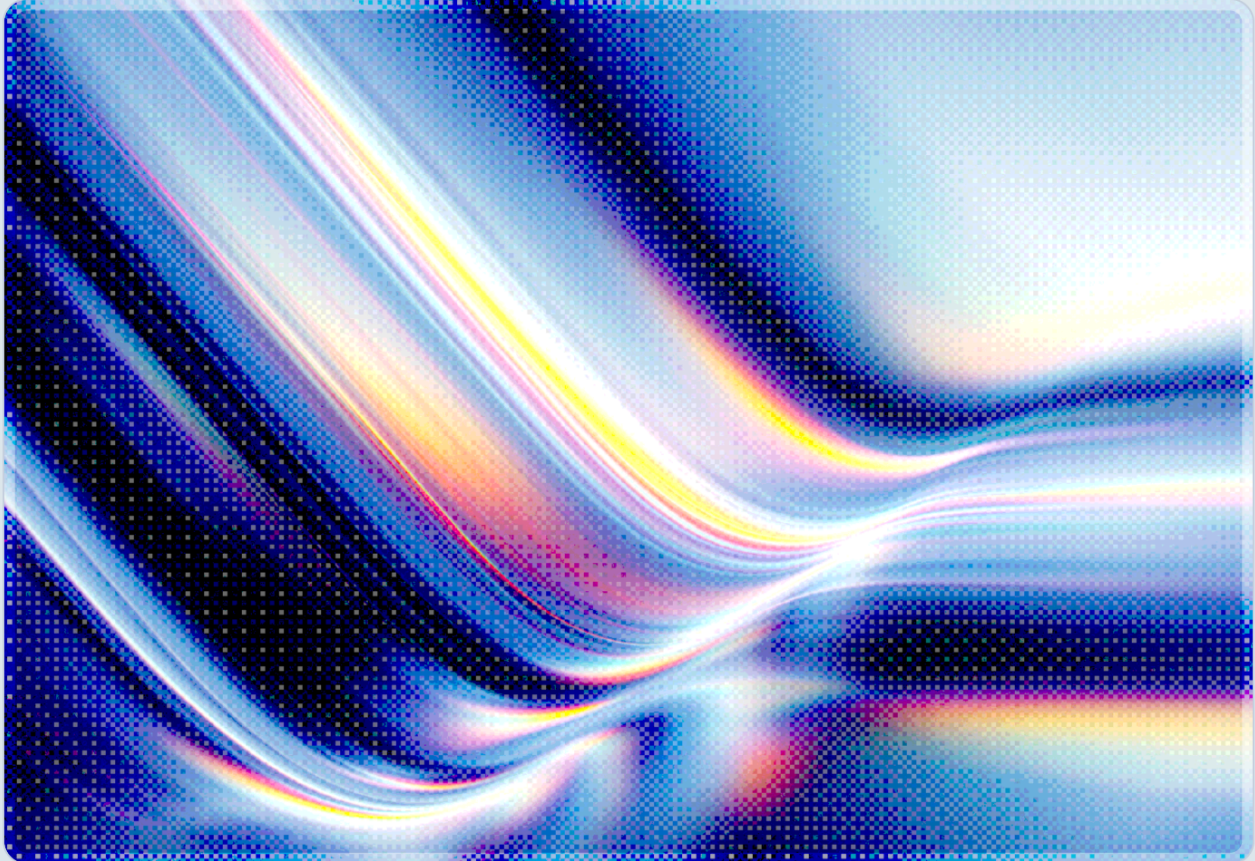


Config Files as Code: Unsloth CVE-2026-93348 and `trust_remote_code`

Security Implications and Guidance for Model-Loading Pipelines

2026-10-30

 AI-assisted Rapid Research



CSAI

cloud
security
alliance®

© 2026 Cloud Security Alliance. Some rights reserved.

You may download, store, display, view, print, redistribute, and link to this document in its original, unmodified form, provided that attribution to the Cloud Security Alliance is maintained and all trademark and copyright notices remain intact.

This document may not be modified or altered. You may quote portions of the document as permitted by the Fair Use provisions of the United States Copyright Act, provided that attribution is given to the Cloud Security Alliance.

This document may be shared on professional and social media platforms in its original form with attribution.

This document was generated with AI assistance and has not undergone official CSA review and approval processes.

Key Takeaways

CVE-2026-93348 is a code injection vulnerability (CWE-94) in Unsloth Zoo, the support library behind the popular open-source Unsloth fine-tuning framework. During model loading, Unsloth read the `model_type` value from a model's `config.json`, inserted it into a generated Python import statement, and ran that statement through `exec()`. Because the value was not checked against a character allowlist, an attacker who publishes a model with a newline and Python source in a nested `model_type` field can run arbitrary code as the user who loads the model for training or inference [1] [2]. VulnCheck, the CVE Numbering Authority, rates the flaw 8.6 under CVSS v4.0 and 8.1 under CVSS v3.1. Red Hat lists a CVSS v3.1 score of 8.8 [1][3].

The most important point for practitioners is that `trust_remote_code=False` does not protect affected versions. The vulnerable code path belongs to Unsloth itself and never passes through the Hugging Face remote-code mechanism, so the setting that many teams treat as the safety switch for untrusted models has no effect [4]. Execution also happens early in the load. Pillar Security's testing indicated that injected code could run before model weights or tokenizers were loaded, which means weight-format controls such as safetensors offer no protection [4]. The affected range is long as well: Unsloth Zoo 2025.9.9 up to (not including) 2026.8.14 and Unsloth 2025.9.9 up to (not including) 2026.8.20 are vulnerable, which covers close to a year of releases [1][2].

The fix is available as a version upgrade. Organizations should move to Unsloth Zoo 2026.8.14 or later and Unsloth 2026.8.20 or later, then rebuild any training or notebook images that pin older versions [2] [4]. Beyond patching, the integrity of fine-tuned outputs deserves attention, because code running inside a compromised fine-tuning job can alter the resulting model before it is published as an organization's own trusted artifact. Finally, the lesson extends beyond Unsloth. Any framework that turns model metadata into executable code makes "configuration" an execution channel, and organizations should treat every field of a downloaded model artifact as untrusted input, not only the files labeled as code.

Background

Unsloth is an open-source framework that speeds up and reduces the memory needed for fine-tuning and serving open-weight large language models. Its performance work happens largely in the companion package `unsloth-zoo`, which patches and recompiles Hugging Face Transformers model code at

load time. This compile step is where the vulnerability sits. To decide which Transformers modules to patch, Unsloth works out a model's architecture by reading the `model_type` values in the model's configuration, including the configs nested inside multimodal and composite models [1][2].

The vulnerability is easier to see against the trust model that Hugging Face Transformers sets for model artifacts. In Transformers, a model repository can include custom Python modeling code that the library downloads and runs only when the caller sets `trust_remote_code=True` [5]. The Transformers documentation tells users to "take extra precaution when loading a custom model" and to pin a specific revision so the model code cannot change underneath them [5]. Because the documentation frames the flag as the gate for running repository code, it is natural for teams to read it as the boundary between loading data and running someone else's code. In CSA's observation, and as reflected in coverage of this disclosure [4], many teams do treat `trust_remote_code=False` that way, and internal model-intake rules often reduce to "never set `trust_remote_code=True` on an unvetted model." Under that reading, a model whose repository contains only JSON configuration and safetensors weights is assumed to be inert.

CVE-2026-93348 breaks that assumption without touching the remote-code mechanism. According to the advisory, the `get_transformers_model_type()` function in `hf_utils.py` collected `model_type` values from nested configs without enforcing a character allowlist, so newlines and Python source survived normalization. The `unsloth_compile_transformers()` function then built an import statement from those values and ran it with `exec()` [1][2]. A crafted `config.json` that ends the import line with a newline and appends attacker-chosen statements therefore gets code execution through an ordinary model load. No custom modeling file, no pickle, and no opt-in flag is involved.

Ariel Fogel of Pillar Security found and reported the flaw [2]. CSO Online reports that Pillar disclosed it privately on August 22, 2026, and that an initial fix reportedly landed within about six hours [4]. The public repository history matches a fast fix. Unsloth Zoo pull request #1083, "Stop building imports from a config-supplied model_type," was merged on August 23, 2026, and a broader, package-wide hardening pull request (#1108) followed on August 26 [6][7]. Pillar's researchers said the maintainers did not respond to later disclosure communications, and the CVE was ultimately assigned through VulnCheck as CNA [2][4]. The CVE record was published on September 28, 2026 [1][2], and wider press coverage appeared in late October [4][8]. Downstream redistributors have since moved. Red Hat's distributed-workloads repository, for example, merged a pull request raising the pinned unsloth-zoo version in its ROCm and CUDA training images to remediate the CVE [9].

This is the second Pillar Security disclosure against Unsloth in 2026. Earlier in the year, Pillar reported that Unsloth Studio's model-inspection backend defaulted to `trust_remote_code=True` when it read a model's configuration, so simply selecting a malicious model in the Studio interface could run code from the repository through an `auto_map` entry. Pillar reports that the maintainers were responsive and shipped a fix, released in Unsloth 2026.6.9. Unsloth did not publish an advisory, citing the feature's beta status and Hugging Face's malware scanning, and no CVE was assigned; Pillar disagreed with the severity assessment but acknowledged the decision as defensible for a beta feature [10][11]. The two flaws differ in kind. The Studio issue was a case of a tool turning remote code on for the user, while CVE-2026-93348 is a case of a tool running code without involving the remote-code mechanism at all. Seen together, they illustrate why a single boolean flag should not be relied on as the trust boundary for model loading.

Security Analysis

Vulnerability Summary

The authoritative record for CVE-2026-93348 is spread across the CNA advisory, the CVE record as aggregated by OpenCVE and OSV, and the upstream fix history. The table below brings these sources together so that responders can see the affected component, version ranges, and severity ratings in one place.

Attribute	Detail
Identifier	CVE-2026-93348 [1][2]
Weakness	CWE-94, Improper Control of Generation of Code [1]
Vulnerable component	<code>get_transformers_model_type()</code> in <code>hf_utils.py</code> , with the <code>exec()</code> sink in <code>unsloth_compile_transformers()</code> [2]
Affected versions	unsloth-zoo $\geq 2025.9.9$, $< 2026.8.14$; unsloth $\geq 2025.9.9$, $< 2026.8.20$ [2]
Fixed versions	unsloth-zoo 2026.8.14; unsloth 2026.8.20 [2]

Attribute	Detail
CVSS v4.0 (CNA)	8.6 High, AV:N/AC:L/AT:N/PR:N/UI:P/VC:H/VI:H/VA:N [2]
CVSS v3.1 (CNA / Red Hat)	8.1 High (CNA); 8.8 High (Red Hat, adds high availability impact) [1]
Trigger	Loading an attacker-crafted model for training or inference [2]
Exploitation status	Not listed in CISA KEV as of the CVE record's last update [1]
Credit	Ariel Fogel, Pillar Security [2]

The CVSS vectors match the attack scenario. The attack is network-reachable in the sense that the malicious artifact is fetched from a remote hub, and it requires no privileges. User interaction is rated passive because loading a model is the user's ordinary, intended action; once a user or automated pipeline chooses to load the model, no further deliberate step is needed to trigger the payload. The confidentiality and integrity impacts are rated high because the attacker's code runs with the full permissions of the loading process [1][2]. The difference between the CNA and Red Hat v3.1 scores comes only from the availability rating and does not change the remediation priority. Shortly after the fix, Unsloth also raised its own dependency floor on `unsloth_zoo` to 2026.8.15, so current Unsloth releases pull in a patched Zoo by default [12].

Why the Trust Flag Does Not Apply

The central lesson of this CVE is architectural. `trust_remote_code` is a gate inside Hugging Face Transformers' `AutoClass` loading logic. It controls whether Transformers will import Python modules shipped in a model repository [5]. It does not control, and cannot control, what downstream frameworks do with configuration values after Transformers has parsed them. Unsloth's compile path sits on top of Transformers and took a string from configuration, put it into source code, and ran that code itself [2] [6]. As CSO Online summarized, setting `trust_remote_code=False` "does not prevent exploitation" because the vulnerable path "does not rely on Hugging Face's remote-code mechanism" [4].

This pattern is likely to recur. Several AI frameworks, Unsloth among them, use dynamic code generation to patch model classes, enable kernel fusions, or support new architectures without hand-written integration code. Where those techniques build source text from metadata in model artifacts, each metadata field becomes a potential injection point, whatever the remote-code setting says. The breadth of Unsloth's own follow-up work supports this reading. Pull request #1108 audited the remaining `exec`, `eval`, and `getattr` sites that received config-supplied values in the compiler, the empty-model builder, vLLM utilities, PEFT utilities, and tokenizer utilities. It also added a CI lint that flags `exec`, `eval`, or `compile` calls whose argument is built from an interpolated string [7]. The maintainers' decision to sweep the whole package rather than patch only the reported site suggests they also saw the problem as a class of bugs rather than a single defect.

Exposure and Impact

Unsloth's typical deployment environments raise the stakes. Fine-tuning usually runs in notebooks, GPU training workers, and container images that hold the credentials the job needs. That can include Hugging Face tokens with write access, cloud storage credentials for datasets and checkpoints, experiment-tracking keys, and sometimes SSH keys. Coverage of the disclosure notes that successful exploitation could expose proprietary datasets, model artifacts, Hugging Face tokens, and cloud credentials [4][8]. Code running inside a training job is also well placed to tamper with outputs, for example by altering fine-tuned weights before they are pushed to a registry. This makes model integrity an exposure in its own right, separate from credential theft. CSA's analysis suggests that the integrity risk deserves particular attention, because a compromised fine-tuning run can produce a backdoored model that is then distributed as an organization's own trusted artifact.

CSA assesses that delivery would not be a significant obstacle for an attacker. Public model hubs host large numbers of community fine-tunes, quantizations, and merges, and practitioners routinely pull them by name for experiments. An attacker needs only to publish a model whose config looks plausible and whose architecture is one Unsloth supports, then attract downloads through naming, model cards, or social promotion. Because the payload sits in a nested `model_type` field of a JSON file rather than in a binary blob, it may not attract attention from scanners focused on pickle deserialization, and it is small enough to miss casual human review. CSA has not found public reports of in-the-wild exploitation, and the CVE record listed an EPSS probability of about 0.4% as of its last update on September 30, 2026 [1]. Given how hard this kind of attack is to detect in ephemeral training workers, the absence of reported exploitation should be read as an absence of evidence, not as evidence of safety.

The Fix and Its Design

The upstream remediation shows what a sound repair looks like. Pull request #1083 normalized `model_type` (rewriting `-`, `/`, and `.` to `_`) and then validated the result against a strict lowercase alphanumeric-and-underscore pattern, at the one function every caller passes through. It replaced the `exec`-built import with `importlib.import_module`, and it replaced an `exec`-built attribute assignment with a walk along the dotted module path [6]. According to the pull request, the maintainers checked that all 5,489 model-type identifiers exposed by Transformers 4.57.6, along with comparable sets across other Transformers versions they tested, still pass the new guard. They also added unit tests covering injection payloads such as quotes, semicolons, newlines, NUL bytes, and comment characters [6].

Pull request #1108 then added defense in depth by re-validating at the sink itself and by checking architecture names against those registered with Transformers before any dynamic attribute lookup [7]. In short, the fix combined input allowlisting, removal of dynamic-execution sinks, and automated enforcement. Teams that maintain their own model-loading glue code can use the same three-part approach.

Recommendations

Immediate Actions

The priority is to find and upgrade every affected installation. Organizations should inventory Python environments, notebooks, Dockerfiles, lockfiles, and base images for `unsloth` and `unsloth-zoo`. They should upgrade to Unsloth Zoo 2026.8.14 or later and Unsloth 2026.8.20 or later, and rebuild and redeploy any container images that pinned earlier versions [2][4]. Teams consuming vendor-built training images should confirm that their vendor has shipped an updated image; Red Hat's image-bump pull request is an example of downstream remediation [9]. Inventory work should include shared GPU hosts and hosted notebook environments, where older environments are often left in place.

Where an affected version may have loaded models from outside a vetted set, teams should treat the environment as potentially exposed. They should review which third-party models were loaded through Unsloth since September 2025 and inspect the `config.json` files of those models, including nested sub-configs, for `model_type` values containing characters other than lowercase letters, digits, underscores, hyphens, and periods. Legitimate model types such as `xlm-roberta` and `vision-encoder-decoder` contain hyphens, so those characters alone are not a signal; newlines, quotes,

semicolons, parentheses, and whitespace are strong indicators of a crafted payload. Teams should also rotate the Hugging Face tokens and cloud credentials that were available to those jobs, and check fine-tuned models produced in suspect environments for integrity before further distribution.

Short-Term Mitigations

Over the next several weeks, organizations should redefine what "trusted model" means in their intake policies. A policy that only says "do not set `trust_remote_code=True`" is no longer enough. Intake controls should validate configuration content itself, for example by rejecting any `model_type` or architecture string that does not match the expected character set, and should pin models to specific commit hashes, as the Transformers documentation recommends [5]. A curated internal model registry or proxy that mirrors only vetted model revisions offers stronger assurance than pulling directly from public hubs at training time.

Model loading should also be isolated from credentials. Training and inference jobs that load third-party artifacts should run with least-privilege identities. They should use short-lived, narrowly scoped tokens, have no standing write access to production registries, and restrict network egress, so that code execution during loading cannot easily reach high-value secrets. Running a first load in a disposable, credential-free sandbox, using the same framework and loading path as the production job and with execution monitoring in place, limits the impact of this class of payload and can surface it before the model enters a privileged training environment. Such isolation should complement, not replace, content validation, since a payload may behave benignly when it detects a sandbox or missing network access. Teams that maintain their own model-loading or patching code should audit it for `exec`, `eval`, `compile`, and dynamic `getattr` calls that receive values from model artifacts, using the CI lint approach Unsloth adopted [7].

Strategic Considerations

Over the longer term, this CVE argues for treating every model artifact as executable input until shown otherwise. The industry's broad move to safetensors has sharply reduced one major code-execution channel, pickle deserialization of weights, wherever it has been adopted. CVE-2026-93348 shows that configuration files, tokenizer definitions, chat templates, and other metadata can become execution channels whenever downstream tooling interprets them dynamically. Security teams should press AI framework maintainers, through procurement requirements, open-source contributions, and vulnerability reporting, to avoid building executable code from artifact metadata and to document clearly which code paths a trust flag actually governs.

Organizations should also treat the AI training stack as a software supply chain subject to the same rigor as production code. That includes software bills of materials for training images, AI bills of materials recording model provenance and revision hashes, dependency monitoring for advisories in fast-moving libraries such as Unsloth, Transformers, and vLLM, and incident response playbooks that cover poisoned model outputs as well as stolen credentials. Disclosure practices matter as well. In both 2026 Unsloth issues, the advisory process did not proceed as the researchers expected, and the earlier Studio flaw was fixed without a CVE [4][10]. Organizations that rely on community AI tooling should therefore track upstream commits and researcher disclosures, not only formal CVE feeds.

CSA Resource Alignment

CSA's most directly relevant prior work is its October 2026 research note on the earlier Unsloth flaw, *Model Inspection as a Code-Execution Vector in Unsloth Studio* [11]. That note described how hardcoded `trust_remote_code=True` settings in LMDeploy, vLLM, InstructLab, and Unsloth Studio overrode user choice, and it recommended auditing `trust_remote_code` usage across the stack. CVE-2026-93348 extends that analysis in an important way. Auditing the flag is necessary but not sufficient, because a framework can reach code execution through paths the flag never governs. Readers who acted on the earlier note's recommendations to sandbox model inspection and keep allowlists of vetted model sources are already partly protected against this variant. The audit of `trust_remote_code` should now be widened into an audit of every dynamic-execution sink that consumes artifact data.

CSA's research note *LangGraph Checkpoint RCE: SQL Injection to Code Execution* [13] documented a parallel pattern in agent frameworks: long-understood application security weaknesses, in that case SQL injection and unsafe deserialization, resurfacing in AI-native components and chaining to code execution. CVE-2026-93348 is another instance. Code injection through unsanitized string interpolation into `exec()` is a textbook CWE-94 defect, made more serious by the high-privilege, credential-rich environments in which AI frameworks run. Both cases support the view that AI framework code should go through the same secure-coding review, static analysis, and dangerous-sink linting expected of any internet-facing application.

The CSA AI Controls Matrix (AICM) v1.1 [14] gives the control vocabulary for operationalizing these recommendations. Its supply chain management, transparency, and accountability controls apply to model provenance, revision pinning, and AI bills of materials for third-party models and libraries. Its application and interface security controls apply to input validation and the removal of dynamic code execution in model-loading code. Its threat and vulnerability management controls apply to tracking advisories and patching training images, and its identity and access management controls apply to

scoping the credentials available to training workloads. Organizations should map their model intake and fine-tuning pipelines against these AICM domains to confirm that model artifacts, and not only application code, are inside the scope of their supply chain controls.

References

- [1] OpenCVE. "[CVE-2026-93348 - Vulnerability Details](#)." OpenCVE, updated September 30, 2026.
- [2] VulnCheck. "[Unsloth Zoo Code Injection via model_type in config.json](#)." VulnCheck Advisory, September 28, 2026.
- [3] Red Hat. "[Bug 2542526 - CVE-2026-93348](#)." Red Hat Bugzilla, September 28, 2026.
- [4] Sharma, Shweta. "[Trusting Unsloth's trust setting could mean executing malicious code](#)." CSO Online, October 28, 2026.
- [5] Hugging Face. "[Loading models](#)." Hugging Face Transformers Documentation, accessed October 2026.
- [6] Unsloth AI. "[Stop building imports from a config-supplied model_type \(Pull Request #1083\)](#)." GitHub, August 23, 2026.
- [7] Unsloth AI. "[Stop config-supplied values reaching exec, eval and getattr sinks \(Pull Request #1108\)](#)." GitHub, August 26, 2026.
- [8] The Hacker News. "[ThreatsDay: Critical Firewall Flaws, Banking Trojans Rented, Weaponized PDF, Token Theft + 26 More Stories](#)." The Hacker News, October 29, 2026.
- [9] Red Hat Data Services. "[Bump unsloth-zoo to 2026.9.9 to fix CVE-2026-93348 \(Pull Request #1091\)](#)." GitHub, October 6, 2026.
- [10] Pillar Security. "[Look, Don't Load: Model Inspection in Unsloth Studio Leads to Critical Arbitrary Code Execution](#)." Pillar Security Blog, September 29, 2026.
- [11] Cloud Security Alliance. "[Model Inspection as a Code-Execution Vector in Unsloth Studio](#)." CSA AI Safety Initiative, October 1, 2026.
- [12] Unsloth AI. "[Update pyproject.toml: require unsloth_zoo >= 2026.8.15 \(commit 92ee020\)](#)." GitHub, August 25, 2026.
- [13] Cloud Security Alliance. "[LangGraph Checkpoint RCE: SQL Injection to Code Execution](#)." CSA AI Safety Initiative, June 12, 2026.
- [14] Cloud Security Alliance. "[AI Controls Matrix \(AICM\) v1.1](#)." Cloud Security Alliance, 2026.