WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal Homepage: www.ijrpr.com ISSN 2582-7421

AuthSentry: An Automated Authorization Testing Tool for API

Aditya Babar, Yash Lohakare, Kshitija Rahangdale, Manish Waghmare,Sharayu Patil

Department of Computer Engineering, Vishwaniketan's Institute of Management Entrepreneurship & Engineering Technology (ViMEET), University of Mumbai

ABSTRACT—

Broken Object Level Authorization (BOLA) is ranked first in the OWASP API Security Top 10. It occurs when an API confirms that a user is logged in but never checks whether the requested object belongs to that user, so changing an identifier in a request can expose another person's data. Because the missing check depends on each application's own logic, signature-based scanners often miss it, manual penetration testing is slow and expensive, and many existing tools need security expertise or source code access. This paper presents AuthSentry, an automated black-box tool that tests a live REST API for BOLA from the outside. Given an API address, a list of endpoints and sample authenticated requests, the tool normalizes the input, identifies object identifiers, sends targeted probe requests that replay requests with another user's token, analyzes the responses against a stored baseline using rule-based detection, and reports each verdict with a plain-language reason. The system uses a Python and Flask backend for probing and analysis, a React web interface and Firebase for data storage. The web interface and the core verdict logic have been implemented and demonstrated on local test servers; measured detection rate, false-positive rate and scan time on named vulnerable APIs are planned and are not claimed in this paper.

Index Terms—Broken Object Level Authorization, API Security, Black-Box Testing, Authorization Testing, REST API, OWASP

I. INTRODUCTION

Modern applications depend on APIs to exchange data between users, servers and services, and the security of these APIs now decides how safe the data behind an application is. When a user opens a profile in a mobile app, the app asks the API for the data of one particular user, usually identified by an ID in the URL. A secure API must check two things: authentication, which confirms who is asking, and authorization, which confirms whether that person may see the requested data.

BOLA arises when only the first check is performed. For example, a request such as /api/users/123/profile can be changed to /api/users/999/profile and, on a vulnerable API, returns the private profile of user 999. OWASP lists this weakness first among API risks [1], and MITRE classifies it as CWE-639, Authorization Bypass Through User-Controlled Key [3]. Finding it today usually means manual penetration testing, which needs expertise and time, or commercial platforms that startups, students and small teams cannot afford.

This paper proposes AuthSentry, an automated black-box tool that closes this gap. Black-box testing means the tool examines an API from the outside, like a real attacker, without source code. The tool is built to be usable by developers without a security background and to give each result a reason they can act on.

Main Contributions of this Work

1. A problem analysis of logic-dependent authorization flaws and the gaps in current testing practice for small teams.
2. An integrated black-box approach combining input normalization, object-identifier detection, probing, response analysis, scoring and reporting.
3. A rule-based detection method that classifies responses as vulnerable, secure or inconclusive using a baseline and an owner control request, and avoids unvalidated models.
4. A layered, modular architecture (React, Flask, Firebase) with a safe-scanning and ethical-use design, and an implemented web interface demonstrated on local test servers.

II. PROBLEM STATEMENT AND OBJECTIVES

A. Problem Statement

BOLA depends on each application's own data model and business rules, so it cannot be recognised from the shape of a request alone. Standard scanners often miss it, manual testing is slow and hard to repeat after every change, commercial tools are priced for large organizations, and free tools need manual

setup, several test accounts and security knowledge. They also rarely indicate how reliable a finding is. As a result, startups, students and small teams often release APIs that were never tested for BOLA. The problem addressed here is the lack of a simple, automated and affordable way to test a live REST API for BOLA without source code, producing findings that a developer without a security background can understand and act on.

B. Objectives

- To design an automated black-box tool that detects BOLA in REST APIs without source code.
- To generate targeted probes by changing object identifiers, swapping authentication tokens and altering headers.
- To analyze responses and identify cases where authorization checks are missing or inadequate.
- To attach a confidence score and a severity rating to findings so that real issues can be told apart from false alarms.
- To produce clear reports with evidence and remediation steps.
- To provide a simple web interface for starting scans and viewing results.
- To evaluate the tool on intentionally vulnerable APIs and measure detection rate, false-positive rate and scan time.

III. RELATED WORK

Fuzzing and mutation-based testing are established ways to find vulnerabilities by sending modified inputs to a program and observing its behavior [4]. RESTler applies this idea to cloud services by generating stateful sequences of REST requests from an API specification [5], showing that black-box testing of REST APIs is practical. These works look for crashes and generic errors; AuthSentry instead mutates only the identity-related parts of a request, namely object identifiers, tokens and headers, and checks whether the responses show a missing authorization check.

The OWASP API Security Top 10 [1], the OWASP Web Security Testing Guide [2] and CWE-639 [3] define what must be tested and how authorization can be checked manually. AuthSentry aims to automate how it is tested.

In practice, BOLA is found through separate and mostly manual activities. Penetration testers examine each endpoint by hand. Developers use proxy tools such as Burp Suite (often with extensions such as Autorize) or OWASP ZAP, or build Postman collections to repeat requests. Static analysis (SAST) tools scan source code but need access to it and cannot confirm that a flaw is exploitable on a running API. Commercial API security platforms offer automated testing at enterprise pricing. Each covers part of the problem, but none gives a small team a simple, self-serve way to test a live API for BOLA with a reliability indication for each result.

The proposed system differs by being dedicated to BOLA, requiring no source code, normalizing input automatically, comparing every probe with a stored baseline, and presenting each verdict with a plain-language reason.

IV. SYSTEM ARCHITECTURE

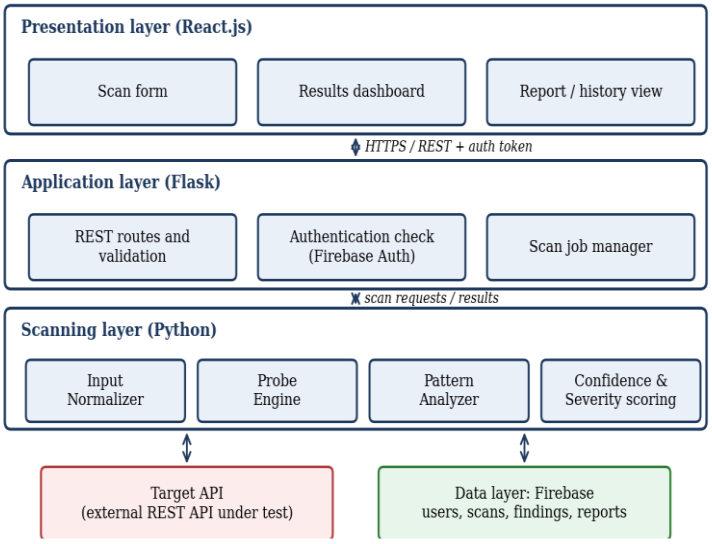


Fig. 1. Layered architecture of AuthSentry.

AuthSentry is a layered, modular system. The presentation layer is a React web application where users start scans and view results. The application layer is a Flask server that exposes the REST interface, validates requests and verifies the user's identity using Firebase Authentication. The scanning layer

contains the Input Normalizer, Probe Engine, Pattern Analyzer and scoring modules, which send requests to the target API and analyze the responses. The data layer is Firebase, which stores users, scans, findings and reports. Keeping the scanning logic separate from the web interface lets each part be tested and changed independently.

A. System Modules

Module	**Purpose**	**Output**
Input Normalizer	Parses endpoints and sample requests; identifies object IDs	Structured test cases
Probe Engine	Generates and sends baseline and probe requests safely	Stored requests and responses
Pattern Analyzer	Compares responses with the baseline; applies detection rules	Flagged cases with signals
Scoring	Computes confidence and severity	Scored findings
Report Generator	Builds report with evidence and fixes	Web view, PDF or JSON
Web Dashboard	Scan form, results and history	User interface

TABLE I — System modules

B. Technologies

The frontend uses React.js. The backend and scanning logic use Python with Flask, with HTTP client libraries such as requests or httpx and pytest for testing. Firebase (Authentication and Cloud Firestore) stores users, scans, findings and reports as documents, which suits scan data because each scan holds a different number of endpoints, probes and findings. Status codes, headers and bearer tokens are interpreted according to RFC 7231, RFC 6750 and RFC 6749 [6]–[8], and severity may be mapped using CVSS v3.1 [9]. No machine-learning models are used.

V. METHODOLOGY

The detection workflow transforms a small amount of input into a BOLA report through seven stages (Fig. 2).

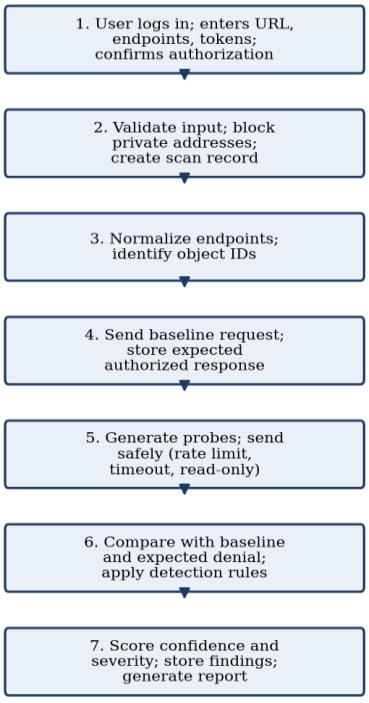


Fig. 2. Scan workflow of AuthSentry.

A. Input and Object Identifier Detection

The user supplies the target base URL, two authentication tokens (User A, the owner of the object, and User B, the attacker), and the endpoints to test, each with a path pattern such as /api/users/{id}, an HTTP method and User A's object ID. User B's own object ID is optional and allows both directions to be tested. The Input Normalizer examines each endpoint to find parts that identify an object: numeric IDs, UUIDs or slugs in the URL path, query string or request body. Each identifier becomes a point at which a probe can be applied.

B. Baseline and Probe Generation

The sample request is first sent as given, and its response (status code, structure, size and key fields) is stored as the expected authorized result. The Probe Engine then builds modified copies of this request, changing only the part being tested so that any difference in the response can be traced to that change. Table II lists the probe types.

Probe	**What changes**	**Expected if secure**
Changed object ID	ID replaced by a neighbouring or other known ID	Denied (401, 403 or 404)
Swapped token	Another user's token with the same object ID	Denied (401, 403 or 404)
Removed / invalid token	Token omitted or altered	Denied (401)
Altered headers	Headers changed or removed	Same denial as baseline

TABLE II — Probe types

C. Safe Execution

Probes are sent with rate limiting and timeouts. Only read operations (GET) are used by default, and state-changing methods run only if the user enables them. The user must confirm authorization before each scan, requests to private or internal addresses are blocked, and tokens and personal data are masked in saved evidence.

D. Detection Rules and Response Classification

The Pattern Analyzer compares each probe response with the stored baseline and with the expected denial, and applies the following starting rule set:

5. A request for another user's object that returns success with real data is a strong BOLA signal.
6. A response with the baseline's structure but different object data suggests access to another object.
7. A missing or invalid token that still returns protected data indicates a possible authentication weakness; by itself it does not confirm BOLA, which needs evidence that one authenticated user can reach another user's object.
8. Different error messages for existing and missing objects reveal information even when access is denied (a weaker signal).
9. A success status with an empty or generic body is ambiguous, not confirmed.

Observation	**Class**	**Treatment**
Other user's object returns success with real data	Possible BOLA (strong)	Flagged
Other user's object returns 401, 403 or 404	Denied (expected)	Treated as secure
Baseline structure, different object data	Access to another object	Flagged
Success with empty or generic body	Ambiguous	Reported, not confirmed
Owner's own control request fails (e.g. 404)	Inconclusive	No conclusion drawn
Different error messages for existing / missing objects	Information leak (weak)	Low weight

TABLE III — Response classes

The owner control request is important: a probe can only be judged if the owner's own request to the same object succeeds. If the control fails, the result is reported as inconclusive rather than being counted as secure or vulnerable, which avoids a false sense of safety.

E. Confidence and Severity

The confidence score is intended to express how strongly the evidence supports a finding. Conceptually it may draw on the number of independent signals that agree, whether the response body differs from the baseline, consistency across repeated probes, and whether the response looks like a genuine denial

or an ambiguous empty result. Severity may be mapped using CVSS v3.1 according to the type of data exposed and whether the operation is read or write [9]. No formula or weights are claimed here: they must be tested on vulnerable APIs, and false-positive and false-negative rates reported, before the score is presented as meaningful.

F. Worked Example

Consider the example of Table IV with User A (ID 123) as owner and User B as attacker. The baseline GET /api/users/123 with User A's token returns 200 with User A's data. The same request with User B's token is then sent. A 403 means the probe is secure; a 200 with User A's data is flagged as possible BOLA, and a finding is stored with its masked request and response and a suggested fix.

Step	Request	Result
Baseline	GET /api/users/123, User A token	200, User A's data stored
Probe (secure)	GET /api/users/123, User B token	403, denied
Probe (vulnerable)	GET /api/users/123, User B token	200, User A's data: flagged

TABLE IV — Worked example (illustrative)

VI. IMPLEMENTATION, RESULTS AND EVALUATION

Measured evaluation results, such as detection rate, false-positive rate and scan time on named vulnerable APIs, were not available when this paper was prepared, so no such figures are claimed. This section presents the implemented interface, the verdicts it produced on local test servers, and the planned evaluation.

A. Implemented Interface

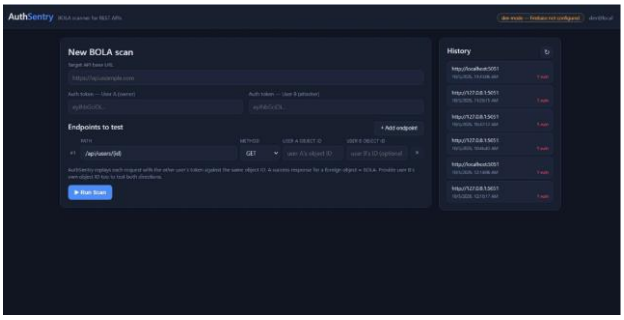


Fig. 3. New BOLA scan form and scan History panel.

Fig. 3 shows the scan form, where the user enters the target URL, both tokens and the endpoints to test, and the History panel listing earlier scans with their target, time and vulnerability count. AuthSentry replays each request with the other user's token against the same object ID; a success response for a foreign object indicates BOLA.

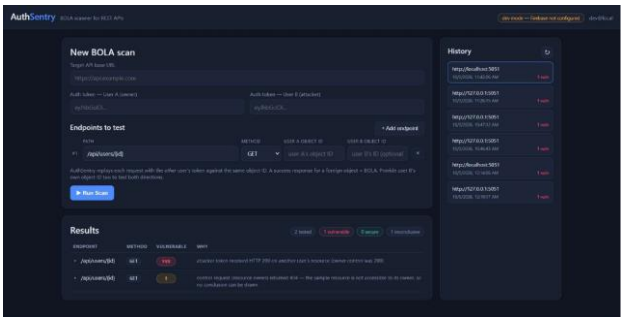


Fig. 4. Scan form with the Results panel after a scan.

Fig. 4 shows the Results panel for the endpoint /api/users/{id} with GET. The summary reports 2 tested, 1 vulnerable, 0 secure and 1 inconclusive. The first row is marked vulnerable because the attacker's token received HTTP 200 on another user's resource while the owner control request also returned 200. The second row is marked inconclusive because the owner's control request returned 404, so the sample resource was not accessible to its owner and no conclusion could be drawn. The screens were captured in development mode (Firebase not configured) against local test servers on localhost.

B. Test Cases

Test cases follow the detection rules. Those marked Observed correspond to verdicts visible in the screenshots; the others are planned.

ID	Scenario / expected outcome	Verdict	Status
TC1	Owner token, own ID: baseline stored	Baseline	Planned
TC2	Attacker token, owner's ID returns 200 with data	Vulnerable	Observed
TC3	Attacker token returns 401, 403 or 404	Secure	Planned
TC4	Owner control request returns 404	Inconclusive	Observed
TC5	Reverse direction: owner token on attacker's ID	Same rules	Planned
TC6	Missing/invalid token still returns data	Auth weakness	Planned
TC7	Success with empty or generic body	Ambiguous	Planned
TC8	Different errors for existing/missing objects	Weak signal	Planned
TC9	Private or internal address as target	Blocked	Planned
TC10	Authorization not confirmed	Not started	Planned

TABLE V — Test cases

C. Comparison with Existing Approaches

Table VI compares AuthSentry with existing approaches qualitatively, as described in Section III. It is a comparison of design properties, not of measured performance.

Approach	Source code needed	Manual effort	Confidence score	BOLA-specific
Manual pen-testing	No	High	No	Tester-dependent
Proxy tools (Burp, ZAP, Postman)	No	High	No	No
SAST tools	Yes	Low	No	No
Commercial API platforms	No	Low	Varies	Varies
AuthSentry (proposed)	No	Low	Designed (unvalidated)	Yes

TABLE VI — Qualitative comparison

D. Evaluation Plan

The final objective is to run AuthSentry against intentionally vulnerable APIs whose BOLA flaws are known in advance, compare findings with the known flaws and record the metrics in Table VII. Values will be reported only after these runs. The environment will use local vulnerable test servers so that no third-party system is scanned without permission.

Metric	Meaning
Detection rate	Share of known BOLA flaws that the tool reports
False-positive rate	Share of reported findings that are not real flaws
False negatives	Known flaws that the tool misses
Scan time	Time taken to scan the endpoints
Confidence calibration	Whether higher confidence matches real flaws more often

TABLE VII — Planned evaluation metrics

VII. DISCUSSION AND LIMITATIONS

AuthSentry's main design choice is to report what it can justify. Inconclusive and ambiguous results are kept separate from vulnerable and secure ones, and every verdict carries a plain-language reason, so a developer can see why the tool reached it. Running only read-only probes by default, with rate limiting, authorization confirmation and internal-address blocking, reduces the risk of harming the target or misusing the tool.

Several limitations apply. The tool covers only BOLA in REST APIs and not other OWASP API risks, GraphQL or WebSocket APIs. Detection depends on response comparison, so it can produce false positives (for example, public data returned for any ID) and false negatives (for example, flaws hidden behind complex business logic, multi-step workflows, MFA or CAPTCHA). Effectiveness and generalizability have not yet been established because no measured evaluation on named vulnerable APIs is available, and the confidence score and detection rules are heuristic. Scans can be affected by rate limits, API changes and inconsistent error handling in the target. Findings with medium confidence need manual verification, and the tool must be used only on APIs the user owns or is authorized to test; unauthorized scanning may be illegal under the Information Technology Act, 2000. A small survey of developers, whose findings are indicative only, suggested that most had only heard of BOLA, relied on manual testing, found tools and audits expensive, and were willing to use a free automated tool; it does not support strong claims about developers' views.

VIII. CONCLUSION AND FUTURE WORK

A. Conclusion

This paper presented AuthSentry, an automated black-box tool for detecting BOLA in REST APIs. It takes an API address, endpoints and sample authenticated requests, sends targeted probes, analyzes responses against a baseline with rule-based detection, and reports verdicts with reasons through a React web interface backed by Python, Flask and Firebase. The intended value is a repeatable, affordable and understandable way for startups, students and small teams to test APIs for BOLA without source code. The paper does not claim that the tool finds every BOLA flaw, replaces professional penetration testing, or outperforms existing tools; these questions depend on the evaluation described in Section VI-D.

B. Future Work

10. Complete the planned evaluation on intentionally vulnerable APIs and calibrate the confidence score.
11. Multi-user comparison testing with two or more test accounts.
12. OpenAPI and Postman import to fill in endpoints and parameters automatically.
13. Broken Function Level Authorization (BFLA) detection.
14. Support for GraphQL and WebSocket APIs.
15. CI/CD integration through a command-line mode and a GitHub Actions workflow that fails a build on high-severity findings.
16. Machine-learning-based response similarity to reduce false positives, subject to evidence that it improves results.
17. More authentication schemes (OAuth 2.0 flows with token refresh, API keys) and report export with trend tracking in PDF or JSON.

ACKNOWLEDGMENT

The authors express their sincere gratitude to their project guide, Prof. Sharayu Patil, for continuous guidance and feedback, to Prof. Charusheela Pandit (Head of Department) and Dr. S. B. Kadam (Principal) for their support, and to the faculty of the Department of Computer Engineering, ViMEET, for the academic environment and resources provided during this work.

REFERENCES

[1] OWASP Foundation, "OWASP API Security Top 10 – 2023." [Online]. Available: https://owasp.org/API-Security/editions/2023/en/0x11-t10/

[2] OWASP Foundation, "Web Security Testing Guide (WSTG)." [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/

[3] MITRE, "CWE-639: Authorization Bypass Through User-Controlled Key." [Online]. Available: https://cwe.mitre.org/data/definitions/639.html

[4] V. J. M. Manès et al., "The Art, Science, and Engineering of Fuzzing: A Survey," IEEE Trans. Softw. Eng., 2019. [Online]. Available: https://arxiv.org/abs/1812.00140

[5] V. Atlidakis, P. Godefroid and M. Polishchuk, "RESTler: Stateful REST API Fuzzing," in Proc. 41st Int. Conf. Software Engineering (ICSE), 2019.

[6] R. Fielding and J. Reschke (Eds.), "RFC 7231: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content," IETF, 2014.

[7] M. Jones and D. Hardt, "RFC 6750: The OAuth 2.0 Authorization Framework: Bearer Token Usage," IETF, 2012.

[8] D. Hardt (Ed.), "RFC 6749: The OAuth 2.0 Authorization Framework," IETF, 2012.

[9] FIRST, "Common Vulnerability Scoring System v3.1: Specification Document," Forum of Incident Response and Security Teams.