LiteBans tasks

Task guides and troubleshooting for LiteBans. Check the installed release before changing configuration or data.

- Verify a LiteBans temporary ban expires at the intended time
- Test whether a LiteBans mute covers a private-message command
- Build a LiteBans appeal-review role without punishment access
- Check a LiteBans moderator’s maximum punishment duration
- Find which plugin handled a ban command on a mixed-plugin server
- Verify the server scope of a LiteBans network punishment
- Investigate a LiteBans database connection failure without losing history
- Reverse one appealed LiteBans punishment while preserving the audit trail

Verify a LiteBans temporary ban expires at the intended time

A temporary ban should end at the recorded expiry without staff manually clearing it. A short controlled trial catches duration mistakes before they affect a real appeal.

Follow the procedure

1. Identify a disposable player account and record its UUID. Check `/checkban PLAYER` first so the trial cannot replace an existing disciplinary restriction.
2. Read the duration formats supported by your installed release. Use minutes for this test; a month and a minute have different suffixes and should never be inferred from appearance.
3. Issue `/tempban PLAYER 2m Duration verification` from an authorized staff account. In the server console, omit the leading slash. Record the punishment ID and displayed end time.
4. Try joining before expiry and retain the refusal message privately. Confirm the displayed reason belongs to this trial rather than a proxy or another ban plugin.
5. Wait beyond the recorded expiry, then repeat `/checkban PLAYER` and join through the normal public address. Check history separately to confirm the expired record is retained.
6. If testing ends early, reverse this specific trial by its punishment ID using the documented unban syntax. Record that staff reversal separately from a natural expiry.

Check the result

The account is blocked during the trial and allowed afterward, while its historical record remains visible to authorized staff. Compare the server time, displayed expiry and actual successful connection rather than assuming a delayed client retry means the ban still exists.

If the result is wrong

If the account remains blocked, inspect the active punishment and its server scope before changing anything. Another account-based or IP restriction can survive this expiry. Do not prune history or reset the database to remove a single trial; reverse only the identified record.

References and related help

- LiteBans maintainer reference
- LiteBans maintainer reference
- Configure LiteBans moderation and preserve punishment history
- Connect a Minecraft plugin to a MySQL database

Test whether a LiteBans mute covers a private-message command

A mute may stop public chat while another plugin still accepts private messages. Test the exact command route your players use instead of treating every chat-like feature as equivalent.

Follow the procedure

1. List the public and private message commands available on this server, including namespaced aliases and proxy commands. Identify which plugin handles each route.
2. Prepare two consenting test accounts without mute bypass permissions. Check the sender’s existing mute status and save the relevant LiteBans configuration before editing its command controls.
3. Apply a short `/tempmute PLAYER 2m Message-route trial` to the sender. Record the punishment ID, scope and expiry; keep the recipient account unrestricted.
4. Try ordinary public chat, then the actual private-message command and its documented aliases. Record whether the server rejects the command or the recipient receives the text.
5. If an unintended route remains open, use the installed release’s documented muted-command configuration to cover that specific route. Check proxy-owned commands in the proxy configuration too.
6. Repeat the same messages after applying the supported configuration change and after the mute expires. Confirm normal private messaging returns without additional staff intervention.

Check the result

During the mute, each command covered by your moderation policy should be rejected and the recipient should receive no message through those routes. After expiry, both public chat and the approved private-message command should work. A blocked public message alone is insufficient evidence.

If the result is wrong

If the test still leaks messages, inspect command ownership and aliases before adding broader restrictions. A backend cannot necessarily block a command intercepted by the proxy first. Remove only the trial mute when finished and restore the specific configuration edit if it blocks unrelated player commands.

References and related help

- LiteBans maintainer reference
- LiteBans maintainer reference
- Configure LiteBans moderation and preserve punishment history
- Connect a Minecraft plugin to a MySQL database

Build a LiteBans appeal-review role without punishment access

An appeal reviewer often needs punishment history but should not be able to ban, delete evidence or inspect players' IP history. Separate those capabilities before assigning the role.

Follow the procedure

1. Choose the exact records reviewers need: a player's history, current ban status and perhaps the staff executor's history. Keep account linkage and IP investigations outside this role.
2. Look up each command's permission in the installed LiteBans permissions reference. Grant the specific read permissions through your permission manager rather than assigning a broad administrative node.
3. Use a non-operator test reviewer with only the proposed group. Query `/history PLAYER` and `/checkban PLAYER` for an approved trial record, then compare the output with an administrator's view.
4. Try a harmless permission-denial check for a punishment or history-deletion command without supplying a real player target. Confirm the reviewer cannot invoke the protected operation.
5. Check whether staff-history and notification permissions expose more information than expected. Limit silent notifications and avoid granting IP-history access simply to make one command work.
6. Write down the review procedure: collect the punishment ID, reason, executor and timestamps privately, then pass any reversal request to the authorized moderation role.

Check the result

The reviewer can inspect the approved evidence and cannot create punishments, erase records or retrieve unnecessary network identifiers. Verify permissions with the actual reviewer account, since an administrator's successful history query does not demonstrate a restricted role works.

If the result is wrong

If a history command is denied, identify its documented permission and inherited context rather than adding a wildcard. If sensitive information appears unexpectedly, remove the new grant immediately and retest a clean session. Keep the underlying punishment history intact while correcting the role.

References and related help

- LiteBans maintainer reference
- LiteBans maintainer reference
- Configure LiteBans moderation and preserve punishment history
- Connect a Minecraft plugin to a MySQL database

Check a LiteBans moderator’s maximum punishment duration

A junior moderator’s temporary-ban permission should not accidentally allow permanent or excessive bans. Check the release-specific duration limits with a short authorized test before granting the role widely.

Follow the procedure

1. Read the maximum-duration permission section for your installed LiteBans build and record the intended limit in ordinary language. Distinguish the right to issue a command from the limit on its duration.
2. Inspect the moderator group’s inherited permissions for unrestricted ban access, permanent-punishment rights or competing limit entries. A restrictive-looking direct node may not describe the effective result.
3. Create a non-operator test moderator and a separate disposable target. Ensure the target has no active punishment and that a senior administrator can reverse this trial.
4. Apply the documented limit permission, then try one short duration within policy. Record the accepted duration and reverse that specific trial punishment after checking its output.
5. Test a duration above the policy limit using the same disposable target. Confirm the command is refused or constrained exactly as the installed reference describes; do not assume it silently clamps the value.
6. Check permanent punishment and other punishment types separately. A temporary-ban duration limit does not automatically establish the policy for mutes, warnings or IP-based restrictions.

Check the result

The restricted moderator can perform the approved short action but cannot exceed the chosen policy or issue an unintended permanent restriction. Save the effective permission result and command response as a staff training example without publishing the target’s identifiers.

If the result is wrong

If the upper-limit test succeeds unexpectedly, remove the trial role’s punishment access while investigating inherited permissions. Do not experiment on a real disciplinary case. Reverse only trial records, then use the exact limit syntax from the matching release before testing again.

References and related help

- LiteBans maintainer reference
- LiteBans maintainer reference
- Configure LiteBans moderation and preserve punishment history
- Connect a Minecraft plugin to a MySQL database

Find which plugin handled a ban command on a mixed-plugin server

When multiple moderation plugins register the same command, a successful-looking response can update the wrong punishment system. Identify the command owner before correcting any player record.

Follow the procedure

1. Record the exact command, execution location and response that caused confusion. A command entered through Velocity may be handled before a Paper backend sees it.
2. Inspect the installed plugin lists on the proxy and relevant backend. Compare command aliases in their maintainer documentation and the server’s command registration information.
3. Query LiteBans directly using a read-only status or history command for the affected account. Compare the record, executor, timestamp and punishment ID with the message staff originally received.
4. On a disposable account, use the documented plugin-qualified command where your platform supports it. Confirm the selected namespace actually resolves before issuing any trial punishment.
5. Review custom command aliases and command-routing plugins. Adjust only the ambiguous alias or staff procedure, keeping the original punishment records unchanged during diagnosis.
6. Repeat an approved trial through the same public route staff normally use. Record the correct command form in the moderation handbook, including whether it runs on the proxy or backend.

Check the result

A staff command should create a record in the intended system, and the corresponding read-only query should identify that same action. The public join result and the record must agree. A console

response alone cannot establish which database was changed.

If the result is wrong

If two systems contain active restrictions, have an authorized moderator review each record independently. Do not remove both or disable a moderation plugin merely to permit a login. Restore the old alias if routing changes affect unrelated commands and retain the diagnostic timestamps.

References and related help

- LiteBans maintainer reference
- LiteBans maintainer reference
- Configure LiteBans moderation and preserve punishment history
- Connect a Minecraft plugin to a MySQL database

Verify the server scope of a LiteBans network punishment

A local punishment and a network-wide punishment can produce different results when players change backends. Confirm the scope explicitly so staff do not overreach or leave an unintended bypass.

Follow the procedure

1. List the backend names shown by `/litebans servers` and identify the public connection path. Use the names LiteBans recognizes rather than a hosting-panel label that may differ.
2. Choose a test account and inspect existing bans or mutes before starting. Record which backends should enforce this trial and which should remain usable.
3. Read the Server Scopes documentation linked from the commands reference. Confirm the installed configuration’s default scope and the permission required to choose a different scope.
4. Issue an approved short trial with an explicit `server:NAME` argument using a recognized backend name. Record the resulting punishment ID and reported scope.
5. Test through the proxy on the selected backend and a second backend. Repeat the relevant action, such as joining or sending chat, rather than relying only on a server-switch message.
6. Reverse the specific trial and verify normal access returns on both backends. Document when staff should choose local scope and when a separately authorized global action is appropriate.

Check the result

Only the intended backend scope should enforce the local trial, and the punishment record should display the expected scope. A global trial requires its own approval and check. Shared database connectivity by itself does not prove all punishments are network-wide.

If the result is wrong

If scope results differ from policy, inspect server names, plugin placement and shared database configuration. Avoid broad global bans as a diagnostic shortcut. End the trial by its exact record and keep the observed backend results before adjusting default scopes.

References and related help

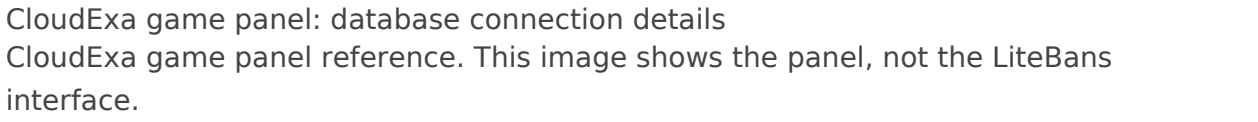
- LiteBans maintainer reference
- LiteBans maintainer reference
- Configure LiteBans moderation and preserve punishment history
- Connect a Minecraft plugin to a MySQL database

Investigate a LiteBans database connection failure without losing history

A database connection error does not mean punishment history is gone. Preserve the existing database and inspect connectivity before changing storage type or allowing players around active restrictions.

Follow the procedure

1. Capture the first LiteBans startup error and its timestamp from the game console. Remove passwords and connection tokens before sharing the log with support.
2. Run the read-only `/litebans info` command when available and record the reported pool state. Distinguish authentication failures from DNS, connection timeout and schema errors.
3. Compare the configured database host, port, database name and username with the panel’s database details. Check which source host is allowed to connect without opening the database publicly.
4. Have the database administrator verify service health and the existing schema. Take a consistent database backup before any migration or schema repair, including the current configuration.
5. Correct only the identified connection detail. Use the installed release’s supported reload procedure, noting that `/litebans reload` reconnects the database and can briefly change availability.
6. Query a known historical record and a current trial restriction after reconnecting. Confirm both persisted data and current enforcement before marking the incident resolved.



CloudExa game panel: database connection details
CloudExa game panel reference. This image shows the panel, not the LiteBans interface.

Check the result

The original database should reconnect and return known punishment records with their previous identifiers and dates. A clean startup alone is insufficient if the plugin has accidentally connected to a new empty database. Compare several established records before resuming routine moderation.

If the result is wrong

If the connection still fails, restore the prior connection configuration and retain the first error for support. Do not reset tables, switch to an empty database or run an upgrade command repeatedly without a matching recovery plan. Database errors require preserving evidence, not erasing it.

References and related help

- LiteBans maintainer reference
- LiteBans maintainer reference
- Configure LiteBans moderation and preserve punishment history
- Connect a Minecraft plugin to a MySQL database

Reverse one appealed LiteBans punishment while preserving the audit trail

An accepted appeal usually needs a targeted reversal, not removal of the entire punishment history. Use the record identifier to avoid lifting a different restriction on the same account.

Follow the procedure

1. Confirm the appeal decision and the player UUID through your normal ownership process. Collect the relevant punishment ID, type, reason and issuing staff member privately.
2. Use `/history PLAYER` and the matching active-status command to distinguish the appealed record from expired warnings or later punishments. Check whether the restriction is account-based or IP-based.
3. Confirm that your staff role can reverse this record under the installed permissions policy. Some roles can reverse their own actions but cannot reverse another moderator’s punishment.
4. Use the documented unban, unmute or unwarn command with the specific punishment ID. Avoid a delete flag: reversal and destruction of historical evidence are different operations.
5. Read the resulting history entry and active status again. Ask the player to test the previously blocked action through the normal connection route.
6. Add the appeal decision and reversal identifier to your private moderation record. Keep unrelated active restrictions in place until separately reviewed by an authorized moderator.

Check the result

The appealed restriction is inactive, the historical action remains available for review, and the player can perform the action that decision permits. If another independent restriction still blocks access, explain that separately rather than claiming the first reversal failed.

If the result is wrong

If the wrong trial record was targeted, stop and have a senior moderator assess the exact action before issuing another punishment. Do not use staff rollback or history pruning for a single appeal. Retain command output and timestamps so the correction remains accountable.

References and related help

- LiteBans maintainer reference
- LiteBans maintainer reference
- Configure LiteBans moderation and preserve punishment history
- Connect a Minecraft plugin to a MySQL database