Vibe-Coding and Emerging Cybersecurity Vulnerabilities: A Case Study of The Moltbook Data Exposure

Anushka Naikwadi[1], Prof. Jyoti Shrote[2]
[1]*Indira College of commerce & Science, Pune*
[2]*Professor, Indira College of commerce & Science, Pune*
doi.org/10.64643/ijirt.208571-459

***Abstract*—Along with the rise of AI-assisted software development, called "Vibe-coding", new platforms can be built quickly using natural-language instructions provided to the AI-assisted systems, which might result in unexpected security issues. In this paper, social networking platform "Moltbook" developed exclusively for AI-agents represents an example of the security threats associated with AI-related applications. The database of this platform was configured incorrectly in February 2026 and has been exposing 1.5 million agent API keys and 35,000 email addresses and personal messages from the agents because of the exposed authentication key in the client-side code. In this paper, we will explore the reason for the issue and analyze how it relates to other 'Vibe-Coding' related vulnerabilities (for instance, Base44). We think that along with easy creation of complex apps with AI-assistance, this ease can be leaked to the security measures traditionally used to discover issues of such applications and highlight the minimum-security requirements for such apps.**

***Index Terms*—Vibe-Coding; AI-Assisted Software Development; Artificial Intelligence; Cybersecurity; Application Security; Data Exposure; API Key Leakage; Database Misconfiguration; AI Agents; Vulnerability Assessment**

I. INTRODUCTION

Building a working web app used to take weeks. Now you can describe what you want to an AI coding assistant in plain English and have something deployable within a few hours. This has come to be called “vibe-coding,” and it has pushed the barrier to entry so low that people with no programming background at all are shipping applications used by thousands of real users. That is a genuinely impressive shift. It is also where the problem in this paper starts: the human review step that used to sit between writing code and shipping it the step where someone checks for exposed secrets, broken authentication, and misconfigured access controls is often just gone.

That gap is not theoretical. In February 2026, Wiz security researchers found that Moltbook, a social platform built entirely for AI agents to interact with each other, had its entire production database sitting open to the public internet. Nobody had to write an exploit for this. There was no privilege escalation, no phishing, no insider involved a single hardcoded credential sitting in client-facing code was enough to read and write over a million user records. What makes this worth studying is not that the flaw itself was new (it very much was not), but that it shows how fast an old, well-documented vulnerability class can come back once the person who normally catches it is taken out of the loop.

This paper works through the Moltbook case in detail, then places it next to a similar vibe-coding failure disclosed roughly seven months earlier on the Base44 platform, to see whether the two incidents point at the same underlying problem. Section II covers what vibe-coding is and what existing research says about the security quality of AI-generated code. Section III is the Moltbook case study. Section IV covers Base44 for comparison. Section V pulls out what the two cases have in common. Section VI lays out some minimum-security check’s teams should be doing, and Section VII closes things out.

II. BACKGROUND AND RELATED WORK

A.What Vibe-Coding Actually Is

The concept refers to a process whereby the user uses an AI tool to develop the majority or entire code for an application based on natural language prompts rather than manually writing the code [9]. This concept has become commonplace in the industry due to platforms such as Base44, Bolt.new, Lovable, Cursor, and

Replit, among others, and it is not something people do as hobbies any longer since some enterprise executives have gone on record to say that a good number of the code developed in their firms is through AI tools [12].

Clearly, the greatest advantage here is efficiency. What would normally require a programmer is now done by a person who has no background in software development. And there lies the challenge in terms of cybersecurity.

Typically, in a normal development process for secure systems, one can assume that there is a human somewhere who is knowledgeable about security measures including authentication and secret management. The issue is that vibe-coded applications often do without such a person mainly because the actual operator of the tool has no clue when the AI produces an insecure default [10].

B. What the Research Says About AI-Generated Code

It's not even a mere assumption it is backed by facts. According to the 2025 GenAI Code Security Report of Veracode, testing more than one hundred large language models found that 45% of the code samples they created included vulnerabilities from the OWASP Top 10 list [9], [12].

In addition, the academic assessment of 1,689 programs produced with AI revealed that approximately 40% of them contained exploitable vulnerabilities, and most of them belonged to the CWE Top 25 list [11].

Commonly, all the studies reveal the same vulnerability categories such as injection flaws, broken or completely absent authentication, use of hard-coded credentials, and the lack of proper endpoint-specific access control checks [11], [13].

The common characteristic of AI-produced code is that it is crafted to pass initial checks on functionality rather than being ready for exploitation. The vibe-coded functionality may work flawlessly during the demo but contain vulnerabilities that become apparent only when actual users and attackers test the system [10], [13].

Moltbook and Base44 fall precisely into this category. Neither of these cases is any kind of anomaly as the vulnerability category involved is a regular part of the mentioned studies and would be found even during an ordinary secure development checklist process.

III. CASE STUDY: THE MOLTBOOK DATA EXPOSURE

A. *What Moltbook Was*

The site was launched on 28 January 2026 as a social media platform similar to Reddit, the only difference being that all the accounts on Moltbook were required to be AI agents rather than real people. It was described as the "front page of the agent internet" and featured features like posts, comments, voting, and the karma system [1]. The founder of Moltbook, Matt Schlicht, admitted openly that "I didn't write a single line of code for the site, it was all generated by AI" [3], [4]. The site quickly gained popularity and saw a huge spike in traffic just hours after its launch.

B. *How the Flaw Was Found, and Why It Happened*

It is important to note that the Wiz researchers were not engaged in any complex attack; they merely looked around the web app as any regular user would do. In less than a minute, they discovered that there was a Supabase API key right there in the client-side JavaScript [2], [3]. First of all, Supabase is a backend service for building your projects using an open-source hosted PostgreSQL database with a REST API [3]. As can be seen, it is popular among vibe-coded projects due to its ease of use [3]. The crucial point here is that the discovery of a Supabase key does not mean anything bad in itself. As can be seen, Supabase is designed in such a way that the key can be openly used provided that Row Level Security (RLS) is enabled in the related database [3], [5]. However, Moltbook did not enable RLS in their project. Therefore, a single found key granted the attackers access to reading, writing, updating and deleting all the data in all the tables in the production database without authentication.

In order to prove the severity of this issue, Wiz conducted a couple of tests and saw for himself that any text written through the vulnerable endpoint would immediately appear on Moltbook's live feed. In other words, this was not only a reading issue; any data could have been injected into the system in real time [1]. But what was behind this one misconfigured endpoint? 1.5 million API keys of the agents, 35,000 email addresses of users, and personal messages between the agents, some of which even included plaintext credentials from third parties, such as OpenAI API keys that users copied into the chat [1],

[5]. And due to the fact that leaking tokens allowed complete impersonation, anyone could post messages, send messages, and act as any one of Moltbook's 1.5 million agents without authentication [2], [4].

C. *What Happened After, and a Second Problem*

Wiz reported the issue to Moltbook, and the team closed the hole within hours with Wiz's help. All the data the researchers had pulled during testing was deleted afterward, following standard responsible-disclosure practice [1]. A separate researcher stumbled onto the same exposure independently around the same time and posted about it publicly [2]. Along with the problem with the database, Wiz also identified another thing that should be mentioned: there was no rate limiting when it came to registering a new agent, and there was nothing preventing the system from figuring out whether this "agent" was in fact an AI or simply a script being executed by a human. The data from Wiz demonstrated that the population of the platform had been made up mainly by a few humans who operated their bot armies, instead of having a huge number of independent agents as claimed [2]. This is a different kind of bug, but it also stems from the same reason features were released without proper access control or verification checks

IV. COMPARATIVE CASE: THE BASE44 AUTHENTICATION BYPASS

Moltbook is not an isolated weird event something very similar happened about seven months earlier on Base44, an enterprise-facing vibe-coding platform that Wix later acquired. Looking at both side by side is really the point of this paper.

A. *What Went Wrong*

With Base44, users can create full applications using natural language prompts only. According to the report made by mid-2025, Base44 had gained around 20,000 users, including solo developers and enterprise teams developing tools, like chatbots, knowledge bases, HR systems that process PII [15]. On 9 July 2025, Wiz revealed that two of Base44's authentication endpoints related to new user creation and one-time-password verification were left wide open without any authentication at all [6], [7]. Each Base44 application has a non-secret app_id parameter, which is readily available via the publicly exposed URL of the application and its manifest.json file [7], [8]. Wiz discovered that by supplying the non-secret app_id parameter to the exposed endpoint it was possible to register an entire verified account on Base44 app, even those which are set as private with mandatory SSO [6], [8]. Then, the hacker could easily gain access through the SSO flow of the application [6].

B. *How Far It Reached*

Since the flaw existed in Base44's shared platform infrastructure rather than in customers' app code, it became immediately visible in each private application created using the platform, even those dealing with sensitive enterprise data [7], [16]. Wiz discovered vulnerable applications in the wild through two approaches – by identifying enterprise custom domains all of which were linked to a single CNAME record in Base44's infrastructure and by conducting an HTML fingerprinting search for the fingerprints of Base44 login pages [6]. The company, which had acquired Base44 just a month earlier, remedied the problem in less than 24 hours after its discovery and found no signs of exploitation prior to that time [7], [8]. Wiz described the exploit as unusually easy to execute – all you needed was basic knowledge of API [6], [16] and remarked how most of the AI-security talk revolved around sophisticated topics such as prompt injection, whereas the real threat was something very old-fashioned and mundane, like broken authentication [6].

V. ROOT CAUSE ANALYSIS

On the surface these two looks pretty, different one is a database misconfiguration; the other is an authentication bypass. But line them up and they start looking like the same problem wearing different clothes.

A. *Neither One Is Actually New*

Neither of these bugs required any new attack technique. Moltbook comes down to a missing RLS policy on a public backend-as-a-service key this is a well-documented failure mode, not something novel [1], [9]. Base44's issue is a broken-authentication problem paired with an insecure direct object reference: a value that should have stayed private was exposed as a public identifier, and no one added a check on the endpoints that used it [6], [8]. Both bugs

sit squarely in the OWASP Top 10, and both are the kind of thing an ordinary penetration test or even a basic secure-coding checklist would have caught [9], [11], [13].

B. *The Real Problem Is Who Was Missing, Not What Broke*

What actually connects these two cases is not a shared technical quirk it is that the same step is missing from both. Vibe-coding takes a natural-language prompt and turns it into a live, internet-facing app almost directly, with nobody security-aware standing between what the AI wrote and what real users hit [10], [12]. Moltbook's own creator said flatly he wrote none of the code [3], [4]. Base44 shipped a default authentication behavior across its shared infrastructure that its own enterprise customers did not know about and had never consciously set up [7], [15]. In both cases, whoever was driving the AI tool did not have the specific knowledge RLS semantics for one, endpoint-level authorization for the other to notice that what the AI handed them worked fine but was not safe.

C. *Because These Are Platforms, One Mistake Goes a Long Way*

While a common web app vulnerability normally affects just the users of that particular app, Base44 was a piece of shared infrastructure that supported several enterprise applications that were built using different brandings, and thus the issue of authentication on the platform affected all these privately created apps simultaneously [7], [16]. It was the same case for Moltbook – when the incorrectly configured API key was discovered, it allowed access to the entire database in production, not just to one account [1]. This aspect requires special attention, because if there are several individually created apps that use an AI-enabled platform or backend-as-a-service, then one forgotten configuration can be simultaneously reflected across them all.

VI. MINIMUM SECURITY CONSIDERATIONS FOR VIBE-CODED APPLICATIONS

Given all of the above, below are some minimum things that need to be done while building anything using AI-assisted development. Nothing is comprehensive, as the point is to hit the exact vulnerability that the two examples above have in common, which is a security-related default value that was missed by the AI and the developer.

A. *Do Not Trust Backend-as-a-Service Defaults Blindly*

When a platform is running a hosted backend such as Supabase or Firebase, it is imperative to check that the access control configurations such as RLS policies in the case of Supabase have been verified. Don't take it for granted that just because a key is supposed to be public, it is alright to share it [1], [5]

B. *Actually, Check Every Authentication and Registration Endpoint*

Go through every endpoint that creates, verifies, or logs in a user, and ask whether it needs anything beyond a value that is already sitting in client code or a public URL, like an app_id [6], [8]. If an endpoint can be hit without prior authentication, treat whatever it returns as untrusted it does not matter what the UI flow claims is required.

C. *Put a Real (Even If Small) Security Review Before Anything Ships*

Since more than 45% of the code generated by the AI contains vulnerabilities belonging to the OWASP Top 10 list [9], [12], it is important to include some sort of review process by a security-aware human being in between the AI coding and deployment of the application. This does not necessarily mean that everything fancy needs to be caught. From both the examples presented in this paper, it is obvious that it was enough to catch the boring, known issues.

D. *Rate-Limit and Verify Any Flow That Creates New Accounts*

The problem with the Moltbook service that there is no rate-limiting for registering agents, and there is no way to verify whether the "agent" was real or an AI system, demonstrates that the access control problem can appear not only in databases but anywhere where new accounts are being created [2].

E. *Remember a Platform Bug Is Your Bug Too*

But if your application is built on a shared AI-development vibe-coding platform, then any vulnerability that is exposed against the platform itself such as Base44's – can affect your application regardless of whether you made any mistakes. There

should be a way for teams to track disclosures coming out of the AI-development vendor [7], [16].

VII. CONCLUSION

Compare Moltbook and Base44 and the conclusion seems pretty straightforward: the entire premise of vibe-coding destroying the wall between an idea and the app that runs in reality involves some real price in security terms once the reviewer who would've spotted these issues before release falls under the same hammer. Neither of the two cases involved any sophisticated attacker: both were discovered by researchers in the process of a routine investigation, and were related to configuration errors that would've been detected by a basic checklist. With the increasing number of non-technical developers releasing production-ready software using AI tools, the security strategy can't afford to rely on the hope that the AI eventually will be able to write secure code automatically. And based on all the material discussed in this review, such situation seems unlikely to change by itself thus, the bare minimum level of security checking has to become a part of vibe-coding procedure.

REFERENCES

[1] Wiz Research, "Hacking Moltbook: How We Found 1.5M Exposed API Keys," *Wiz Blog*, Feb. 2026. [Online]. Available:
[2] K. Chen, "AI Agent Social Media Network Moltbook Is a Security Disaster," TechRadar Pro, Feb. 2026. [Online]. Available:
[3] Infosecurity Magazine, "Vibe-Coded Moltbook Exposes User Data, API Keys and More," Apr. 2026. [Online]. Available:
[4] CX Today, "Security Flaw in AI Agent Social Network Moltbook Exposes Risks in AI-Built Platforms," Feb. 2026. [Online]. Available:
[5] The Cyber Express, "AI-Coded Moltbook Platform Exposes 1.5 Mn API Keys Through Database Misconfiguration," Feb. 2026. [Online]. Available:
[6] Wiz Research, "Critical Vulnerability in AI Vibe Coding Platform Base44," Wiz Blog, Jul. 2025. [Online]. Available:
[7] K. Poireault, "Critical Authentication Flaw Identified in Base44 Vibe Coding Platform," Infosecurity Magazine, Apr. 2026. [Online]. Available:
[8] R. Lakshmanan, "Wiz Uncovers Critical Access Bypass Flaw in AI-Powered Vibe Coding Platform Base44," The Hacker News, Jul. 2025. [Online]. Available:
[9] Veracode, "2025 GenAI Code Security Report," referenced in: NordLayer, "Vibe Coding Security: Risks and How to Prevent Them," Jul. 2026. [Online]. Available:
[10] Retool, "The Risks of Vibe Coding: Security Vulnerabilities and Enterprise Pitfalls," Retool Blog, Mar. 2026. [Online]. Available:
[11] Cycode, "Vibe Coding Security: Risks and Vulnerabilities," Cycode Blog, 2026. [Online]. Available:
[12] M. Sastry, "Vibe Coding: Managing the Strategic Security Risks of AI-Accelerated Development," Infosecurity Magazine, Feb. 2026. [Online]. Available:
[13] Cloud Security Alliance, "Vibe Coding's Security Debt: The AI-Generated CVE Surge," CSA Research Note, May 2026. [Online]. Available:
[14] OX Security, "Vibe Coding Security: Why 62% of AI-Generated Code Ships with Vulnerabilities," May 2026. [Online]. Available:
[15] K. Poireault, "Critical Flaw in Vibe-Coding Platform Base44 Exposed Private Enterprise Applications," SecurityWeek, Jul. 2025. [Online]. Available:
[16] Search Engine Journal, "Vulnerability Uncovered in Wix Vibe Coding Platform," Jul. 2025. [Online]. Available: