04 NOV 2026

Any networked co-player can act as the server in a GodotSteam game

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

15 1 2 Share

GodotSteam is a third-party plugin that connects Godot games to Steamworks; the maintainer's opt-in games list names 475 Steam games/apps that use it. The plugin's multiplayer code identifies every connected player by a numeric peer ID, and the first packet on a new connection is four bytes containing the sender's chosen ID. The plugin records that number without checking it. Godot treats peer ID 1 as the server, so any other player in the session can send a packet containing 1 and be recorded as the server on the victim's machine. The finding was source-level, verified against the plugin's v4.23.2 source and the Godot 4.7 engine source. A runtime demonstration on two machines followed the same day; described at the end of this page.

The sender chooses the peer ID; the plugin checks only its length

A connection starts with no peer ID. Until one exists, any four-byte packet arriving on that connection is read as the handshake (`godotsteam_multiplayer_peer.cpp:134`); the only validation is the length (`steam_packet_peer.cpp:147`). The whole handshake is one unsigned integer: `struct PeerIDPacket { uint32_t peer_id = 0; }` (`steam_packet_peer.h:81`). The plugin stores the claimed value as the connection's user data (`steam_packet_peer.cpp:150`, definition at `:75`) and reports it as the sender of every later packet (`godotsteam_multiplayer_peer.cpp:75`). When the connection completes, the plugin adds the peer to its peer table, the map from ID to connection, under the claimed number (`godotsteam_multiplayer_peer.cpp:565`), never checked for range, collision, or ownership.

Steam's relay proves which account owns each connection; the peer ID travels inside the game's own packets, where the sender fills it in. Godot's own ENet transport validates the claimed ID when a peer connects. The code comment beside the check reads "Client joined with invalid ID, probably trying to exploit us." (`modules/enet/enet_multiplayer_peer.cpp:207`). The Steam transport performs no such validation.

Every engine authority check trusts the claimed ID

Godot's high-level multiplayer treats peer 1 as the server, and every node's authority defaults to 1. The engine accepts a command only from the peer whose ID matches the node's authority. The identity it checks is the ID the transport reports. The RPC check is `scene_rpc_interface.cpp:259`; spawn and despawn checks use the same reported ID (`scene_replication_interface.cpp:576`, `:672`), as does state synchronization (`:876`). A player recorded with ID 1 passes all of them, on any node still using the default authority. These checks hold when the transport reports senders truthfully; this plugin reports a value the sender chose.

Forged packets address nodes and methods by numeric IDs. Those come from owning a copy of the game (public `.pck` extraction tooling) or from observing traffic that a lobby member already receives.

The attacker uses the same packet against a player or a server

1. **Against a player.** A co-player claims 1; the victim's peer table now maps the server's entry to the attacker. Packets the victim sends to the server are delivered to the attacker, and packets the attacker sends are executed by the victim as if they came from the server. A client opens a listening socket in every mode and accepts every incoming connection (`godotsteam_multiplayer_peer.cpp:519`, `:528`).
2. **Against a host or dedicated server.** An ordinary client claims 1 on its own connection. `create_host` gives the host ID 1 but never adds the host to the peer table (`:498`), so the ID-1 entry is empty and the attacker's claim fills it. The server executes the attacker's server-only commands and accepts its spawn and state-sync traffic; changes to replicated state reach clients through the server's ordinary replication and visibility rules.

Games whose multiplayer runs over SteamMultiplayerPeer are affected

Only the multiplayer transport is affected; not achievements, input, or leaderboards, and not ENet or WebSocket transports. The claim works in any session, friends-only lobbies included; sessions where strangers play together, in public lobbies, matchmaking, or open servers, are the riskiest. GodotSteam's own games list names 475 Steam apps. A store-category pass over that list finds 135 with multiplayer categories; 80 of them have the PvP category flag (the situation where attacks would be most likely). I checked one of them, a game with several thousand reviews: it calls the affected lobby functions. I have not tried the attack against it.

Mitigation

Developers can detect an invalid ID-1 claim and close the session with the plugin's own APIs: Steam verifies the account behind each connection, while the peer ID is only the sender's claim. The plugin emits `peer_connected` after installing each peer, including after an overwrite of the server's entry, so the guard below runs immediately after admission:

```
extends Node

var lobby_id: int = 0 # set after Steam.createLobby / Steam.joinLobby succeeds

# Direct create_client connections: set this to the Steam ID you passed to
# create_client. Lobby flows can leave it 0; the guard then reads the lobby
# owner, and connect_to_lobby itself refuses a lobby with no owner.
var expected_server_steam_id: int = 0

func _ready() -> void:
    multiplayer.peer_connected.connect(_on_peer_connected)

func _on_peer_connected(id: int) -> void:
    if id != 1:
        return
    var peer := multiplayer.multiplayer_peer as SteamMultiplayerPeer
    if peer == null:
        return
    if multiplayer.is_server():
        # The host is peer 1; a remote peer arriving as 1 is an attacker.
        _fail_closed()
        return
    var expected := expected_server_steam_id
    if expected == 0 and lobby_id != 0:
        expected = Steam.getLobbyOwner(lobby_id)
    if expected == 0 or peer.get_steam_id_for_peer_id(1) != expected:
        _fail_closed()
```

```
func _fail_closed() -> void:
    push_error("Peer-ID guard: a remote peer claimed the server's ID (1).")
    multiplayer.multiplayer_peer.close()
    multiplayer.multiplayer_peer = null
    # Return the player to your menu / offline state here.
```

This guard was reviewed against the plugin (v4.23.2) and engine (4.7) source in two independent review rounds; it has not been tested in a running game. It guards the server slot (peer ID 1 under default authority) and closes the session on detection, because the plugin exposes no per-peer disconnect. Dedicated servers on the main plugin's `create_host` are in scope; the separate `godotsteam-server` binding has no multiplayer classes.

Verified on two machines

I verified the exploit on two computers on the same network, with two different Steam accounts. The host ran an exported game built with stock Godot 4.7.2 and stock GodotSteam 4.23.2 and called `create_host`. The attacker ran a small client that uses only the plugin's public API. That client opened a connection and sent the four-byte handshake claiming peer ID 1; the host's log recorded `peer_connected: 1`. The client then registered the game's root node path under the claimed peer 1 and sent a forged command packet for that node. The host executed it: `RECEIVED grant_gold(1000000) from peer 1 (SERVER)`, and its gold counter changed to 1000000. The host never issued that command — its own button produces `SERVER ISSUED grant_gold(50)` first.

Two details belong with any reproduction. Both processes must enable Steam's callback dispatch (`Steam.steamInit(0, true)`); without it the connection never completes and the host never sees it. The attacker must also open a listen socket in symmetric-connect mode before dialing, matching how the plugin itself dials (`godotsteam_multiplayer_peer.cpp:540`).

The reproduction kit is at https://github.com/bitbutter/godot-security-pocs/tree/main/2026-11-godotsteam-peer-id-claim.

I assess the severity at CVSS 3.1 8.6 (High)

Vector: `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:H/A:L`. This is my own assessment. With lobby membership scored as a required privilege, the vector is 7.6. Dropping the game-dependent disclosure path gives 8.2. Every variant stays High.

- **Attack Vector: Network** (spec 2.1.1). Packets reach the victim over the internet through Steam's relay; any Steam account able to join the lobby or the server can send them.

- **Attack Complexity: Low** (spec 2.1.2). One deterministic packet on the attacker's own connection.
- **Privileges Required: None** (spec 2.1.3). Joining a lobby with a free Steam account gives the attacker no privilege on the victim's machine.
- **User Interaction: None** (spec 2.1.4). The victim is playing the game.
- **Scope: Unchanged** (spec 2.2). The impact stays inside the victim's game process.
- **Confidentiality: Low** (spec 2.3.1). Traffic the game addresses to the server reaches the attacker once the slot is stolen; the primitive reads nothing by itself.
- **Integrity: High** (spec 2.3.2). Arbitrary control of networked game state: authority-gated RPCs, spawns, despawns, and state synchronization, all verified in engine source, and the RPC path demonstrated at runtime.
- **Availability: Low** (spec 2.3.3). Session-ending state corruption; no crash demonstrated.

Every version I checked is unpatched

The affected classes arrived at v4.17 (2025-12-10) and carry the defect through v4.23.2 (2026-10-16, current as of my check on 2026-11-03). The retired standalone addon (`godotsteam/multiplayerpeer`) is affected from at least v4.11-mp (2024-09-27) through v4.16.2 (2025-11-10, its last tag). My report suggested the validation ENet already applies (reject a claimed ID that is invalid or already taken), plus server-assigned IDs as the structural fix. The v4.23.2 "poisoned client" fix only unregisters callbacks in `_close()`; the handshake is byte-identical. My 2026-11-03 check found no newer tag; the single commit since v4.23.2 (2026-10-28) touches none of the four handshake files.

The acknowledgment window closed without a reply

I reported privately to security@godotsteam.com on 2026-10-21. The finding was the result of an LLM-directed audit, and the report was drafted by an LLM. Godotsteam has a 'No "AI" Policy'**.** My email offered to hold publication for up to 90 days if a fix was in progress, and asked for an acknowledgment within 14 days. No acknowledgment arrived by publication.

Timeline

- 2026-10-21: reported privately to security@godotsteam.com.
- 2026-11-04: the 14-day acknowledgment window ends; no reply received.
- 2026-11-04: disclosure published.

Credit

The vulnerability was found and verified during an LLM-directed source audit of GodotSteam v4.23.2 and the Godot 4.7 engine source. Prepared with LLM assistance. Approved, reported, and disclosed by Tomasz Kaye.

Enjoyed this article?

Join our free newsletter and never miss an update.

Subscribe with Bluesky

@ alice.bsky.social Subscribe

‹ I do not have a line-level understanding of the...

Related Articles

I do not have a line-level understanding of the code in the bugs I report

The six Godot security bug reports I sent so far all disclosed that they were prepared with LLM assistance and contain...

28 OCT 2026

♡ 18 💬 5

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...

27 OCT 2026

♡ 4 💬 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...

27 OCT 2026

5 1

© 2026. All rights reserved.

Published with Offprint