21 SEP 2026

Three memory-safety bugs in Godot's untrusted-file parsers

Axe Ghost. On Steam!
@axeghostgame.bsky.social

25 6 Share

I found three serious bugs in Godot 4.7 in an audit conducted with an LLM agent. The bugs are due to missing bounds checking in code that reads untrusted files. Using files that are deliberately crafted, Godot can be made to read or write beyond an allowed buffer. I reported the bugs to the Godot security team through email. I provided an example of effective reproduction and suggested a solution for each.

Two of the three bugs have been in Godot since the engine was first open-sourced in 2014. The third is nine years old, introduced in 2017. All three are present, unchanged, in current builds as of this writing (2026-09-21). My audit was not extensive and it turned up other leads which I did not explore.

Working reproduction archives for all three bugs are linked in the correspondence below.

How they were found

I directed the audit by prompting a Claude Code agent, using Opus 4.8, to:

> Find serious bugs in Godot 4.7 that were previously unknown.

The agent then downloaded the Godot source, identified promising areas, found the bugs and created working reproduction projects that demonstrated them, causing crashes against official release binaries. The first working reproduction was ready a couple of hours after the initial prompt. Others followed later that same day.

How these bugs activate

The .hdr bug fires during Godot's automatic import of assets. The .res bug fires when a malicious binary resource is loaded. The translation bug fires later, the first time code looks up a string in a maliciously crafted translation file. Some exported games may be at risk: any game that loads .hdr images, binary resources, or translations it did not ship (e.g. mods, downloads, user uploads).

Severity

The way bugs of this kind have been treated in the past suggests that they warrant serious attention.

Similar buffer bugs were found in 2021 with Godot's TGA decoder - CVE-2021-26825 and CVE-2021-26826. The issues were assigned CVE ids and scored 7.8 on CVSS - qualifying as 'high' severity.

I created reproductions that caused Godot to crash but didn't create any weaponised payloads - that would require significantly more effort.

The bugs

A file format defines numbers intended to hold data describing the file. Those numbers don't have to be true. In the three bugs I found Godot trusts these numbers without checking them against the file's real characteristics.

HDR file loader: .hdr files are special images often used for sky or lightmaps. To keep the files small, they contain special instructions to encode things like 'repeat this color for the next 20 pixels'. If Godot imports a maliciously created .hdr file, and sees instructions to create more pixels of color than it has reserved space for, it will write past the end of the buffer it reserved for the image, into memory it's not supposed to touch with the operation. Godot doesn't check that there's room to carry out the operation. The data written out of bounds comes from the file the attacker crafted.

Binary resource loader: When loading .res files Godot looks for special values in them that are supposed to announce how long a corresponding string inside that file is, so Godot can allocate the right amount of memory to hold it. But an attacker can falsely author a negative integer as a string length. Godot sees the negative value, treats it as effectively zero, and concludes that it's not necessary to allocate space for the string. Then Godot interprets that same value as a huge unsigned int when copying from the file into memory, causing the data to be written unsafely into adjacent areas.

Translation lookup: A crafted translation file can contain an index that points beyond one of Godot's buffers. When code looks up a translated string, Godot reads outside that buffer. This can crash Godot, or make `tr()` return bytes from elsewhere in the process's memory as the translated string.

The fixes

Before trusting any file metadata, check that file lengths or indexes fit within the relevant buffer. Reject the file if there's any mismatch. I included a suggested patch with each report.

Response from the Godot team

On 2026-09-19 Clay John replied to the Godot issue with the following comment.

> Thank you for your interest in contributing to Godot.
>
> Our security team reviewed your emails and determined that there were no security issues. In all three cases, the vulnerability requires the attacker to already have control over your filesystem and feed in specially crafted files to the Godot editor in order to make it write to an out of memory location and crash.
>
> Please go ahead and open a regular bug reports since we do routinely fix memory safety issues as we discover them. There is no need for any special handling.
>
> Please remember to follow the bug report template and note our rules on AI-assisted contributions https://contributing.godotengine.org/en/latest/development/contribution_rules.html#ai-assisted-contributions
>
> Also, as a point of advice. If you expect an answer to your emails, it is best to hand write them so that it is clear they are coming from a person. Using an LLM to assist you is fine, but it was very clear in the emails that the reports and email were generated by an LLM. Our volunteers have to prioritize how they spend their time so it is no surprise they decided not to respond after confirming that the reports were ordinary memory safety issues and not security issues.

These *are* security issues.

John is wrong when he claims that the vulnerabilities require an attacker to already have control over the filesystem. The team seems only to have considered how these bugs affect users running the Godot editor, and not their implications for exported games. An exported Godot game might deliberately accept and parse player/community uploaded files. In a game like this an attacker could use the bugs as memory-corruption/read primitives without having general system control.

In February 2024, Godot lead developer Juan Linietsky wrote about 'real security', that it's "where you care that what you are doing can't become an attack vector for common types of exploits (such as stack or memory overwriting, predictor attacks, etc)" and "on the Godot side, the only real security measures are when handling marshalls, protcols and binary files." Two of the bugs I reported are in code that parses binary files.

The US National Vulnerability Database scored two Godot TGA decoder overflow bugs that received CVEs in 2021. The database recorded that the attacker needed *no privileges on the victim's machine* and the vulnerability needs only for the victim to load the file. Both issues scored 7.8 out of

10 for severity. The three bugs I reported share the same relevant characteristic: they are unsafe memory operations triggerable by maliciously crafted files.

A point of order: My HDR bug report included a note: "This report was drafted by Claude Opus 4.8 and Claude Sonnet 4.6, directed by me." The other two stated: "Security report and reproduction case created by Claude Sonnet 4.6." When John says the emails were "clearly LLM-written" this misleadingly implies that this was surreptitious LLM use uncovered by the team, rather than something I announced clearly myself.

The relevant questions for maintainers should be whether an incoming report is correct, reproducible, and useful. 'Did an AI model write it?' is a terrible proxy for that, *even though* bad AI assisted contributions undoubtedly exist. I've read the justifications, and the problem of overwhelmed maintainers is a strong reason to engage with this technology more thoughtfully:

- Use LLMs to triage reports and PRs, and optionally to coach contributors in improving their contributions. Only contributions that are already filtered, and improved where necessary, reach human maintainers, protecting their time, energy, and enthusiasm.
- Use LLMs to audit the project yourselves so that these kinds of vulnerabilities don't need to be reported to you in the first place.

Update, 2026-10-14

MITRE assigned and published CVE identifiers for all three bugs on 2026-10-14

- Radiance HDR image loader, heap out-of-bounds write: CVE-2026-110734
  https://www.cve.org/CVERecord?id=CVE-2026-110734
- Binary resource loader, heap out-of-bounds write from a signed length converted to unsigned: CVE-2026-110735
  https://www.cve.org/CVERecord?id=CVE-2026-110735
- OptimizedTranslation lookup, out-of-bounds read from an unvalidated offset: CVE-2026-110736
  https://www.cve.org/CVERecord?id=CVE-2026-110736

Each record lists this page and https://github.com/godotengine/godot/issues/123608 among its references. All three records are tagged disputed, and carry a note recording the supplier's position that the research identified no security issues because the required attacker access falls outside the Godot Engine threat model.

The records carry CVSS 3.1 base scores of 2.5, 2.5 and 3.6, all in the Low band. The assigning authority set those scores. I asked the same authority to reconsider all three on 2026-10-14, citing the descriptions in the records and the authority's own 2021 scoring of comparable Godot defects. The three low scores stood when this update was published. You can find the case for interpreting CVSS severity for the three bugs as High here.

Update, 2026-10-23

Godot merged fixes for all three defects on the master branch on 2026-10-20. Maintainer akien-mga wrote the patches, each citing the CVE record for the bug it closed.

- Radiance HDR image loader, CVE-2026-110734
  https://github.com/godotengine/godot/pull/124697
- Binary resource loader, CVE-2026-110735
  https://github.com/godotengine/godot/pull/124699
- OptimizedTranslation lookup, CVE-2026-110736
  https://github.com/godotengine/godot/pull/124701

The fixes went in labelled `bug` and `crash` and queued for cherry-picking to 4.5, 4.6 and 4.7. The HDR pull request restates the supplier's position: "We don't consider it severe, but it's still a bug that should be fixed." For the record my position has not been that these were *severe* bugs, but that they were unambiguously *security* bugs, and ought to have been treated as such.

Timeline

I reported the HDR loader bug to security@godotengine.org on 2026-06-22, with a working reproduction and a fix. The next day I reported the binary resource loader and translation loader bugs to the same address, also with working reproductions and fixes. I stated that I would disclose the bugs publicly after 90 days.

On 2026-09-18 I sent a reminder to security@godotengine.org stating the planned publication date of this document, and an earlier draft of it. On the same day I posted an issue on the Godot engine's GitHub repository saying I'd sent these emails three months ago without reply and would be publishing the disclosure on the 21st.

On 2026-09-19 Clay John replied to the github issue stating that the security team had determined that the bugs were not security issues, and recommended that they be filed as regular bugs. John closed the issue as "not planned."

On 2026-09-20 I requested CVE identifiers for the three bugs from MITRE.

I published this disclosure on 2026-09-21. As of this date Godot 4.7.2-stable and 4.8 development branches still carry the three bugs.

On 2026-10-14 MITRE published CVE identifiers for the three bugs: CVE-2026-110734, CVE-2026-110735 and CVE-2026-110736.

On 2026-10-20 Godot merged fixes for all three bugs on the master branch.

Addendum (2026-09-26): Correspondence

These are the four messages I sent to security@godotengine.org about the three bugs described above. They are reproduced verbatim except where marked. None of the four received a reply from the security team. Before publication, the only response was a public GitHub comment on 19 September 2026 from a project maintainer, which characterized the reports.

Subject: [Security] Heap OOB Write in HDR Image Loader — Godot 4.7 (stable)

Mon 22 Jun, 21:57 CEST

Hello Godot Security Team,

I'm reporting a heap out-of-bounds write in the HDR image loader, verified on the current stable release. Reproduction files are linked at the end of this email.

Planned public disclosure: 2026-09-20 (90 days). Happy to coordinate an earlier date if a patch ships sooner, or discuss a brief extension if needed.

PART 1. VULNERABILITY REPORT

FILE AND FUNCTION

modules/hdr/image_loader_hdr.cpp, lines 104-118 ImageLoaderHDR::load_image

CONFIRMED AFFECTED VERSIONS

v4.7.stable.official.5b4e0cb0f (directly tested) v4.7.rc2.official.3df26a02c (directly tested)

ROOT CAUSE

The loader allocates a single contiguous buffer for the entire image: imgdata.resize(height * width * 4) at line 65. For this PoC: 16 rows x 8 pixels x 4 bytes = 512 bytes total.

Channel bytes are written directly into this buffer — there is no intermediate per-channel temp scanline buffer. (temp_read_data at line 71 is a 128-byte file-read staging area only.) Each channel k writes to interleaved positions in imgdata via ptr[(j * width + i++) * 4 + k] at lines 111 and 117.

The inner loop checks (i < width) once before reading each opcode, not after the writes inside the body. The comparison at line 106 is strictly greater-than: if (count > 128). A Dump opcode of 0x80 = 128 is not strictly greater than 128, so the Dump branch is taken and i advances by count = 128. On the last scanline (j=15), a Dump of count=128 starting at i=0 writes to pixel positions (j × width) + 0 through (j × width) +127, covering byte offsets 480-991 in imgdata. Positions i=0..7 (offsets 480-511) are within the 512-byte allocation; i=8..127 (offsets 512-991) are past the end.

Triggering this on all four channels covers offsets 512-991 — 480 bytes of adjacent heap — with fully attacker-controlled content.

SUGGESTED FIX

The fix is to bound the advance against the remaining scanline width before each body — not to change the opcode comparison. Changing > 128 to >= 128 would incorrectly reject count=128 as a dump, which is a legitimate value. The correct fix adds a check in both branches:

Run branch, after count -= 128: ERR_FAIL_COND_V_MSG(count > width - i, ERR_FILE_CORRUPT, "Corrupt HDR RLE run.");

Dump branch, before get_buffer: ERR_FAIL_COND_V_MSG(count > width - i, ERR_FILE_CORRUPT, "Corrupt HDR RLE dump.");

This fails loudly on malformed input and leaves valid files unaffected.

CRAFTED FILE STRUCTURE (evil4ch.hdr, 746 bytes)

[crafted-file byte layout withheld — available on request]

REPRODUCTION

Unzip the reproduction archive and run from PowerShell:

.\verify.ps1 -Godot "<path-to-Godot_v4.7-stable_win64.exe>"

Expected output:

verify.ps1 runs both files through repro.gd (a four-line GDScript that calls Image.new().load(path)) and compares exit codes. 0xC0000374 is STATUS_HEAP_CORRUPTION — the Windows kernel exception code for heap integrity failure. The process prints completion before crashing; the OOB writes occur inside load_image() and the exit code is consistent with the heap check firing when imgdata is freed during teardown.

SECURITY IMPACT

The payload bytes are fully attacker-controlled. While this PoC results in a crash, an OOB heap write with controlled content is a standard primitive for heap exploitation and carries RCE potential. The most immediate attack surface is editor auto-import: a malicious .hdr in a shared project, asset pack, or Asset Library submission crashes the editor on project open. Games loading user-supplied HDR files at runtime face a player-facing attack surface.

PART 2. HOW THIS WAS FOUND, AND A RECOMMENDATION

Note on authorship: This report was drafted by Claude Opus 4.8 and Claude Sonnet 4.6, directed by me. I verified the crash directly - I ran the reproduction script and confirmed the STATUS_HEAP_CORRUPTION exit code against the clean baseline. The technical details of the root

cause and the fix are beyond what I can personally validate, but I trust the tooling enough to be convinced the issue is real and worth your attention.

This vulnerability was discovered through an autonomous LLM-driven code audit using Claude Code that took about four hours. The discovery phase used Claude Opus 4.8, which downloaded the Godot 4.7 source, identified the HDR loader as a promising attack surface, and traced the inner-loop bounds logic. The verification phase - crafting the PoC, running the stock binary, confirming STATUS_HEAP_CORRUPTION - used Claude Sonnet 4.6 after a safety-policy flag triggered on the Opus session. The initiating prompt was as follows:

"I want an autonomous LLM flow to discover a serious and previously unknown bug in Godot 4.7. The bug that's found is preferably something that could result in lost work, or a security vulnerability, either in the editor or in a runtime exported from it. I will need simple instructions for how I can verify any bug found, for myself. It will be your responsibility to download the source, investigate promising areas, test anything that looks like a bug, then once found, write up instructions for me on how to replicate it."

This HDR bug was one of 27 flagged leads the session produced. [details of the two other flagged leads withheld — shared privately on request] I'm happy to share the full list privately if that would be useful to your team.

Given that a single session with no prior codebase knowledge produced a confirmed heap OOB write in a stable release, I'd encourage the Godot security team to consider running their own LLM-assisted audit across the codebase - and particularly across the remaining leads - as part of the release process.

REPRODUCTION FILES

https://drive.google.com/file/d/1JvgN0Tobio83skfalRJGYzSL08__hyKC/view?usp=sharing

Archive contains: evil4ch.hdr, ok.hdr, repro.gd, verify.ps1, project.godot.

Please confirm receipt and let me know your expected patch timeline.

Best regards, Tomasz

Subject: [Security] Heap OOB write in `ResourceLoaderBinary::get_unicode_string` — Godot 4.7

Tue 23 Jun, 20:38 CEST

Hello Godot security team,

I am reporting a heap out-of-bounds write in Godot 4.7's binary resource loader.

This bug was discovered by Claude Opus 4.8 during an audit of the Godot source code. Security report and reproduction case created by Claude Sonnet 4.6. Verified by Tomasz Kaye, who confirmed that the bug crashes the Godot editor as described. Tomasz has not independently validated the further security implications (attacker-controlled write primitive, RCE potential) that the models warn about - those assessments are the models' own analysis and should be reviewed by the Godot security team.

I intend to follow a 90-day disclosure norm: I will not publish details publicly until a fix is available or 90 days have passed from the date of this report, whichever comes first. Please let me know if you need additional time or any further information.

What follows is Sonnet 4.6's report.

Summary

`ResourceLoaderBinary::get_unicode_string()` (`core/io/resource_format_binary.cpp`, line 864) reads a 32-bit string length into a **signed** `int` with no masking. A crafted `.res` file with a length field of `0x80000000` exploits this in three steps:

1. The signed comparison `len > str_buf.size()` is false (`-2147483648 > 16` is false), so the heap buffer is **not** grown.
2. The zero-check `len == 0` is also false, so execution continues.
3. `FileAccess::get_buffer(uint8_t *, uint64_t)` receives the negative `int` as a `uint64_t` argument, where it sign-extends to `0xFFFFFFFF80000000`. The underlying `fread()` call uses this as its byte count with no destination-size clamp, writing the entire file tail into the under-sized heap allocation.

Both the overflow length and the written bytes are fully attacker-controlled. The adjacent `_get_string()` helper already masks the same field with `& 0x7FFFFFFF`; `get_unicode_string()` does not.

The static `get_ustring()` helper (line 856) has the identical pattern and should be hardened at the same time.

Affected Version

Godot 4.7 (official stable build). The pattern is present in earlier 4.x releases.

Severity

High — attacker-controlled heap OOB write, reachable from an untrusted `.res` file with no user interaction beyond opening a project in the editor.

Not rated critical because the write is sequential from a fixed file buffer (not an arbitrary write-what-where primitive) and reliable RCE depends on heap layout; the common stock-build outcome is heap corruption or a `STATUS_HEAP_CORRUPTION (0xC0000374)` crash.

Trigger Paths

Trigger A — Editor filesystem scan

Copy `evil.res` into a Godot project folder **before** opening it in the editor. `EditorFileSystem` scans resource dependencies on startup, calling:

The editor crashes on startup with `STATUS_HEAP_CORRUPTION`. No script, no explicit load, no user click required.

Trigger B — Headless runtime load

Exit code: `0xC0000374` (`STATUS_HEAP_CORRUPTION`) on Godot 4.7-stable Windows.

Reproduction Package

Reproduction package: `godot47_binres_oob_repro.zip`
Download: https://drive.google.com/file/d/1GdgzHu4YFGx1DYjLWz7CuqMBfNQcjC_n/view?usp=sharing

Contents:

`README.md` Step-by-step repro instructions

`bug_analysis.md` Full technical write-up

`make_repro.py` Python 3 script to regenerate `evil.res`

`proj/project.godot` Minimal Godot 4.7 project

`proj/test.gd` Trigger B script

`proj/evil.res` Pre-generated 8 MiB malicious resource

To regenerate a smaller file for use under ASan or Windows PageHeap:

Under ASan/PageHeap, 4 KiB is sufficient for an immediate `heap-buffer-overflow WRITE` report at `resource_format_binary.cpp:872`.

Root Cause

Suggested Fix

Mirror the masking already used in `_get_string()`:

Defense in depth: also bound `len` against `f->get_length() - f->get_position()` and apply the same fix to `get_ustring()` (line 856).

Regards, Tomasz

Subject: [Security] Heap OOB read in `OptimizedTranslation::get_message` — Godot 4.7

Tue 23 Jun, 21:24 CEST

Hello Godot security team,

I am reporting a heap out-of-bounds read in Godot 4.7's translation system.

This bug was discovered by Claude Opus 4.8 during an audit of the Godot source code. Security report and reproduction case created by Claude Sonnet 4.6. Verified by Tomasz Kaye, who confirmed that the bug crashes the Godot editor as described. Tomasz has not independently validated the further security implications (info-leak primitive, heap byte exfiltration via `tr()`) that the models warn about — those assessments are the models' own analysis and should be reviewed by the Godot security team.

I intend to follow a 90-day disclosure norm: I will not publish details publicly until a fix is available or 90 days have passed from the date of this report, whichever comes first. Please let me know if you need additional time or any further information.

What follows is Sonnet 4.6's summary of the bug.

Summary

`OptimizedTranslation::get_message()` (`core/string/optimized_translation.cpp`, line 259) uses a `uint32_t` value read verbatim from the resource's `hash_table` array as a direct index into `bucket_table` with no bounds check. The only guard rejects the sentinel `0xFFFFFFFF`; any other out-of-range value proceeds unchecked, forming a pointer billions of bytes past the heap allocation. Two exploitable primitives result:

1. **Crash (Variant 1):** `hash_table[0] = 0x40000000` causes `&btptr[0x40000000]` — a 4 GiB offset into an 8-byte allocation — and the immediate dereference of `bucket.func` at line 261 faults on unmapped memory. The crash is deterministic on any 64-bit stock binary.
2. **Info-leak (Variant 2):** A crafted bucket at `p = 0` with an `str_offset` past the end of the `strings` array causes `get_message()` to return `String::utf8(&sptr[str_offset], len)` — reading attacker-chosen bytes from heap memory beyond the allocation and returning them as the `tr()` result, which reaches GDScript callers and potentially game UI, logs, or network output.

The identical pattern appears in `get_translated_message_list()` at line 299.

All three backing arrays (`hash_table`, `bucket_table`, `strings`) are written verbatim from the resource file by `_set()` (lines 202–217) with zero validation, making any loaded `.translation` or `.tres` file a full control surface.

Affected Version

Godot 4.7 (official stable build). The pattern is present in earlier 4.x releases.

Severity

Medium–High — two usable primitives from a single crafted translation resource:

- *Crash*: reliable on any stock 64-bit binary, editor and export template.
- *Info-leak*: `tr()` return value is attacker-readable; no memory instrumentation required to construct the payload, though confirming the leak on a victim machine without tooling requires observing the translated string value.

The realistic attack surface is supply-chain substitution of `.translation` files (the output of the CSV importer) or malicious asset distribution. Both reach the bug with no victim action beyond running a game or opening a project.

Trigger Paths

Auto-load path (no explicit load in GDScript)

`TranslationServer::load_project_translations()` (`translation_server.cpp:758`) calls `ResourceLoader::load(path)` followed by `add_translation()` for every path listed in the project setting `internationalization/locale/translations`, with no validation of the loaded object's internal arrays. Placing a crafted `.translation` file there and making any `tr()` call triggers the bug with no further user action.

Editor path (supply-chain)

The CSV translation importer (`editor/import/resource_importer_csv_translation.cpp`) produces `OptimizedTranslation` resources saved as `.translation` files. Substituting a project's `.translation` file with a crafted one is the most realistic supply-chain vector.

Explicit load path

Exit code: access violation / SIGSEGV on Godot 4.7-stable Windows (Variant 1).

Stock binary confirmation — Variant 2 (info-leak), no instrumentation

[transcript withheld — available on request. Summary: Variant 2 (info-leak) was confirmed on a stock binary with no instrumentation; `tr()` returned heap memory, not resource content.]

Reproduction Package

Reproduction package: `godot47_optimized_translation_oob_repro.zip`
https://drive.google.com/file/d/1PMewCT6wDOdyS-hF-I_H0yQMCizO0_sI/view?usp=sharing

Contents:

`README.md` Step-by-step repro instructions
`bug_analysis.md` Full technical write-up

`make_repro.py` Python 3 script to regenerate all project files
`proj/project.godot` Minimal Godot 4.7 project
`proj/main.tscn` Empty main scene (required for the project to run)
`proj/trigger.gd` Autoload that loads the translation and calls `tr()`
`proj/bad_translation.tres` Crafted `OptimizedTranslation` — crash payload
`proj/leak_translation.tres` Crafted `OptimizedTranslation` — info-leak payload

To regenerate:

For Variant 2 (info-leak), edit `proj/trigger.gd` and set `VARIANT = "leak"` on line 4, then re-run the project. Under AddressSanitizer the report points to `optimized_translation.cpp:277` with a `heap-buffer-overflow READ`.

Root Cause

`_set()` loads all three arrays verbatim:

No validation of logical consistency (index ranges, size relationships) occurs at load time.

Suggested Fix

Add bounds validation before the unchecked dereference in both `get_message()` and `get_translated_message_list()`:

Additionally, guard each `sptr` read before using `str_offset`:

Apply both guards to `get_translated_message_list()` at line 299 identically.

Defense in depth: `_set()` could validate logical consistency of the three arrays at load time rather than deferring to `get_message()`.

Regards, Tomasz

Subject: Final notice before public disclosure on 2026-09-21: three memory-safety bug reports (Godot 4.7)

Fri 18 Sep, 14:27 CEST

Hello,

On June 22 and 23 I sent three security reports to this address:

1. 2026-06-22, 21:57 CEST: heap out-of-bounds write in the HDR image loader
2. 2026-06-23, 20:38 CEST: heap out-of-bounds write in ResourceLoaderBinary::get_unicode_string
3. 2026-06-23, 21:24 CEST: heap out-of-bounds read in OptimizedTranslation::get_message

Each report included a working reproduction and a suggested fix. I have not received a reply to any of them.

The 90-day disclosure windows I stated in those reports close on 2026-09-20 (report 1) and 2026-09-21 (reports 2 and 3). I plan to publish a write-up of all three bugs on 2026-09-21.

If the team is actively working on fixes and needs more time, please reply before 2026-09-21. If the reports never reached you, let me know and I will resend them.

Below is the draft of the text I intend to publish, so there are no surprises about scope or tone.

Regards, Tomasz Kaye

security godot ai-use

---------- DRAFT FOR PUBLICATION ON 2026-09-21 ----------

[the publication draft that followed here is omitted — it is an earlier draft of the text published above on 2026-09-21]

Enjoyed this article?

Join our free newsletter and never miss an update.

Subscribe with Bluesky

@ alice.bsky.social Subscribe

CVSS scoring for Godot vulnerabilities. CVE-202... ›

Related Articles

An attacker can crash a Godot game server using DTLS with a single UDP datagram

DTLS is the engine's optional encryption layer for UDP multiplayer. In every version of Godot I inspected the source of — 3.2.2 and 4.7...

27 OCT 2026

♡ 1

A malicious game server can crash the Godot clients that join it

Joining a multiplayer server is an ordinary player action. In Godot 4.0 and later, in games using the engine's standard spawning setup, a...

27 OCT 2026

2 1

A community translation can run attacker-chosen code in an exported Godot game

A .po translation file is meant to be data. In Godot 4.1 and later, part of it is executed as code (the chain is present in the engine source...

26 OCT 2026

46 2

axeghost

RSS Terms Privacy Cookies

© 2026. All rights reserved.

Published with Offprint

axeghost.offprint.app