CSAI Foundation

Cloud Security Alliance AI Safety Initiative

Marimo Pre-Auth RCE: AI Development Toolchain Under Attack

CVE-2026-39987 Enables Unauthenticated Shell Access on Exposed Notebook Servers

Unofficial AI-assisted Research

2026-04-10

Key Takeaways

- CVE-2026-39987 is a critical pre-authentication remote code execution vulnerability (CVSS v4.0: 9.3) in Marimo, a reactive Python notebook widely used in AI development workflows, affecting all versions at or below 0.20.4 and patched in version 0.23.0 [1][2].
- The vulnerability exists in Marimo's integrated terminal WebSocket endpoint (`/terminal/ws`), which accepted connections without performing any authentication check, while the application's primary WebSocket endpoint (`/ws`) correctly enforced authentication via `validate_auth()` ; a single WebSocket handshake was sufficient to obtain a full interactive PTY shell [1][3].
- The Sysdig Threat Research Team, monitoring honeypot instances across multiple cloud providers, observed the first exploitation attempt in the wild within 9 hours and 41 minutes of the advisory's publication, with a complete credential theft operation executed in under three minutes — and no public proof-of-concept code existed at the time [3][4].
- Marimo instances configured with AI integrations may hold cloud provider credentials, LLM API keys (OpenAI, Anthropic, Google Gemini), and access tokens stored in environment files or shell histories, making successful exploitation a potential gateway to broader cloud account compromise and AI infrastructure takeover [1][5].
- This incident continues a documented pattern of critical RCEs in AI developer tooling — including Langflow CVE-2026-33017 (exploited in 20 hours) and Flowise CVE-2025-59528 — in which attackers move from advisory publication to working exploits without requiring pre-existing proof-of-concept code [6][7].
- Organizations running Marimo in any networked environment — including local development, shared GPU servers, cloud instances, or Docker deployments — should upgrade to version 0.23.0 immediately and audit whether exposed instances were accessed during the exploitation window beginning April 8, 2026 [2][3].

Background

Marimo is an open-source reactive Python notebook developed as a modern alternative to Jupyter, designed to address persistent reproducibility and state management problems that have long characterized notebook-based data science workflows. Unlike Jupyter notebooks, which store outputs alongside code in JSON format and require users to manually manage cell execution order, Marimo stores notebooks as pure Python files and implements a dataflow execution model: when a cell runs or a variable changes, Marimo automatically propagates updates to all dependent cells, keeping code and outputs consistent without user intervention [5][8]. The platform additionally supports SQL queries against Python dataframes, interactive UI widgets, and deployment modes that allow notebooks to be served as interactive web applications — capabilities that contribute to its adoption in data science, machine learning, and AI research contexts.

Marimo's positioning as an AI-native development platform is central to its recent growth trajectory. The tool integrates AI code generation and cell-level assistance directly into the notebook interface, with support for commercial model providers including OpenAI, Anthropic, and Google Gemini as well as locally hosted models [5]. Because Marimo integrates natively with these commercial LLM providers, deployments configured to use AI-assisted features will contain active API keys. Where these keys are present, they carry billing authority and, in enterprise contexts, may be scoped to access proprietary model configurations or usage data [5]. The platform is actively used by engineering and research teams at organizations including Stanford, Mozilla AI, OpenAI, and BlackRock, and the project had accumulated approximately 19,600 GitHub stars as of the time of this vulnerability's disclosure [5][8].

The server architecture underlying Marimo's notebook interface is WebSocket-based. The application maintains multiple WebSocket endpoints to handle real-time notebook cell execution, UI widget state, and — on platforms that support it — an integrated browser terminal that provides shell access to the environment running the Marimo process. This terminal feature is designed to allow notebook users to run shell commands, manage files, and inspect running processes without leaving the notebook interface, a convenience that proves to be the mechanism through which CVE-2026-39987 operates.

Marimo notebooks are commonly deployed on Docker-based infrastructure, GPU cloud instances via frameworks such as SkyPilot, and platform-as-a-service environments including Hugging Face Spaces and CoreWeave [9]. Many of these deployment patterns result in Marimo processes running with elevated privileges — default Docker images for Marimo run as root — and with direct access to the underlying host's network and file system. The convergence of elevated process privileges, internet-facing deployment patterns, and credential-rich environments makes the exploitation of a pre-authentication shell access vulnerability in Marimo particularly consequential.

Security Analysis

The Vulnerability: Authentication Absent Where It Matters Most

CVE-2026-39987, tracked under GitHub Security Advisory GHSA-2679-6MX9-H9XC, is classified under CWE-306 (Missing Authentication for Critical Function) [2][10]. The root cause is straightforward but severe: Marimo's server implemented authentication enforcement inconsistently across its WebSocket endpoints. The primary notebook communication endpoint, `/ws`, correctly called the application's `validate_auth()` function to verify user identity before accepting connections. The terminal WebSocket endpoint, `/terminal/ws`, performed no equivalent check. Instead, it verified only whether the application was running in a mode that supported terminal access and whether the host platform supported PTY allocation — conditions that are almost always true in the networked deployment scenarios where the vulnerability would be exploited [1][2].

The practical consequence is that any network-adjacent or internet-reachable attacker could establish a WebSocket connection to the `/terminal/ws` endpoint and receive a fully interactive terminal session running as the Marimo process user. The connection required no credentials, no session cookie, no token, and no prior interaction with the application. The attacker received a complete PTY shell with the same permissions as the Marimo process itself — commonly root in Docker-based deployments [1][3]. Endor Labs, who analyzed the vulnerability in depth, characterized it as "root in one request": a single WebSocket handshake was sufficient to complete the attack [1].

The asymmetry between the authenticated `/ws` endpoint and the unauthenticated `/terminal/ws` endpoint suggests the terminal feature may have been added or enabled at a later stage without full integration into the authentication framework that governs the primary WebSocket endpoint. The fix, released in Marimo version 0.23.0 via pull request #9098, closes the authentication gap on the terminal WebSocket by bringing it into alignment with the validation logic already applied to other endpoints [2][3].

Exploitation in the Wild: Speed Without a Playbook

The exploitation timeline documented by Sysdig is notable because it illustrates how minimal the barrier to active exploitation has become for well-resourced threat actors monitoring security advisory feeds. The advisory for CVE-2026-39987 was published on April 8, 2026. Sysdig's honeypot infrastructure — running vulnerable Marimo instances across multiple major cloud providers — recorded the first exploitation attempt at 9 hours and 41 minutes after publication [3][4]. No public proof-of-concept code

existed at that time; the attacker appears to have synthesized a working exploit from publicly available information alone — the advisory was precise enough about the vulnerable endpoint and its missing authentication check that reverse engineering the source code may not have been necessary [3].

The observed attack proceeded methodically once the attacker established shell access. After connecting to the unauthenticated terminal endpoint, the attacker manually explored the compromised environment — examining directory contents, reviewing configuration files, and searching for credential material. A complete credential theft operation, from initial connection to data exfiltration, was completed in under three minutes [3]. The efficiency of this operation reflects attacker familiarity with the conventions of AI development environments: `.env` files, shell histories, Jupyter-compatible credential stores, and cloud SDK configuration directories all represent predictable locations where API keys and cloud provider tokens accumulate in AI development contexts.

This exploitation pattern — advisory-driven, no PoC required, credential-focused, sub-10-hour window — has now characterized multiple AI toolchain RCE incidents in the preceding months. Langflow CVE-2026-33017, a critical code injection vulnerability disclosed in March 2026, saw working exploits appear within 20 hours of advisory publication; that vulnerability similarly required no prior authentication and no existing proof-of-concept code [6]. Flowise CVE-2025-59528, a CVSS 10.0 node code injection vulnerability, demonstrated that even a six-month patch lag does not guarantee exploits remain dormant: active exploitation was detected long after the fix was available [7]. The consistent thread across these incidents is that AI development tooling occupies a gap in organizational security posture — treated as development infrastructure rather than production systems, and therefore subject to weaker patch management, less network segmentation, and less monitoring.

Credential Exposure at the Intersection of AI and Cloud

The specific risk profile of a compromised Marimo instance is shaped by what AI developers store in their working environments. Unlike a compromised web server that might expose user data or database contents, a compromised AI development notebook environment typically exposes the credentials that give access to the full breadth of an organization's AI infrastructure. Marimo's native integrations with commercial LLM providers mean that active API keys for OpenAI, Anthropic, and Google are a common fixture of configured environments [5]. Cloud provider credentials — whether sourced from `.env` files, environment variables, or cloud SDK configuration directories — are equally common in notebook environments that connect to training infrastructure, cloud storage, or managed database services.

A successful exploitation of CVE-2026-39987 therefore functions not merely as a server compromise but as a credential harvesting operation with potentially far-reaching consequences. API keys exfiltrated from a Marimo instance may enable an attacker to submit queries to LLM providers under the victim organization's account, incurring direct billing cost, and may expose usage metadata — including model

selection and prompt volume patterns — that reveal proprietary AI development activities. In organizations where the same API key governs access to fine-tuned models or batch inference pipelines, the scope of exposure extends further. Cloud credentials may enable lateral movement to cloud storage containing training datasets, model artifacts, or inference pipelines — infrastructure assets that carry both competitive sensitivity and regulatory significance where the underlying data includes personal information.

Recommendations

Immediate Actions

The single most important action for any organization running Marimo is to upgrade to version 0.23.0 or later. The upgrade closes the authentication gap in the terminal WebSocket endpoint and is the primary remediation for CVE-2026-39987 [2][3]. Version verification should confirm that `marimo --version` reports 0.23.0 or above; organizations relying on container images should update both the base image and any pinned dependency specifications.

Because exploitation was observed within 10 hours of the April 8, 2026 advisory publication, organizations that ran Marimo on internet-accessible infrastructure before upgrading should treat any deployment as potentially compromised and initiate incident response procedures accordingly. This means reviewing access logs for WebSocket connections to `/terminal/ws` — particularly those occurring on or after April 8 — and rotating all credentials that were accessible within the compromised environment, including API keys for LLM providers and cloud service credentials. Cloud provider access logs should be reviewed for anomalous API calls or resource provisioning activity that may indicate downstream use of exfiltrated credentials.

For instances that cannot be immediately upgraded, a network-level control that restricts access to the Marimo port (by default 2718) to known, trusted IP addresses represents a compensating control that eliminates the attack surface for unauthenticated remote exploitation. This does not substitute for upgrading, but it reduces the exposure window for organizations that require additional change management before deployment.

Short-Term Mitigations

Organizations deploying Marimo in shared environments — shared GPU servers, team-accessible cloud instances, or cloud-based notebook infrastructure — should implement reverse-proxy-based authentication in front of the Marimo server. Even with the CVE-2026-39987 patch applied, defense in depth principles call for an additional authentication layer in front of any interactive development tool that provides shell access, since application-layer authentication alone cannot eliminate the risk of future bypass vulnerabilities [11]. Solutions such as Nginx with OAuth2 proxy, Cloudflare Access, or AWS ALB authentication listeners provide a suitable perimeter that reduces the blast radius if future vulnerabilities emerge in Marimo's own authentication logic.

Container deployments should be reviewed against least-privilege principles. Default Marimo Docker images run as root, meaning that a shell obtained through the terminal endpoint carries root privileges on the container [1]. Modifying Dockerfiles to run Marimo as a non-root user — consistent with the example configurations documented in Marimo's own deployment guides [9] — significantly reduces the impact of any future terminal-level compromise. Separately, secrets should not be stored in `.env` files within containers at all where cloud provider secrets managers or injected environment variables can be used instead, as this practice limits the credential exposure window for any container-level compromise. Marimo's published security documentation provides additional guidance on hardening server deployments [12].

AI development tooling more broadly should be brought within the scope of existing vulnerability management programs. Marimo, Langflow, Flowise, JupyterHub, and similar platforms represent a distinct class of infrastructure — development-facing, commonly privileged, frequently credential-rich — that often falls outside the scope of production-grade security monitoring programs, despite carrying risk profiles comparable to production systems. Integrating these platforms into software composition analysis (SCA) pipelines and subscribing to their security advisories through GitHub advisory watches or package management security feeds ensures that critical disclosures reach the teams responsible for patching these environments before the exploitation window closes.

Strategic Considerations

The Marimo vulnerability is best understood not as an isolated defect in a single tool but as a data point in an emerging threat pattern: the systematic targeting of AI development toolchains by actors who recognize that these environments combine high credential value with low security maturity. The advisory-to-exploitation timelines observed across Marimo, Langflow, and Flowise — ranging from under 10 hours to under 24 hours — suggest that threat actors are actively monitoring AI development tool security advisories and maintaining the capability to convert advisory-quality information into working

exploits rapidly. Organizations should plan on the assumption that this capability will continue to mature and that the window between disclosure and active exploitation may continue to shrink — making patch-cycle-only strategies insufficient without additional preventive controls in place.

This requires a structural shift in how organizations classify AI development infrastructure. Notebook servers and AI workflow platforms that hold production API keys and connect to cloud data infrastructure are not development conveniences — they are privileged access systems that warrant the same security controls as production-grade infrastructure. Network segmentation should ensure these systems are not internet-facing without authentication; access should be logged and reviewed; and the credentials they hold should be scoped to the minimum necessary privileges and rotated on regular cycles rather than being treated as long-lived configuration.

At the organizational level, the pattern of AI toolchain vulnerabilities calls for a formal inventory of AI development tools, their network exposure, their privilege levels, and the credentials accessible from them. This inventory provides the foundation for prioritized patch management, targeted monitoring, and coherent incident response planning for future critical vulnerabilities in this class of tooling.

CSA Resource Alignment

CVE-2026-39987 and the broader AI toolchain exploitation pattern it represents connect directly to several areas of active CSA guidance. The MAESTRO framework's threat modeling methodology for agentic AI systems identifies the AI development toolchain as a distinct attack surface requiring explicit threat analysis [13]. The missing authentication in Marimo's terminal WebSocket illustrates precisely the class of unauthenticated interface vulnerabilities that AI threat modeling frameworks are designed to surface during structured threat analysis. Organizations applying MAESTRO to their AI development environments should include interactive notebook platforms and AI workflow tools in their threat model scope, not just the production AI systems those tools are used to build.

The AI Controls Matrix (AICM), CSA's comprehensive AI security control framework, addresses supply chain security and AI development environment controls across its control domains [14]. AICM guidelines covering application providers and orchestrated service relationships address authentication controls on internal development infrastructure and the management of API credentials within AI development environments. The Marimo incident illustrates the practical risk that AICM's credential management and access control domains are designed to mitigate, and organizations using AICM as a governance framework should map the relevant controls in those domains to their Marimo and equivalent tooling deployments.

CSA's Zero Trust guidance applies directly to the network access model that would have prevented exploitation of CVE-2026-39987 in most cases [15]. A Zero Trust architecture that enforces identity-based access control at the network edge — rather than relying on the application-layer authentication of individual tools — provides a compensating control that limits the impact of authentication failures within individual applications. The Marimo exploitation scenario, in which an unauthenticated network request reached the vulnerable endpoint, would be blocked in a Zero Trust architecture that required authenticated network sessions before any traffic reached the Marimo port.

CSA's STAR for AI program is developing a Catastrophic Risk Annex expected to encompass scenarios in which AI development infrastructure compromise leads to broader AI system manipulation or supply chain poisoning. The credential exfiltration pattern observed in the Marimo exploitation honeypot — where attackers focused on obtaining LLM provider API keys and cloud credentials within minutes of gaining shell access — represents the initial access phase of precisely the kind of AI supply chain attack scenario the Annex is designed to address.

References

[1] Endor Labs. "Root in One Request: Marimo's Critical Pre-Auth RCE (CVE-2026-39987)." Endor Labs Blog, April 2026.

[2] GitHub Security Advisory. "GHSA-2679-6MX9-H9XC: Unauthenticated Remote Code Execution in marimo Terminal WebSocket." GitHub, April 2026.

[3] Sysdig Threat Research Team. "Marimo OSS Python Notebook RCE: From Disclosure to Exploitation in Under 10 Hours." Sysdig Blog, April 2026.

[4] The Hacker News. "Marimo RCE Flaw CVE-2026-39987 Exploited Within 10 Hours of Disclosure." The Hacker News, April 2026.

[5] marimo-team. "marimo: A reactive notebook for Python." GitHub, accessed April 2026.

[6] The Hacker News. "Critical Langflow Flaw CVE-2026-33017 Triggers Attacks within 20 Hours of Disclosure." The Hacker News, March 2026.

[7] Dark Reading. "RCE Flaw in AI Coding Tool Poses Software Supply Chain Risk." Dark Reading, April 2026.

[8] marimo-team. "marimo | a next-generation Python notebook." marimo.io, accessed April 2026.

[9] marimo Documentation. "Deploying marimo with Docker." marimo Docs, accessed April 2026.

[10] GitLab Advisory Database. "Marimo: Pre-Auth Remote Code Execution via Terminal WebSocket Authentication Bypass." GitLab GLAD, April 2026.

[11] OWASP. "WebSocket Security Cheat Sheet." OWASP Cheat Sheet Series, accessed April 2026.

[12] Marimo Documentation. "Security – marimo." marimo Docs, accessed April 2026.

[13] CSA AI Safety Initiative. "MAESTRO: A Multi-Layered AI Threat Modeling Framework." Cloud Security Alliance, 2025.

[14] Cloud Security Alliance. "AI Controls Matrix (AICM)." Cloud Security Alliance, 2025.

[15] Cloud Security Alliance. "Zero Trust Advancement Center." Cloud Security Alliance, accessed April 2026.

