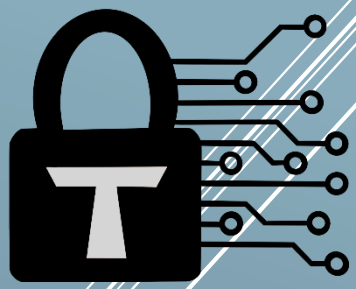
Trust Security



Smart Contract Audit

OlympusDAO – Incentive Distribution

30/04/2026

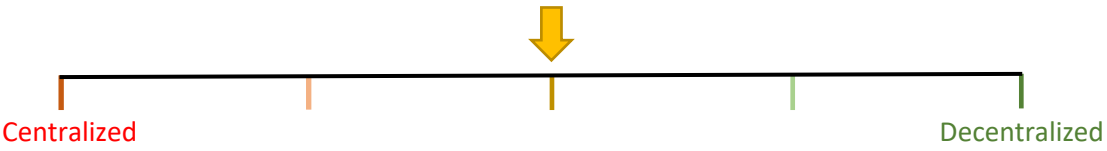
Executive summary

Category	Options
Audited file count	11
Lines of Code	1068
Auditor	HollaDieWaldfee
Time period	01/04/2026 - 07/04/2026

Findings

Severity	Total	Fixed	Acknowledged
High	0	0	0
Medium	2	2	0
Low	2	2	0

Centralization score



Signature

Digitally signed by Trust
Date: 2026.04.30 19:30:18 +03'00'

Document properties

Versioning

Version	Date	Description
0.1	07/04/2026	Client report
0.2	30/04/2026	Mitigation review

Contact

Trust

trust@trust-security.xyz

Introduction

Trust Security has conducted an audit at the customer's request. The audit is focused on uncovering security issues and additional bugs contained in the code defined in scope. Some additional recommendations have also been given when appropriate.

Scope

- src/policies/incentives/BaseIncentiveDistributor.sol
- src/policies/incentives/IncentiveDistributorConvertible.sol
- src/policies/interfaces/incentives/IIncentiveDistributor.sol
- src/policies/interfaces/incentives/IIncentiveDistributorConvertible.sol
- src/policies/incentives/convertible/ConvertibleOHMTeller.sol
- src/policies/incentives/convertible/ConvertibleOHMToken.sol
- src/policies/incentives/convertible/interfaces/IConvertibleOHMTeller.sol
- src/external/clones/Clone.sol
- src/external/clones/CloneERC20.sol
- src/external/clones/CloneERC20Permit.sol
- src/proposals/IncentiveDistributorProposalConvertible.sol

Note:

The exact scoping specification is described in *audit/2026-03_incentives/README.MD*. Some files are reviewed based on a trusted source.

Repository details

- **Repository URL:** https://github.com/OlympusDAO/olympus-v3
- **Commit hash:** 1674171ea9ac1151fd2839cc0d771a5ecc741ef9
- **Mitigation hash #1**: fcd51d851e96d5c63dcf6de5d4bc5e95793e3819
- **Mitigation hash #2:** 1af32cad9ee1e6714fa10e9344a7d2bac2593d43

About Trust Security

Trust Security has been established by top-end blockchain security researcher Trust, in order to provide high quality auditing services. Trust is the leading auditor at competitive auditing service Code4rena, reported several critical issues to Immunefi bug bounty platform and is currently a Code4rena judge.

About the Auditors

HollaDieWaldfee is a distinguished security expert with a track record of multiple first places in competitive audits. He is a Lead Auditor at Trust Security and Senior Watson at Sherlock.

Disclaimer

Smart contracts are an experimental technology with many known and unknown risks. Trust Security assumes no responsibility for any misbehavior, bugs or exploits affecting the audited code or any part of the deployment phase.

Furthermore, it is known to all parties that changes to the audited code, including fixes of issues highlighted in this report, may introduce new issues and require further auditing.

Methodology

In general, the primary methodology used is manual auditing. The entire in-scope code has been deeply looked at and considered from different adversarial perspectives. Any additional dependencies on external code have also been reviewed.

Qualitative analysis

Metric	Rating	Comments
Code complexity	**Excellent**	Code is simple and well structured.
Documentation	**Good**	Project is documented with inline documentation and README.
Best practices	**Good**	Project adheres to best practices.
Centralization risks	**Moderate**	The audited contracts are integrated into the OlympusDAO governance system. A trusted incentive manager is introduced to manage the incentive distribution.

Findings

Medium severity findings

TRST-M-1: DAO multisig can enable arbitrary minting of OHM

- **Category:** Privilege issues
- **Source:** ConvertibleOHMTeller.sol
- **Status:** Fixed

Description

The DAO multisig, which will be configured to hold the teller admin role, is allowed to call *setMintCap()*, and there are no restrictions in place for the maximum cap that can be configured. The cap can be as high as **type(uint256).max**, enabling the lower privilege incentive manager to mint and steal arbitrary OHM amounts.

Since the economic value of OHM is at the core of the OlympusDAO protocol, minting OHM should be protected by the highest-level privilege which is the admin role held by on-chain governance. Thus, the ability of the DAO multisig to configure the mint cap arbitrarily presents a privilege escalation vector. This is evidenced by the documentation which references OIP-152. The OIP states that critical roles have been phased out from the DAO multisig and transferred to governance.

Recommended mitigation

For the intended use case of dynamically adjusting the mint cap while the incentive distribution system is being rolled out, it Is sufficient if the DAO multisig can adjust the mint cap within a reasonable range, but not arbitrarily.

Therefore, the first recommendation is to introduce a limit for the minted OHM that is configurable by the admin. As a result, even though the DAO can arbitrarily set the mint cap (i.e., the mint approval), the amount of OHM minted is restricted.

To provide further protection against unauthorized minting of OHM, there should be separate admin mint limits for each incentive distributor to avoid contamination across incentive distributor boundaries.

Finally, as has been suggested as part of TRST-L-2, a **minEligibleDelay** should be implemented to provide the emergency role with a time window to disable malicious policies before they can mint OHM.

The code changes for the third suggestion are part of TRST-L-2, while the first two suggestions are detailed below:

```
diff --git a/src/policies/incentives/convertible/ConvertibleOHMTeller.sol
b/src/policies/incentives/convertible/ConvertibleOHMTeller.sol
index 2969976f..730125ad 100644
--- a/src/policies/incentives/convertible/ConvertibleOHMTeller.sol
+++ b/src/policies/incentives/convertible/ConvertibleOHMTeller.sol
@@ -61,8 +61,9 @@ contract ConvertibleOHMTeller is
     /// @notice The OHM token decimals
     uint8 private constant _OHM_DECIMALS = 9;
```

```
-    /// @notice Expected byte length of enableData_ (one ABI-encoded uint256)
-    uint256 private constant _ENABLE_DATA_LENGTH = 32;
+    /// @notice Minimum byte length of enableData_ (one ABI-encoded uint256 mintCap +
two dynamic arrays)
+    /// @dev    abi.encode(uint256, address[], uint256[]) with empty arrays = 32 + 32
+ 32 + 32 + 32 = 160
+    uint256 private constant _MIN_ENABLE_DATA_LENGTH = 160;

     /// @notice Minimum decimals required for quote tokens (used by _formatPrice)
     uint8 private constant _MIN_QUOTE_TOKEN_DECIMALS = 2;
@@ -87,6 +88,10 @@ contract ConvertibleOHMTeller is
     /// @inheritdoc IConvertibleOHMTeller
     uint48 public override minDuration;

+    mapping(address creator => uint256) public ohmMinted;
+
+    mapping(address creator => uint256 adminMintLimit) public adminMintLimits;
+
     // ========== CONSTRUCTOR ========== //

     /// @param kernel_ The address of the Olympus kernel
@@ -149,11 +154,27 @@ contract ConvertibleOHMTeller is

     /// @notice Overrides _enable to set the initial minting cap
     /// @param enableData_ ABI-encoded (uint256 mintCap)
+    /// @notice Enables the teller with a mint cap and optional per-creator mint
limits.
+    /// @param enableData_ ABI-encoded (uint256 mintCap, address[] creators,
uint256[] limits)
     function _enable(bytes calldata enableData_) internal override {
-        if (enableData_.length != _ENABLE_DATA_LENGTH) revert Teller_InvalidParams(0,
enableData_);
+        if (enableData_.length < _MIN_ENABLE_DATA_LENGTH)
+            revert Teller_InvalidParams(0, enableData_);
+
+        (uint256 mintCap, address[] memory creators, uint256[] memory limits) =
abi.decode(
+            enableData_,
+            (uint256, address[], uint256[])
+        );
+
+        if (creators.length != limits.length)
+            revert Teller_InvalidParams(1, abi.encodePacked(creators.length,
limits.length));

-        uint256 mintCap = abi.decode(enableData_, (uint256));
         _setMintCap(mintCap);
+
+        for (uint256 i = 0; i < creators.length; ++i) {
+            if (creators[i] == address(0))
+                revert Teller_InvalidParams(1, abi.encodePacked(creators[i]));
+            adminMintLimits[creators[i]] = limits[i];
+        }
     }

     // ========== TOKEN DEPLOYMENTS ========== //
@@ -241,7 +262,7 @@ contract ConvertibleOHMTeller is
         (
             ConvertibleOHMToken token,
             address quoteToken,
-            ,
+            address creator,
             uint48 eligible,
             uint48 expiry,
             uint256 price
@@ -257,6 +278,9 @@ contract ConvertibleOHMTeller is
         // Burn convertible tokens
         token.burnFrom(msg.sender, amount_);
```

```
+        ohmMinted[creator] += amount_;
+        require(ohmMinted[creator] <= adminMintLimits[creator], "mint limit
exceeded");
+
         // Mint OHM to user
         MINTR.mintOhm(msg.sender, amount_);

@@ -554,6 +578,10 @@ contract ConvertibleOHMTeller is
         minDuration = duration_;
     }

+    function setAdminMintLimit(address creator_, uint256 limit_) external onlyEnabled
onlyAdminRole {
+        adminMintLimits[creator_] = limit_;
+    }
+
     /// @inheritdoc IConvertibleOHMTeller
     function setMintCap(uint256 cap_) external override onlyEnabled
onlyAdminOrTellerAdminRole {
         _setMintCap(cap_);
```

Team response

Fixed by commit fcd51d8.

Mitigation review

The finding has been mitigated by removing the role for the DAO multisig. Instead, there are now per-creator mint caps that can only be configured by the admin. Furthermore, the **minEligibleDelay** as recommended in TRST-L-2 has been implemented as well.

TRST-M-2: Teller name mismatch in *IncentiveDistributorProposalConvertible* breaks proposal submission

- **Category:** Deployment issues
- **Source:** IncentiveDistributorProposalConvertible.sol
- **Status:** Fixed

Description

The *IncentiveDistributorProposalConvertible* looks up **olympus-policy-iohm-teller**, but *addresses.json* declares the entry as **olympus-policy-convertible-ohm-teller**. Consequently, proposal submission via *IOHMIncentiveDistributorProposalScript.run()* reverts in *_build()* with **"Address: olympus-policy-iohm-teller not set on chain: 1",** preventing on-chain submission of the proposal and breaking the deployment flow.

Recommended mitigation

It is recommended to rename every instance referencing **olympus-policy-iohm-teller** to **olympus-policy-convertible-ohm-teller**, matching the entry already present in *addresses.json*. This brings the proposal, test, and address registry into agreement on a single canonical name.

Team response

Fixed in commit 1af32ca.

Mitigation review

Verified, the recommendation has been implemented.

Low severity findings

TRST-L-1: *endEpoch()* allows setting a future epochEndDate

- **Category:** Validation issues
- **Source:** IncentiveDistributorConvertible.sol
- **Status:** Fixed

Description

The *endEpoch()* function validates in *BaseIncentiveDistributor._setMerkleRoot()* that the epoch end date is formatted as 23:59:59 UTC (end of day) and is at least **MIN_EPOCH_DURATION** after the last epoch, but it does not check that **epochEndDate_** is in the past relative to **block.timestamp**. Logically, an epoch end date represents the end of a distribution period and setting it in the future implies distributing rewards for a period that hasn't ended yet. As such, **block.timestamp <= epochEndDate_** is an inconsistent state that should be protected against.

Recommended mitigation

It should be checked that **epochEndDate** is in the future.

```
diff --git a/src/policies/incentives/IncentiveDistributorConvertible.sol
b/src/policies/incentives/IncentiveDistributorConvertible.sol
index f12febb9..ee8183b6 100644
--- a/src/policies/incentives/IncentiveDistributorConvertible.sol
+++ b/src/policies/incentives/IncentiveDistributorConvertible.sol
@@ -89,10 +89,7 @@ contract IncentiveDistributorConvertible is
             (IIncentiveDistributorConvertible.EndEpochParams)
         );

-        // Validate that the token expires after the end of the epoch (users need
time to claim their tokens)
-        // Note: epochEndDate_ is always 23:59:59 UTC (validated by _setMerkleRoot),
so
-        // p.expiry > epochEndDate_ guarantees expiry falls on a strictly later day
-        if (p.expiry <= epochEndDate_) revert IncentiveDistributor_InvalidToken();
+        if (block.timestamp <= epochEndDate_) revert
IncentiveDistributor_InvalidToken();

         // Set and validate the merkle root
         _setMerkleRoot(epochEndDate_, merkleRoot_);
```

This recommendation replaces the **p.expiry <= epochEndDate_** check with **block.timestamp <= epochEndDate_**. The consistency of **p.expiry** with **epochEndDate_** will be checked downstream in *ConvertibleOHMTeller* as detailed in the next finding.

Team response

Fixed in commit d60002b.

Mitigation review

Verified, the recommendation has been implemented. The check for **p.expiry <= epochEndDate_** still exists which acts as a redundant second layer of protection.

TRST-L-2: Truncated block.timestamp in eligibility check allows eligibility in the past

- **Category:** Validation issues
- **Source:** ConvertibleOHMTeller.sol
- **Status:** Fixed

Description

The *deploy()* function truncates both **eligible_** and **block.timestamp** to UTC midnight before comparing **if (eligible_ < _truncateToUTCDay(uint48(block.timestamp)))**. This means any **eligible_** timestamp within the current UTC day passes the check, even though it may be hours in the past relative to the actual **block.timestamp**. Combined with the **eligible_ == 0** shorthand, which sets **eligible_ = block.timestamp** and then truncates it to midnight, a token deployed at 23:59 UTC would have an eligible timestamp of 00:00 UTC on the same day, nearly 24 hours in the past.

This has two consequences. Firstly, the token is immediately exercisable upon deployment, undermining the concept of a future eligibility date. Secondly, the effective time until expiry is up to one day less than the validated **minDuration**, since the duration check uses the truncated **eligible_** rather than the actual current time.

Recommended mitigation

The validation should use the raw **block.timestamp** without truncation, ensuring **eligible_** is genuinely in the future and that the full **minDuration** is preserved from the moment of deployment.

Furthermore, a **minEligibleDelay** should be implemented, like **minDuration**, which enforces a minimum delay until a token becomes eligible. This hardens the logic against centralization risks, ensuring the emergency role has sufficient time to disable the contract if malicious Merkle roots are detected.

Finally, to preserve the semantics of **eligible_ = 0**, it must set the eligibility to the earliest possible allowed timestamp.

```
diff --git a/src/policies/incentives/convertible/ConvertibleOHMTeller.sol
b/src/policies/incentives/convertible/ConvertibleOHMTeller.sol
index 2969976f..6b045806 100644
--- a/src/policies/incentives/convertible/ConvertibleOHMTeller.sol
+++ b/src/policies/incentives/convertible/ConvertibleOHMTeller.sol
@@ -87,6 +87,8 @@ contract ConvertibleOHMTeller is
     /// @inheritdoc IConvertibleOHMTeller
     uint48 public override minDuration;

+    uint48 public minEligibleDelay;
+
     // ========== CONSTRUCTOR ========== //

     /// @param kernel_ The address of the Olympus kernel
@@ -104,6 +106,7 @@ contract ConvertibleOHMTeller is

         // Set the minimum duration during which a convertible token must be eligible
for exercise to 1 day initially
         minDuration = uint48(1 days);
+        minEligibleDelay = uint48(1 days);
     }

     // ========== POLICY CONFIGURATION ========== //
@@ -172,15 +175,18 @@ contract ConvertibleOHMTeller is
```

```
          nonReentrant
          returns (address)
      {
-         // If eligible is zero, use the current timestamp
-         if (eligible_ == 0) eligible_ = uint48(block.timestamp);
+         if (eligible_ == 0) {
+             // Smallest UTC midnight >= block.timestamp + minEligibleDelay
+             uint48 minTimestamp = uint48(block.timestamp) + minEligibleDelay;
+             eligible_ = _truncateToUTCDay(minTimestamp + 1 days - 1);
+         }

          // Note: Convertible tokens are only unique to a day, not a specific
timestamp
          (eligible_, expiry_) = _truncateBothToUTCDay(eligible_, expiry_);

          // Revert if eligible is in the past, we do this to avoid duplicates tokens
with the same parameters otherwise
          // Truncate block.timestamp to the nearest day for comparison
-         if (eligible_ < _truncateToUTCDay(uint48(block.timestamp)))
+         if (eligible_ < uint48(block.timestamp) + minEligibleDelay)
              revert Teller_InvalidParams(1, abi.encodePacked(eligible_));

          // Revert if the difference between eligible and expiry is less than min
duration or eligible is after expiry
@@ -554,6 +560,11 @@ contract ConvertibleOHMTeller is
          minDuration = duration_;
      }

+     function setMinEligibleDelay(uint48 delay_) external onlyEnabled onlyAdminRole {
+         if (delay_ < uint48(1 days)) revert Teller_InvalidParams(0,
abi.encodePacked(delay_));
+         minEligibleDelay = delay_;
+     }
+
      /// @inheritdoc IConvertibleOHMTeller
      function setMintCap(uint256 cap_) external override onlyEnabled
onlyAdminOrTellerAdminRole {
          _setMintCap(cap_);
```

Team response

Fixed by commit [fcd51d8](fcd51d8).

Mitigation review

Verified, the recommendation has been implemented.

Additional recommendations

TRST-R-1: Option token implementation should be locked as a best practice

The *ConvertibleOHMToken* implementation contract deployed in the *ConvertibleOHMTeller* constructor is not locked against direct interaction. Since *_getImmutableArgsOffset()* reads the last 2 bytes of calldata as the immutable args length, and the EVM allows appending arbitrary bytes to calldata, an attacker can craft a call to *mintFor()* on the implementation that includes immutable args with their own address at the teller offset. This allows arbitrary minting on the implementation's storage. While there's no impact to interacting with the implementation contract, it is best practice to lock proxy implementations. This recommendation may be omitted if it is determined that it's more important to minimize gas costs.

TRST-R-2: Missing zero address check in *previewClaim()*

The *previewClaim()* function does not return early when **user_** is **address(0)**. In comparison, the actual *claim()* flow calls *TELLER.create()* which reverts for **address(0)**. This means *previewClaim()* deviates from the behavior of *claim()*. Adding an early return for **user_ == address(0)** (alongside the existing array length check) makes the preview accurately reflect the claim outcome.

TRST-R-3: Unsafe int8 cast in *_getPriceDecimals()*

The *_getPriceDecimals()* function casts **tokenDecimals_** to **int8**, which has a range of **-128** to **127**. For **tokenDecimals_ > 127**, the cast silently wraps to a negative value. For example, **int8(129) = -127** and **int8(255) = -1**. This causes the subtraction to produce an incorrect positive result instead of the correct deeply negative value. For instance, with **price_ = 1** and **tokenDecimals_ = 129**, the function returns **+127** instead of the correct **-129**, which would pass the **priceDecimals < -int8(quoteDecimals / 2)** validation. The practical impact is negligible since no real ERC20 token has more than 127 decimals, and such decimals cause a revert downstream in *_formatPrice()*, when **10**tokenDecimals_** is evaluated. Nonetheless, it is recommended to revert gracefully when **tokenDecimals_ > 77** which is the maximum possible decimals count for any **uint256** price.

TRST-R-4: Token name silently truncates for large strike prices

The token name is constructed as **"OHM/" + quoteSymbol + " " + price + " " + YYYYMMDD** and cast to **bytes32**, which silently discards any bytes beyond 32. With a 4-character quote symbol (e.g., USDS), the fixed portions consume 18 bytes, leaving only 14 bytes for the formatted price string. Since the price format is **whole digits + "." + 2 fractional digits**, a strike

price with 12 or more whole digits causes the date suffix, and potentially part of the price itself, to be truncated. This is only a cosmetic issue, the token's functional parameters (*strike()*, *expiry()*, etc.) are unaffected, and such extreme strike prices are economically unrealistic. It is recommended to document the maximum representable strike price, or to dynamically adopt a scientific notation (e.g., **"111e12"**) when the whole digits exceed the available space.

TRST-R-5: Expiry timestamp has no upper bound check, causing corrupted name and symbol

The *deploy()* function validates that **expiry_** is after **eligible_** by at least **minDuration** and that **eligible_** is not in the past, but it does not enforce any upper bound on how far in the future **expiry_** can be. An authorized distributor could deploy a convertible token with an **expiry_** that is thousands or even millions of years in the future. Consequently, the *Timestamp.toPaddedString()* library produces incorrect or misleading date strings for years beyond 9999 (due to **year % 10_000** truncation). And *toPaddedString()* is also documented with a disclaimer that it is not tested for years greater than 2345. It is recommended to check **expiry_** against a reasonable maximum like 1-2 years ahead of **block.timestamp**. This ensures the date in the token name and symbol is well-formed. It must be noted that the ability for the incentive manager to set an **expiry_** far into the future is not a security issue, since the role is already fully trusted to provide the correct Merkle root, so the finding only concerns the corrupted name and symbol.

TRST-R-6: Missing kernel validation in *BaseIncentiveDistributor*

Unlike *ConvertibleOHMTeller*, *BaseIncentiveDistributor* does not validate that the kernel is not **address(0)** in its constructor. This missing check should be implemented for consistency.

Centralization risks

TRST-CR-1: Admin is fully trusted

The admin role within the audited incentive distribution system is the same that is used throughout the whole protocol. As such, the admin is fully trusted. Specifically, within the incentive distribution contracts, the admin can enable policies, disable policies, call *setMinDuration()*, *setMinEligibleDelay()* and *setCreatorMintCap()*. This role is held by the on-chain governance.

TRST-CR-2: Emergency role can disable policies

Like the admin, the emergency role is scoped to the whole protocol. It can disable policies, i.e., prevent all interactions with *IncentiveDistributorConvertible* and *ConvertibleOHMTeller*.

TRST-CR-3: Incentive manager can set Merkle roots

The incentive manager is responsible for calling *endEpoch()*, which sets the merkle root for an epoch and deploys the option token. As a consequence, a malicious incentive manager can steal all OHM up to the *IncentiveDistributorConvertible* contract's mint cap. All other option tokens already created remain unaffected since their OHM approval has already been granted. More broadly, the malicious minting of OHM damages the whole protocol economically.

Systemic risks

TRST-SR-1: External quote token risk

The incentive manager role can call *IncentiveDistributorConvertible.endEpoch()*, which accepts the **quoteToken** that the option is configured with. Unlike OHM, quote tokens are external tokens, and so there's a systemic risk that the quote token loses its value. This damages the protocol economically since options can be exercised at an unexpectedly low value. On the other hand, rising quote token prices make exercising the options less profitable. Notably, one malicious quote token that is introduced can only affect the options that are configured with this token. There is no disruption for other options, except for the broader economic damage incurred by the protocol.