",
                    "username": "forgetful"
                },
                "id": 137,
                "deleted": false,
                "rating": 1
            },
            {
                "comments": "nil",
                "user": {
                    "admin": false,
                    "active": true,
                    "id": 2,
                    "email": "dylanliew0503@gmail.com",
                    "username": "forgetful"
                }
```


Scenario :
The attackers sniff the traffic of the product reviews response which return user sensitive information such as email

Fixes

Filter Data on Server Site

Censor parts of sensitive information
E.G. Phone Number : ******34

AWS Symmetric Key Encryption on database stored data at rest with AWS KMS for key management

API 5

Broken Function Level Authorization

By : Dylan

Vulnerability

Attackers can exploit APIs by sending legitimate API calls to API endpoints which the attacker should not have access to such as View All Users.

```
{
    "message": "Unauthorised Access",
    "success": false
}
```

Scenario :
The attacker sends a GET request to retrieve all users without authentication/user function level authorization which lead to expose of sensitive information of users

Fixes

Use of JWT Token to authenticate user identity which then validate for user function level

API 6

Mass Assignment

By: Kent

Vulnerability

This vulnerability occurs when multiple objects/entities of the same class have a common attribute that is assigned across these objects.

```
{
    "username": "New Tester",
    "email": "new_tester@test.com",
    "admin":true
}
```

Scenario :
The user edit the request such that he/she can modify admin attribute of himself/herself

Fixes

Whitelist attributes that can edited

Blacklist attributes that should not be edited

Sanitize user input

API 8

Injection

By : Kent

Vulnerability

This vulnerability occurs when an attacker supplied malicious input to the API which is processed by the interpreter, resulting in a change of the execution of the API.

```
{
    "email": "test@test.com ' or 1=1 --",
    "password": "not_correct_password"
}
```

Scenario :
The attacker enter input that contains SQL query into the username/password field to bypass login authentication

Fixes

Use secure query functions

Sanitize user inputs

API 4

Lack of Resources & Rate Limiting

By: William

Vulnerability

The API is not protected against an excessive amount of API calls or payload sizes which can be use for Denial of Service (DoS) and authentication flaws such as brute force attacks.

```
password: ashley

Attempt 9
HTTP status: (401)
email: dylanliew0503@gmail.com
password: abc123

Attempt 124
HTTP status: (429)
email: dylanliew0503@gmail.com
password: richard
Connection Throttled
Attempt 125
HTTP status: (429)
email: dylanliew0503@gmail.com
password: 112233
Connection Throttled
```

Scenario :
The attacker can brute force login system

Fixes

Pagination

Amazon API Gateway

API 10

Insufficient Logging & Monitoring

By: William

Vulnerability

Insufficient logging and monitoring allows attacker to abuse systems without being noticed.

Scenario :
Due to insufficient logging, the company is not able to assess what data was accessed by malicious actors.

Fixes

AlienVault OSSIM

API 7

Security Misconfiguration

By: william

Vulnerability

Misconfigured systems allow attackers to gain unauthorized access or knowledge of the system.

Fixes

Usage of AWS Secret Manager of manager secrets

Secure Random Generator

X-Frame-Options header set to "Sameorigin" to prevent clickjacking attacks

X-Content-Type-Options set to "nosniff" to protect against MIME sniffing vulnerabilities

Use of Secure Hash Functions

Testing

SAST & DAST

SAST

- Bandit
  - Use of GitHub Actions to perform Bandit Testing

Insecure Version	Secure Version
Use of insecure MD2, MD4, MD5, or SHA1 hash function	Solved by using Bcrypt
Standard pseudo-random generators are not suitable for security/cryptographic purposes.	Solved by using Cryptographically Secure Pseudo-Random Number Generator
Possible Hardcoded Password	Solved by using AWS Secret Manager
Possible SQL injection vector through string-based query construction.	Solved by using SQLAlchemy Functions

DAST

- OWASP ZAP
  - Use of Postman JSON File which is then converted to OpenApi 3.0 (Json)
  - For OWASP Zap to do automatic scan

Insecure Version	Secure Version
SQL Injection	Solved by using SQLAlchemy Functions
Buffer Overflow	Solved by turning off the debugger (Not Considered as a vulnerability)
X-Frame-Options Header Not Set	X-Frame-Options header set to "Sameorigin" to prevent clickjacking attacks
X-Content-Type-Options Header Missing	X-Content-Type-Options set to "nosniff" to protect against MIME sniffing vulnerabilities
Absence of Anti-CSRF Tokens	Solved by turning off the debugger (Not Considered as a vulnerability)
Cross Site Scripting	Solved by using Marshmallow to ensure it's treated as a string instead of script

DAST

Insecure Version	Secure Version
Application Error Disclosure	Solved by turning off the debugger
Information Disclosure - Sensitive Information in URL	Not Considered as a vulnerability
Timestamp Disclosure - Unix	Remove timestamp
Information Disclosure - Suspicious Comments	Solved by turning off the debugger

THANKS