GrimAC tasks

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

- Give GrimAC alerts to one observer without granting exemptions
- Remove a GrimAC exemption and verify when checks resume
- Collect a GrimAC prediction log for a false-positive report
- Review a GrimAC punishment interval before enabling enforcement
- Check a GrimAC issue against the actual client and protocol path
- Preserve GrimAC incident evidence before reloading configuration

Give GrimAC alerts to one observer without granting exemptions

An anticheat observer needs to read alerts without bypassing checks or gaining configuration access. Separate the alert permission from exemption permissions before assigning a monitoring role.

Follow the procedure

1. Check the installed Grim version and its Permissions reference. Identify `grim.alerts` for alert access and distinguish it from debug, verbose and configuration permissions.
2. Create a limited observer group in your permission manager. Grant only the documented alert capability and inspect inherited groups for exemptions such as `grim.exempt` or `grim.disabled`.
3. Use a non-operator test observer and the documented `/grim alerts` control. Confirm the account can enable or disable its own supported alert view.
4. Observe genuine existing test activity or a maintainer-supported diagnostic example. Do not encourage players to use a cheating client on production to manufacture evidence.
5. Confirm an ordinary player without the observer role does not receive staff alerts. Review what alert messages disclose before allowing screenshots in public channels.
6. Check that the observer cannot reload configuration or exempt itself. Record where alerts are expected to appear and who should review them before taking moderation action.

Check the result

The observer receives the permitted alert stream and remains subject to normal checks. A quiet period does not prove alert delivery is broken; record the toggle result and verify against a known diagnostic event when available. Alerts are evidence to review, not automatic proof of cheating.

If the result is wrong

If alerts are absent, inspect the role’s effective node, personal alert state and current punishment configuration. If alerts reach ordinary users, withdraw the unintended grant immediately. Avoid broad administrative permissions or exemption nodes as a way to fix a missing monitoring feature.

References and related help

- GrimAC maintainer reference
- GrimAC maintainer reference
- Introduce GrimAC alerts before automated punishments
- Diagnose Minecraft lag with TPS, MSPT and spark

Remove a GrimAC exemption and verify when checks resume

Grim has different exemption mechanisms, and some require a player to rejoin after removal. Identify the actual node before concluding that an exemption change has failed.

Follow the procedure

1. Inspect the player’s effective permissions and identify the precise exemption. The current documentation distinguishes `grim.disabled`, which stops checks, from `grim.exempt`, which unregisters the player.
2. Record why the exemption exists and confirm permission to end it. Check direct grants, parent groups and server contexts so an inherited node cannot immediately restore access.
3. Remove only the intended exemption in your permission manager. Do not clear the entire user profile or remove unrelated staff permissions.
4. For `grim.exempt`, follow the documented rejoin requirement after removal. For other nodes, check their own refresh behaviour rather than assuming every exemption is identical.
5. Use the documented profile or debug controls to verify that Grim tracks the account again. Perform ordinary movement and observe the expected diagnostic state without introducing prohibited client modifications.
6. Check related per-check, no-setback and packet-modification exemptions separately. Record which specific protections are active instead of treating any one node as a complete on/off switch.

Check the result

The account should return to the intended tracking and checking state after the documented refresh step. Removing `grim.exempt` without a rejoin can produce a misleading result. Verify the effective permissions and profile information together before reporting that checks have resumed.

If the result is wrong

If the account remains exempt, inspect inherited groups and the release’s supported platform integrations. Reapply only the prior approved exemption if troubleshooting requires preserving a known state. Do not repeatedly reload Grim as a diagnostic shortcut, since its reload command resets online violations.

References and related help

- GrimAC maintainer reference
- GrimAC maintainer reference
- Introduce GrimAC alerts before automated punishments
- Diagnose Minecraft lag with TPS, MSPT and spark

Collect a GrimAC prediction log for a false-positive report

A useful false-positive report needs the matching prediction evidence, versions and circumstances. Collect a narrow diagnostic log while protecting player information and avoiding changes to enforcement during the capture.

Follow the procedure

1. Record the Grim build, server software, client version, ViaVersion-related components and the exact movement circumstance. Include a timestamp and the alert check name from the incident.
2. Read the current command reference for debug and prediction logging. Confirm the staff role has the documented diagnostic permissions without granting those tools to ordinary players.
3. Use `/grim help` to confirm syntax on the installed build. The documented `/grim log` command accepts a prediction entry index within its supported range; choose the entry corresponding to the observed problem.
4. Capture the generated diagnostic link or output privately. Debug material can include player identifiers and should be reviewed before attaching it to a public issue.
5. Reproduce only the legitimate action that caused the concern, ideally on a staging server with the same plugin combination. Avoid changing thresholds halfway through a capture.
6. Provide the maintainer with the narrow reproduction, versions and relevant log. State whether the event caused an alert, setback or punishment so those outcomes are not conflated.

Check the result

The report should identify a specific check event and contain the corresponding diagnostic evidence, not merely a screenshot saying the player lagged. A reproducible ordinary action with matching versions is more useful than a large unrelated console dump.

If the result is wrong

If the chosen prediction entry is unavailable, capture a fresh occurrence with the supported diagnostic workflow. Turn off any temporary verbose view afterward. Do not upload complete private logs blindly or exempt all players while waiting for analysis; use only a reviewed narrow mitigation if needed.

References and related help

- GrimAC maintainer reference
- GrimAC maintainer reference
- Introduce GrimAC alerts before automated punishments
- Diagnose Minecraft lag with TPS, MSPT and spark

Review a GrimAC punishment interval before enabling enforcement

Grim’s punishment configuration can run an action repeatedly after a threshold. Check both the starting threshold and repeat interval so one noisy check cannot trigger an unintended action loop.

Follow the procedure

1. Open the punishment file generated by your installed Grim release. Save a copy through the panel Files view and identify the exact check group you intend to review.
2. Read the maintainer’s threshold-and-interval explanation. In the documented `N:M` notation, the first number selects the threshold and a positive second number controls later repetitions.
3. Write out the first several firing points for each configured action. A zero repeat interval has a different one-time meaning in the documented format and should be reviewed explicitly.
4. Check the violation-removal window and the checks included in that group. Do not assume counts accumulate forever or that every alert belongs to the same group.
5. Rehearse edits on staging using notification or logging actions before adding a punitive command. Verify command ownership, player substitution and permissions against the installed example file.
6. Have another administrator review the expected sequence. Enable a reviewed production policy only after normal movement and known compatibility cases have been observed without unexplained enforcement.

   CloudExa game panel: server file manager
   CloudExa game panel reference. This image shows the panel, not the GrimAC interface.

Check the result

The configured action sequence should match the written thresholds and intervals, including repetition and violation expiry. Alerting, logging and punitive commands can have different firing points. Checking the first action alone does not prove later repetitions are safe.

If the result is wrong

If an action repeats unexpectedly, revert that group to its saved definition and preserve the alert sequence. Avoid fixing a repeat loop by globally exempting players or deleting evidence. Remember that Grim reload resets online violations, so record the pre-change state before using the supported reload control.

References and related help

- GrimAC maintainer reference
- GrimAC maintainer reference
- GrimAC maintainer reference
- Introduce GrimAC alerts before automated punishments
- Diagnose Minecraft lag with TPS, MSPT and spark

Check a GrimAC issue against the actual client and protocol path

A movement problem limited to one client version may depend on protocol translation rather than the player’s connection alone. Record the full path before changing anticheat thresholds.

Follow the procedure

1. Collect the affected client version and edition, backend Minecraft version, proxy software and any protocol translation plugins. Record where each component runs.
2. Compare this combination with Grim’s current supported-platform and compatibility guidance. A plugin enabling successfully is not a guarantee that every translated client path is supported.
3. Ask the affected player to describe the exact ordinary action and location. Note whether the result is an alert, setback, disconnection or another plugin’s punishment.
4. Repeat that action with an unmodified test client matching the affected version. Compare a directly supported client version through the same public route without opening protected backend ports.
5. Collect the matching Grim profile or prediction diagnostic and relevant server timing evidence. Change one compatibility variable at a time in staging, not several production plugins together.
6. Use an official compatible update or documented mitigation when the evidence identifies a known integration issue. Keep the previous matched plugin set available for rollback.

Check the result

The comparison should establish whether the symptom follows the client/protocol combination, a location or general server load. Record a result for both test clients. Different symptoms should remain separate rather than being grouped under a generic false-positive label.

If the result is wrong

If compatibility remains uncertain, send the maintainer the exact version matrix and narrow diagnostic log. Do not promise support from a different release’s documentation. Restore any unsuccessful trial update with its matching configuration, and avoid weakening all movement checks for an unconfirmed cause.

References and related help

- GrimAC maintainer reference
- GrimAC maintainer reference
- Introduce GrimAC alerts before automated punishments
- Diagnose Minecraft lag with TPS, MSPT and spark

Preserve GrimAC incident evidence before reloading configuration

A Grim reload resets violations for online players according to its command documentation. Capture the evidence first so a configuration correction does not erase the context needed to review an incident.

Follow the procedure

1. Record the incident time, player identity, check names and relevant alert sequence privately. Separate a current enforcement problem from a routine configuration edit.
2. Use supported profile and diagnostic commands to capture the current state. Save only the necessary console excerpt rather than distributing a complete private server log.
3. Back up the current Grim configuration and identify the exact file and setting being changed. Check the installed release’s reload behaviour before scheduling the operation.
4. Notify moderators that online violation counts will reset when using `/grim reload`. Avoid treating post-reload low counts as proof that the earlier incident was false.
5. Apply the reviewed change and use the supported reload command during a suitable window. Read the output for parsing failures and confirm the intended configuration is active.
6. Check normal movement and alert delivery afterward. Retain the before-and-after timestamps so staff can explain the reset when reviewing later enforcement decisions.

Check the result

The configuration should load cleanly while the saved incident material remains available outside the reset live counters. Verify normal operation after reload and document the reset boundary. A fresh zero count is expected behaviour here, not independent evidence that the underlying issue is fixed.

If the result is wrong

If parsing fails, restore the saved configuration and use the documented control to return to the previous policy. Preserve both error output and prior evidence. Do not repeatedly reload during an active investigation or fabricate a continuous violation history across the reset.

References and related help

- GrimAC maintainer reference
- GrimAC maintainer reference
- Introduce GrimAC alerts before automated punishments
- Diagnose Minecraft lag with TPS, MSPT and spark