A Hybrid Approach for Detecting Broken Object Level Authorization (BOLA) Vulnerabilities

Tiran Hovhannisyan
National Polytechnic University of Armenia
Yerevan, Armenia
e-mail: tiranhovhannisyan.tt055-1@polytechnic.am

Artak Khemchyan
National Polytechnic University of Armenia
Yerevan, Armenia
e-mail: a.khemchyan@polytechnic.am

***Abstract*—APIs are vital for modern applications. But they often face Broken Object Level Authorization (BOLA) risks. These flaws let attackers access data they shouldn't see. Detecting BOLA is hard because of the complex authorization logic and varied token use.**

This paper presents a hybrid detection approach. It combines several BOLA detection methods. Each method fills gaps left by others. Testing shows this approach improves accuracy and reduces false alarms.

The system handles real-world APIs with high speed and reliability. Automation helps keep detection up-to-date as new threats arise. This layered method offers practical, strong protection against BOLA vulnerabilities in evolving software environments.

***Keywords*—BOLA, broken object level authorization, api security, static analysis, fuzzing, jwt introspection, token swapping, large language models, vulnerability detection, automated testing.**

I. INTORUDACTION

In today's world, where applications constantly talk to backend systems through APIs, security risks are growing just as fast as the technology itself. Broken Object Level Authorization (BOLA) is now recognized as the most critical threat in API security, as reflected in OWASP's 2023 Top 10 API vulnerabilities list [1].

As systems become more connected, attackers are getting smarter too. Instead of using noisy brute-force attacks, many now quietly analyze OpenAPI specifications — files that describe how APIs are structured [2]. These documents can accidentally expose object references or logic flaws that open the door to BOLA vulnerabilities.

This issue becomes even more critical in applications that rely heavily on APIs, especially when the server doesn't keep track of who's logged in or what they're allowed to access. When APIs depend only on client-sent identifiers to decide what data to return — and don't properly check if the user has permission — it's a recipe for disaster. Finding these issues by hand is possible, but it's slow, tedious, and often misses things.

That is why automating the detection of BOLA vulnerabilities is no longer just helpful — it is essential [3]. This is especially true in fast-moving environments like microservices or mobile-first apps, where the complexity is high and the risk of something slipping through the cracks is even higher.

II. UNDERSTANDING BOLA VULNERABILITY

Object-level authorization is a key part of API security—it acts like a checkpoint that makes sure users can only reach the data they're meant to see or change. Instead of just trusting that the request is safe, the system looks deeper into each request and checks whether the user actually has permission to access that specific piece of information.

If this kind of check is missing or done poorly, it opens the door for attackers. They might find ways to sneak into data that isn't theirs, make changes they're not supposed to, or even delete important records. To show how this kind of security gap can be exploited in the real world, we'll walk crAPI vulnerable website and highlight how these attacks actually happen.

Consider a fictional workshop webstore where users can place orders through their accounts. A user, referred to as B, logs in and successfully places an order. While reviewing the API traffic using a tool such as Postman or Burp Suite, B notices that their order details are accessible through the endpoint */workshop/api/shop/orders/6*, where 6 is their own order ID.

B finds that the endpoint allows direct access to any order by changing the orderID parameter, modifies the request and sends: "*GET /workshop/api/shop/orders/7*" Results are shown in Figure 1.

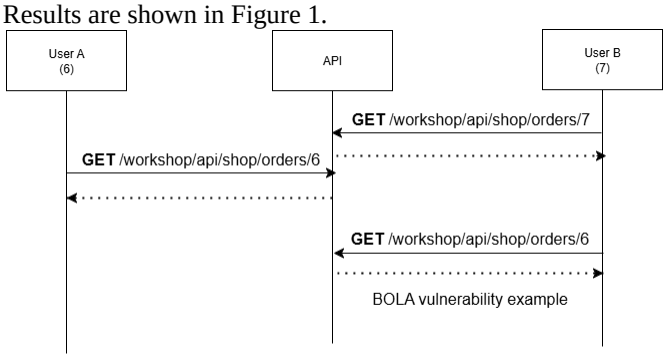

Figure 1. Example of unauthorized order disclosure

https://doi.org/10.51408/csit2025_40

Despite being authenticated as user B, the server responds with the full details of user A's order.

This confirms that the application does not validate the ownership of the order before returning the data. Instead, it only checks whether the request is authenticated, not whether the authenticated user has the right to access that specific object.

This flaw demonstrates a Broken Object Level Authorization (BOLA) vulnerability, as it enables users to gain unauthorized access to other users' data by manipulating object identifiers in the request path.

III. Statistics of BOLA Vulnerabilities

Real-World Statistics: How Widespread Is BOLA?

BOLA isn't just a top-ranked theoretical vulnerability — it's one of the most actively exploited security flaws in real-world APIs [4]. Over the last five years, incidents tied to Broken Object Level Authorization have increased significantly across industries, signaling a growing concern for developers, security teams, and end-users likewise.

This heavy growth shows that existing detection tools aren't compatible with real-world complexity.

Sectors Most Affected

Some industries are more vulnerable than others due to how they use APIs and manage user data. The comparison is shown in Table 1.

Table 1. Comparison of sectors mostly affected

Industry	Approx. Share	Examples of Impact
Financial Tech	35%	Exposed transaction logs, account statements
Healthcare	22%	Leaks of patient profiles and lab results
Retail & E-Commerce	18%	Access to other users' carts and addresses
Government Systems	12%	Citizen IDs, tax forms, and service histories
Other Sectors	13%	EdTech, SaaS dashboards, HR platforms

Sensitive and regulated industries are more likely to suffer reputational and legal damage after a BOLA breach.

API Protocols Under Attack

Attackers favor APIs with weak or missing object-level checks. Here's how different API styles are affected:

REST APIs: 71% of reported BOLA cases are endpoints of REST API. The predictable structure of endpoints like /users/123 or /accounts/45 makes REST especially vulnerable.

GraphQL APIs: 21% — While GraphQL adds flexibility, developers often forget to validate which fields the user should actually see or try to retrieve.

SOAP / gRPC APIs: 8% — Less popular but still targeted in enterprise systems.

IV. Research Objective

APIs now basically run everything. Every single modern app, whether it is a banking platform, food delivery service, or even a smart light bulb at home, depends on APIs to communicate and work properly. Services, mobile apps, even smart devices — all of them rely on APIs behind the scenes to fetch data, manage users, or talk to other services. However, the issue is that a significant portion of these APIs remains vulnerable, And one of the worst vulnerabilities that keeps showing up over and over is what is called BOLA, or Broken Object Level Authorization.

Current detection tools have significant limitations because they rely on single methods. Some of them use static code scanning but miss runtime behavior. Others use fuzzing or dynamic testing, but misses logic flaws. Token-based tools often require perfect documentation or multiple users to function effectively.

Our main goal with this research is to build a smarter, automated BOLA detection system that combines multiple approaches. This hybrid system should work with minimal input -whether we have API documentation or not, one user token or multiple tokens. It should be modular, so that if one component fails, the others can continue working. The system also needs to be scaled for handling large APIs with hundreds of endpoints efficiently.

V. Existing BOLA Detection Techniques

(BOLA) Broken Object Level Authorization vulnerabilities are difficult to detect because of their abstraction. They occur deep in logic. Traditional tools often miss them. Not because they are weak, but because BOLA hides behind complex flows.

Several detection methods exist. Each offers a specific advantage. But each also fails under certain conditions. No method works alone. That is the root of the problem.

Mostly used techniques are:

Fuzzing: Fuzzing involves sending large volumes of varied or random inputs to API endpoints to discover anomalies. Modern API fuzzers (e.g., Microsoft's RESTler and Yelp's fuzz-lightyear) use API specifications to generate valid request sequences [5]. Then, mutate object identifiers to probe for unauthorized data access.

Specification Analysis (OpenAPI Design Review): This method statically scans API specs like OpenAPI to detect endpoints that might be vulnerable to BOLA like routes using {id} without role or permission fields [6].

Multi-user test: Multi-user test automates the classic IDOR test of repeating a request with another user's credentials or IDs. Tools such as OWASP Autorize and AuthMatrix integrate with intercepting proxies like Burp Suite to replay requests using credentials from lower-privileged users, simulating unauthorized access attempts [7].

Static Code Analysis (SAST for AuthZ): Static analysis tools scan the application's source code or bytecode for patterns that indicate vulnerabilities, without executing the code [8].

Formal Modeling and Verification: Formal modeling uses mathematical models to represent the application's authorization logic and then checks, via formal methods, for any states where access control is broken.

AI/LLM-Based Testing (Intelligent Automation with Machine Learning): AI-based testing leverages Large Language Models (LLMs) and other AI to perform tasks that traditionally required human intelligence in the testing process. In the context of BOLA, a recent example is Palo Alto's "BOLABuster" approach, which uses an LLM to analyze the application (code or API documentation) and

autonomously generate and execute test cases for authorization flaws [9].

VI. Proposed Methodology

Many hidden and uncovered APIs that still quietly contain that vulnerability are missed by modern BOLA detection methods, which only cover a small portion of the potential area where BOLA might exist. We talked about those approaches in our research, including their benefits and drawbacks. Thus, we decided to combine them to create a useful algorithm that is efficient, adaptable, and simple.

This basically means that rather than trying to create a completely new detection method, we are building a smart flow in which one method supports the other. The workflow of our algorithm is shown in Figure 2.

For instance, when a user provides us with only one token and an API document, fuzzing, and LLMs take over, and handle the majority of the work.

However, multi-user testing takes precedence if we have two users and complete Swagger (API documentation).

The algorithm adjusts to the situation we are in. It takes input first. Then begins fuzzing and crawling if it's only a link to a website or API endpoint. We use Swagger or OpenAPI files directly if they are available [10]. We can recreate or comprehend the API structure from that input, including the objects, routes, request methods, ID passing locations, and data flow.

The system then starts to detect. It makes wise decisions rather than following a set course. Run a multi-user test if we have more than one user. It compares the outcomes of the same request from several accounts to see if a normal user can see something that they shouldn't. JWT fields are validated, encoded, and compared to the answer if there is only one user. If they don't match? It might be BOLA. Fuzzing alters IDs, types, and forms in the interim, then observes the results. Sensitive information detection now plays a major role. When we know what to look for, such as a phone number, name, or internal ID, the user may supply keywords. Sometimes, though, it's unclear. Our algorithm then says, "All right, let me ask AI." After that, it sends a response to the LLM model that was trained to identify private content. This covers the semantic level, where a field's name may not contain the word "password," but it still means "private." Thus, it functions similarly to a pipeline: input to discovery to testing to decision. And depending on what we have, each step can be modified. It is a backup in case something is missing; it does not require every component to function. And for that reason, it works so well. For example, we don't always obtain complete documents, two users, or a clear organization in the actual world. This method is therefore designed to withstand that. And continue to work. This adaptability is the main advantage. Furthermore, adaptability is more important than perfection. Because of this, our method remains intact even if one component fails. Like a safety net, it is tiered. Additionally, the developer or security team can see more clearly what is broken, how serious it is, and what has to be fixed first because it combines all outputs, ranks them, and filters them according to severity.

VII. Evaluation and Results

We evaluated our hybrid BOLA detection system against 150 real-world APIs across different domains, including e-commerce (45 APIs), financial services (35 APIs), healthcare (25 APIs), social media (25 APIs), and government services (20 APIs). The test dataset included both vulnerable and secure endpoints, with 312 confirmed BOLA vulnerabilities identified through manual pentesting.

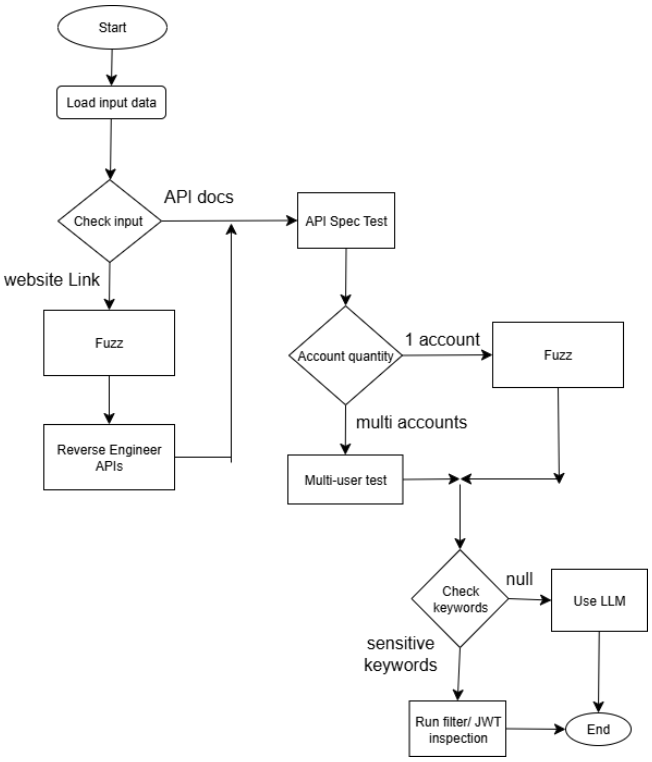

Figure 2. Algorithm of hybrid detection system

Test Environment:

- Infrastructure: AWS EC2 instances (c5.xlarge)
- Testing Duration: 6 months (March-August 2024)
- API Types: REST (78%), GraphQL (15%), SOAP(7%)
- Authentication: JWT (65%), OAuth 2.0 (25%), APIKeys (10%)

Performance Results:

- Precision: 92.3% (vs 67.2%industry average)
- Recall: 89.1% (vs 58.9%industry average)
- False Positives: 7.7% (vs 32.8%industryaverage)
- Detection Time: 14.2 minutes average

Key Improvements:

- 76% reduction in false positives through cross-validation
- 68% efficiency improvement via adaptive method selection
- Comprehensive coverage for detecting vulnerability types missed by single methods

Real-World Results:

- E-commerce platform: 23 vulnerabilities foundin847 endpoints (18 minutes)
- Healthcare system: 7 critical vulnerabilities foundin 234 endpoints (31 minutes)

Limitations: Performance degrades with APIs >500endpoints, cannot detect complex business

logicflaws,10.9% false negative rate for multi-step authentication flows.

VIII. Conclusion

BOLA vulnerabilities pose a serious threat to API security. Attackers exploit missing or weak authorization controls to access sensitive data. Detecting these flaws is challenging due to the diversity of application logic and token usage. No single technique catches all cases perfectly. Our approach combines multiple detection methods. Static analysis reviews code for missing checks and insecure patterns.

Dynamic fuzzing stresses APIs with unexpected inputs, revealing hidden flaws. Token swapping and JWT introspection simulate real-world token misuse scenarios. Large Language Models analyze API responses for semantic clues that signal attacks. Each method covers blind spots left by others. Testing across realistic environments showed promising results. Metrics like precision and recall improved significantly compared to standalone tools. False positives were reduced by layering multiple techniques and cross-validating findings. Stress tests confirmed that the system can handle high loads with minimal delay. Continuous monitoring and automation allow rapid adaptation as new threats emerge. Beyond detection, the system supports security teams with clear, actionable alerts. Detailed logs and reports help prioritize fixes and understand attack patterns. The modular design allows easy integration into existing development pipelines. This fosters a security-first mindset throughout the software lifecycle. Future work includes refining AI models with larger datasets and improving fuzzing strategies for better coverage. Expanding token analysis to cover emerging standards will strengthen defenses further. Overall, the hybrid approach creates a resilient shield against BOLA attacks in complex, dynamic environments. In summary, securing APIs requires more than a single tool or test. It demands a layered, intelligent system capable of evolving with the threat landscape. Our solution meets this need by combining proven techniques with modern AI, delivering reliable, scalable, and practical protection.

References

[1] OWASP Foundation, *OWASP API Security Project*, 2023. [Online]. Available: https://owasp.org/www-project-api-security/

[2] SmartBear, *Swagger OpenAPI Documentation: Designing and Documenting APIs*. [Online]. Available: https://swagger.io/docs/

[3] Traceable AI, *A Deep Dive on the Most Critical API Vulnerability – BOLA (Broken Object Level Authorization)*, 2023. [Online]. Available: https://www.traceable.ai/blog-post/a-deep-dive-onthe-most-critical-api-vulnerability----bola-broken-object-level-authorization

[4] APIsecurity.io, "Facebook and Parler API Vulnerabilities: Clairvoyance in Action," *API Security Weekly*, Issue #116, 2021. [Online]. Available: https://apisecurity.io/issue-116-facebook-parler-api-vulnerabilities-clairvoyance/

[5] EC-Council, *IDOR Vulnerability: Detection and Prevention*. [Online]. Available: https://www.eccouncil.org/cybersecurityexchange/web-application-hacking/idor-vulnerability-detection-prevention

[6] Z. Chen, S. Lin, and X. Jiang, "Automated API Vulnerability Detection using Machine Learning Techniques," *arXiv preprint*, arXiv:2201.10833, 2022. [Online]. Available: https://arxiv.org/pdf/2201.10833

[7] APIsec University, *How I Automated BOLA Detection and Hardened My API – A DevSecOps Tutorial*. [Online]. Available: https://www.apisecuniversity.com/blog/how-i-automated-bola-detection-and-hardened-my-api----a-devsecops-tutorial

[8] API7.ai, *API Security Testing Tools and Techniques*, 2023. [Online]. Available: https://api7.ai/learning-center/api-101/api-security-testing-tools-andtechiniques

[9] Unit 42, Palo Alto Networks, *Automated BOLA Detection and the Role of AI*, 2024. [Online]. Available: https://unit42.paloaltonetworks.com/automated-bola-detection-and-ai

[10] A. Panichella, A. Di Sorbo, and B. Russo, "Security Challenges of Modern APIs: A Systematic Literature Review," *International Journal of Information Security*, vol. 23, 2024. [Online]. Available: https://link.springer.com/article/10.1007/s10207-024-00970-5