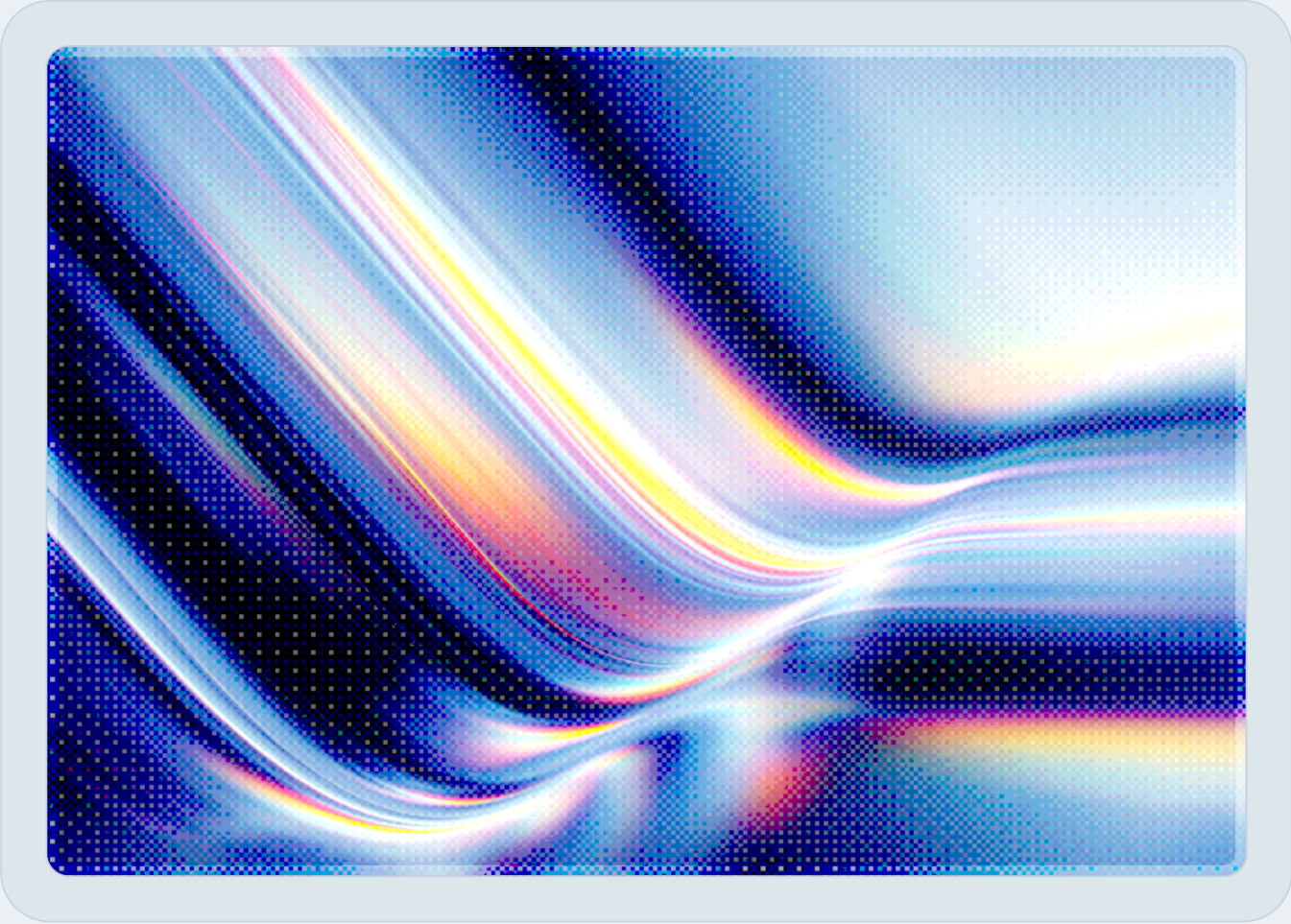
CSAI Foundation | Cloud Security Alliance

FaceHugger: Diffusers Flaws Bypass trust_remote_code

2026-08-03

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

- Three vulnerabilities in Hugging Face's Diffusers library, collectively named FaceHugger by the researchers who found them, allow a malicious model repository on the Hugging Face Hub to execute arbitrary code on a victim's machine even when the caller explicitly sets `trust_remote_code=False` [1][2].
- The three flaws share a structural root cause: Diffusers' trust check runs at one call site (`download()`) while the code that actually executes custom modules runs at a separate, downstream call site [1][2]. The specific mechanism differs across the three CVEs, however: CVE-2026-45804 is a genuine time-of-check-to-time-of-use race across the roughly 0.3-second gap between two HTTP requests, while CVE-2026-44513 and CVE-2026-44827 stem from code paths where the check is never invoked, or evaluates the wrong condition, rather than from a race condition [3][4][5][6].
- The three tracked issues are CVE-2026-44513 [3][4] and CVE-2026-44827 [5] (both CVSS 8.8) and CVE-2026-45804 [6] (CVSS 7.5); all three were fixed in Diffusers 0.38.0, released May 1, 2026 [1][2].
- Diffusers is downloaded millions of times a month and is embedded in production inference services, CI/CD pipelines, and container images, so any organization pulling model repositories or custom pipelines from the Hub without pinning a patched version remains exposed [1][2].
- The pattern is not unique to Diffusers: a related bypass of the same `trust_remote_code` safeguard in the separate Transformers library (CVE-2026-4372) surfaced around the same period, suggesting that both libraries' trust checks share a similar structural weakness — code loaded through multi-step processes is prone to leaving at least one path unchecked — though it is not yet established whether this reflects a broader pattern across the Hugging Face ecosystem or is specific to these two libraries [2].

Background

Hugging Face's Diffusers library is one of the most widely used open-source frameworks for running diffusion-based generative models, with over 7-8 million monthly downloads [1][2]. It is typically invoked through a single high-level call, `DiffusionPipeline.from_pretrained()`, that downloads model weights and pipeline code directly from a repository on the Hugging Face Hub. Because Hub repositories can bundle arbitrary Python files alongside model weights, Diffusers has long included a `trust_remote_code` parameter intended to gate whether that custom code is allowed to run; the parameter defaults to `False`, and organizations that call `from_pretrained()` without explicitly setting it to `True` reasonably expect that no author-supplied Python will execute during the load. That expectation is the basis for how most enterprises reason about the risk of pulling third-party models: the trust flag is treated as a hard boundary between "just weights" and "code that runs with my credentials."

Security researchers at Zafran Security, Gal Zaban and Ido Shani, examined that boundary and found it did not hold in several distinct ways. Their research, published July 27, 2026, and picked up by outlets including The Hacker News and Infosecurity Magazine, showed that a specially crafted Hub repository could cause Diffusers to execute attacker-controlled code during a call that appeared, from the caller's perspective, to have every safety flag correctly set [1][2]. The researchers grouped their findings under the name FaceHugger, and Hugging Face's maintainers, working with independent reporter Vancir on the primary variant, shipped a coordinated fix in Diffusers 0.38.0 on May 1, 2026, ahead of the July public disclosure [3][4].

The underlying design flaw follows the time-of-check-to-time-of-use pattern common in multi-step validation systems, though only one of the three CVEs turns out to be a true race condition; the other two involve the check simply never being invoked, or being invoked against the wrong file, on certain code paths. Loading a pipeline from the Hub is not a single atomic operation; it involves at least two sequential HTTP requests, one to retrieve configuration metadata and a later one (or a separate code path) to fetch and load the actual pipeline module. The `trust_remote_code` gate lived entirely inside `DiffusionPipeline.download()`, which handles the first phase, while the code that actually imports and executes a Python module — the dynamic module loader — sat downstream and was never revisited by the trust check [1][2][4]. Any technique that caused the loader to see custom code the initial gate had not evaluated therefore bypassed the protection completely, regardless of how the calling application had configured the flag.

Security Analysis

The first and most severe variant, tracked as CVE-2026-44513 (CVSS 8.8, vector `CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H`), affects the `custom_pipeline` argument, a feature that lets a caller point Diffusers at a named pipeline implementation hosted in a Hub repository. GitHub's advisory for the issue describes three distinct exploitation paths within this single CVE: a cross-repository case where the `custom_pipeline` value resolves to code in a different, attacker-controlled repository than the one that passed validation; a local-snapshot case where a locally cached model path bypasses `download()` entirely, so the security gate never runs at all; and a local-custom-components case where Python files present in a local snapshot execute without any validation [4]. All three variants let an attacker's code run under the identity and permissions of whatever process invoked `from_pretrained()` — a CI runner, an inference server, or a data scientist's notebook — while the call site's code showed no indication that anything untrusted had been permitted to execute.

The second variant, CVE-2026-44827, also rated CVSS 8.8, exploited an unrelated but similarly structural quirk: when a caller does not supply a custom pipeline argument at all, Diffusers' loader constructs a literal filename, `None.py`, and checks a separate code path to determine whether that file exists in the target repository. That existence check did not flag `None.py` as untrusted custom code, so a Hub repository containing a file with that exact name could smuggle arbitrary Python past the safeguard using the library's own default, no-argument code path — the no-argument, default-configuration case, which offers no signal to the caller that anything untrusted was permitted to execute [1][5]. The third variant, CVE-2026-45804 (CVSS 7.5), targeted the race window between the two sequential HTTP requests Diffusers issues when resolving a repository, reported at roughly 0.3 seconds. An attacker who controlled the target repository could alter its configuration to reference malicious custom code after the initial validation request had already completed but before the second, execution-triggering request was issued, again producing code execution the trust flag was supposed to prevent [2][6].

Successful exploitation of any of the three variants grants the attacker the same outcome: code execution with the privileges of the process loading the model. In many enterprise deployments, the process loading a model also holds credentials well beyond the model itself — cloud provider tokens, internal network access, source-control credentials, or API keys for adjacent services — so the practical impact extends to credential theft, data exfiltration, tampering with downstream artifacts, and lateral movement into connected infrastructure. Diffusers is not a niche dependency; The Hacker News reported over 8.1 million downloads of the package in July 2026 alone, and Infosecurity Magazine's reporting placed monthly download volume in the range of seven million, noting the library's presence "inside production AI pipelines, CI/CD systems and container images" (the two outlets' figures differ

slightly, likely reflecting different measurement windows, but agree on an order of magnitude in the millions) [1][2]. Any organization that fetches model repositories or custom pipelines from the Hub, whether directly or through a downstream tool that wraps Diffusers, inherited this exposure for as long as it ran a version older than 0.38.0.

It is worth situating FaceHugger within a broader pattern rather than treating it as an isolated library bug. Reporting around the same period covered a separate, unrelated vulnerability in Hugging Face's Transformers library, CVE-2026-4372, in which an attacker could achieve remote code execution by adding an innocuous-looking parameter, `_attn_implementation_internal`, to a remote model's configuration file, again defeating a `trust_remote_code=False` setting; that issue has since been patched in Transformers 5.3.0, released in March 2026 [2][9]. The two libraries implement their trust checks independently and the specific mechanics differ, but both incidents point toward a shared lesson for the AI supply chain. In CSA's assessment, they illustrate that a boolean safety flag is only as strong as the completeness of the code paths it actually gates, and that loaders which fetch code and configuration through multiple, decoupled steps are structurally prone to leaving at least one path unchecked. Organizations that treat `trust_remote_code=False` as a sufficient control, rather than as one layer in a defense-in-depth posture around model provenance, should revisit that assumption in light of both incidents.

Recommendations

Immediate Actions

Organizations running Diffusers in any capacity should upgrade to version 0.38.0 or later without delay, since this is the first release in which the trust check was relocated to the actual dynamic-module load site rather than the earlier download phase [3][4]. Security and platform teams should also inventory every location where Diffusers is pinned — application dependency files, base container images, CI/CD pipeline definitions, and any vendored or forked copies — because a patched top-level application dependency does not help if an older version ships inside a container base image or a third-party wrapper library. Teams that load models via `custom_pipeline` or from local snapshots pulled from the Hub should treat any repository whose provenance is not fully controlled as untrusted until the patched version is confirmed in place, and should audit recent pipeline loads for signs that a repository containing a suspicious file (including one literally named `None.py`) was fetched during the exposure window.

Short-Term Mitigations

Beyond patching, organizations should reduce the blast radius of any future loader-level bypass by running model-loading processes with the least privilege the workload actually requires, isolating credentials for cloud services, source control, and internal APIs away from the process that resolves and imports Hub repositories. Pipelines that fetch models from the Hub as part of CI/CD or automated retraining should pin specific repository revisions rather than tracking mutable references, and should route Hub traffic through an allowlist or internal mirror where feasible so that only vetted repositories can be resolved at load time. Security teams should also extend existing software composition analysis and vulnerability monitoring to explicitly cover AI/ML libraries like Diffusers and Transformers, which are frequently absent from traditional SCA tooling scoped to conventional application dependencies.

Strategic Considerations

FaceHugger and the parallel Transformers issue both illustrate that model-loading libraries are, in effect, package managers for executable code, and should be governed with the same supply chain discipline applied to package registries such as PyPI or npm: provenance verification, version pinning, quarantine windows before adopting new releases, and monitoring for anomalous process behavior at load time. Organizations building or maturing an AI supply chain security program should treat model repositories, custom pipelines, and any dynamically loaded model code as build-tier artifacts requiring inventory, attestation, and change control, rather than assuming that a library-level trust flag provides a durable guarantee. Because these bypasses were discovered through independent security research rather than vendor-initiated review, procurement and vendor-risk processes for AI tooling should also confirm whether suppliers maintain active third-party security research engagement and rapid patch cadences for their model-loading and inference libraries.

CSA Resource Alignment

FaceHugger falls within the AI supply chain risk category CSA has examined in prior analysis of developer-toolchain trust models, which argues that build-tier components — extensions, plugins, and, by direct extension, model-loading libraries — must be governed with the same provenance, version-pinning, and quarantine-window discipline as any other software supply chain artifact rather than left to individual developer judgment. That analysis's core recommendation, treating third-party code paths as build-tier artifacts requiring inventory and attestation, applies directly to the model repositories and

custom pipelines at the center of the Diffusers flaws: a Hub repository is functionally equivalent to a package or extension that an organization is trusting to run with local privileges, and the same discipline recommended for extensions and MCP servers is warranted for model repositories.

The structural root cause behind FaceHugger — a security gate implemented at one point in a multi-step process while the sensitive action occurs at a different, unguarded point — closely mirrors the pattern CSA documented in LiteLLM RCE Chain: AI Gateway Under Active Exploitation, where two independently minor issues combined because validation and execution were not co-located in the request path [7]. Both cases demonstrate that AI infrastructure components frequently decompose a single logical operation into multiple network calls or processing stages, and that any security control confined to one stage should be assumed incomplete until the entire call path has been audited end to end. Organizations remediating FaceHugger should apply that same end-to-end audit discipline to other model-loading and inference tooling in their stack, not just to Diffusers itself.

More broadly, these flaws map to the AI supply chain security and application security domains of CSA's AI Controls Matrix (AICM) v1.1, which calls for verifying the provenance and integrity of third-party AI models and code before execution and for maintaining vulnerability management processes that explicitly cover AI/ML-specific libraries [8]. Programs seeking to operationalize the recommendations in this note should use AICM's supply chain and secure-development control families as the baseline against which model-loading practices, including Diffusers version governance and Hub repository vetting, are assessed.

References

[1] The Hacker News. "Hugging Face Diffusers Flaws Could Let Model Repositories Execute Arbitrary Code." The Hacker News, August 2026.

[2] Infosecurity Magazine. "Bugs in Hugging Face Diffusers Bypass Custom Code Safeguard." Infosecurity Magazine, August 2026.

[3] GitHub Advisory Database. "Diffusers has a trust_remote_code bypass via custom_pipeline and local custom components (CVE-2026-44513)." GitHub, May 2026.

[4] Hugging Face. "Security Advisory: trust_remote_code Bypass in Diffusers (GHSA-98h9-4798-4q5v)." GitHub, May 2026.

[5] Hugging Face. "None.py Trust Remote Code Bypass (CVE-2026-44827)." GitHub, May 2026.

[6] Hugging Face. "TOCTOU Trust Remote Code Bypass (CVE-2026-45804)." GitHub, May 2026.

[7] Cloud Security Alliance. "LiteLLM RCE Chain: AI Gateway Under Active Exploitation." CSA AI Safety Initiative, 2026.

[8] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." Cloud Security Alliance, 2026.

[9] CSO Online. "Hugging Face Transformers RCE Flaw Enables Stealthy Compromise via AI Model Configs." CSO Online, June 2026.

