UMA Audit – Phase 3

Fixes & Refactors

OPENZEPPELIN SECURITY | SEPTEMBER 9, 2020 Security Audits

UMA is a platform that allows users to enter trust-minimized financial contracts on the Ethereum blockchain. We previously audited the decentralized oracle and a particular financial contract template. In this phase, the UMA team asked us to audit seven pull requests that make changes to Solidity contracts that we have already audited in previous phases.

Since the pull requests make local, isolated, changes to contracts that were previously audited and described, this report describes each change along with any findings from our audit.

Flash loan mitigation

Implemented in PR#1767.

The UMA voting mechanism uses a commit-reveal scheme, where votes are weighted in proportion to the token balance of the voters. The corresponding snapshot of the token distribution is taken at the start of each reveal phase, either through a dedicated function call, or automatically alongside the first revealed vote.

However, it is possible for users to take out a flash loan of UMA tokens from a decentralized exchange, trigger the snapshot and then repay the loan in a single transaction. This would arguably give them undue influence over the result of the vote. A similar effect is also possible via short-term loans using dedicated lending platforms, although this scenario may be more tolerable since token holders are more directly choosing to lend their tokens and corresponding governance

The reviewed pull request addresses the flash-loan scenario by requiring the address that triggers the snapshot to provide an ECDSA signature over a known message, which guarantees that it is an externally owned account. For simplicity, the ability to automatically trigger a snapshot on the reveal has also been removed, although the same effect can be achieved with multiple function calls.

After auditing the pull request, our only recommendations involve minor improvements to code quality:

- The new `snapshotMessage` variable should be renamed `snapshotMessageHash`.
- The direct conversion `bytes("Sign For Snapshot")` is cleaner than `abi.encodePacked("Sign for Snapshot")`.

Update: *Our recommendations are being implemented in PR#1971.*

New events in the expiring multi-party template

Implemented in PR#1753

This pull request adds additional parameters to existing events:

- The `LiquidationCreated` event, emitted during execution of the `createLiquidation` function, now includes a `liquidationTime` parameter.
- The `LiquidationWithdrawn` event, emitted during execution of the `withdrawLiquidation` function, now includes a `settlementPrice` parameter.

After auditing the pull request, we do not have any recommendations.

Update: *During our audit, the UMA team identified a minor bug in the data logged by the `LiquidationWithdrawn` event. The corresponding fix has been implemented in PR#1912.*

Removal of expiring multi-party creator restrictions

Implemented in PR#1746

`ExpiringMultiPartyCreator` contract, use a global whitelist registered with the `Finder` contract.

- Remove the allowed expiration whitelist and instead allow the contracts to expire at any future timestamp.
- Remove the hard-coded liquidation liveness and withdrawal liveness parameters and allow the contract creator to choose any non-zero time limits. Previously, these were set to 7200 seconds each.

After auditing the pull request, our recommendations are:

- Explicitly confirm that the expiration timestamp is in the future in the `_convertParams` function of the `ExpiringMultiPartyCreator` contract, so that the configuration can be reasoned about from within the contract, even though this restriction will be enforced during construction when the template is deployed.
- Consider re-adding all relevant documentation related to the liquidation and withdrawal liveness parameters. In particular, related to how low values might affect usability for sponsors and liquidators, and the risks users might be exposed to when interacting with contracts with extremely low liquidation and withdrawal liveness periods.
- Consider putting a hard-cap on the liquidation liveness and withdrawal liveness parameters. Currently, the applied changes allow setting arbitrarily large values, which could cause undesired reverts due to overflows. In particular:
  - At lines 306 and 559 of `Liquidatable.sol`. This would allow deployers to, accidentally or maliciously, craft contracts that might not be able to undergo a liquidation process.
  - At lines 187 and 332 of `PricelessPositionManager.sol`. This would allow deployers to, accidentally or maliciously, craft contracts that might not allow transfers of positions, or request withdrawal of funds.

Update: *Our recommendations are being implemented in PR#1971.*

Token creation collateralization

Implemented in PR#1844

(GCR) is used as a safeguard, since it would be an overestimate of the required value under some reasonable economic assumptions. Whenever a new position is opened, the user is required to match this GCR with the corresponding new collateral.

However, this may be unnecessarily restrictive, since the collateralization ratio of each change in the position is not individually relevant. This pull request relaxes the requirement to ensure that either the final user position matches the GCR, or the new collateral and minted tokens match the GCR (as before).

After auditing the pull request, our only recommendation is to change the error message associated with an under-collateralized position from “New CR below GCR” to something more generic like “Insufficient collateral”. This is because there are cases when the original position was below the GCR, the change in position is above the GCR and the resulting position is still below the GCR. This is a valid operation that (correctly) does not trigger the error, but nevertheless matches the text of the error message.

Update: *Our recommendation is being implemented in PR#1971.*

Reset withdrawal timer

Implemented in PR#1859

When a withdrawal that would put a position below the Global Collateralization Ratio is requested, liquidators have a time window to liquidate the position if the withdrawal is invalid. However, if the position is sufficiently large, it may be difficult for liquidators to (collectively) obtain enough synthetic tokens within the allotted time window.

This pull request resets the withdrawal time window whenever a partial liquidation that is sufficiently large occurs within the window. It should be noted that although this mechanism can be used to delay valid withdrawals, potentially until expiration, it does not remove the usual penalty associated with false liquidations, which would have to be paid for every delaying transaction. This issue is already acknowledged by the UMA team.

After auditing the pull request, we suggest the following modifications:

condition of the added `if` statement should be rephrased or removed to avoid confusions, as it is not related to the actual condition being evaluated (that is, that the position is undergoing a slow withdrawal that has not yet expired).

- The use of the `minSponsorToken` parameter, as a lower bound to decide whether the maximum liquidated amount is sufficiently large, is not self-explanatory, and should therefore be better documented to favor readability and avoid confusions.
- The bounds comparison should use `tokensLiquidated` instead of `maxTokensToLiquidate` to better capture the intent of the check.

Update: *Our recommendations are being implemented in PR#1971.*

Trim excess collateral

Implemented in PR#1975

Each `ExpiringMultiParty` holds collateral deposits in escrow so they can be eventually distributed to users. A fraction of those funds are also sent to the `Store` contract to pay for the DVM. Any additional collateral or other tokens held by the contract are unnaccounted for, and are trapped inside the contract.

This pull request adds a new `excessTokenBeneficiary` address, which is chosen by the creator and fixed upon deployment. This address must be different from the zero address. It also introduces a `trimExcess` function that can be called by anyone to send any excess funds to the `excessTokenBeneficiary` address.

After auditing the pull request, our only recommendation is to add docstrings to the `trimExcess` function following the Ethereum Natural Specification Format (NatSpec). This will improve readability and ease maintenance in future changes to the code base.

Update: *Our recommendation is being implemented in PR#1975.*

Redeem after expiration

Implemented in PR#1968

they are first burned (reducing the token debt). Assuming the position is fully collateralized, the final result will be the same if token holders were able to simply redeem their tokens, which is simpler and does not rely on the price being resolved.

This pull request removes the time restriction on the `redeem` function so it can be called even after the contract expires. It also allows the `cancelWithdrawal` function to be called after expiration, so that a pending slow withdrawal will not stall the token redemption.

After auditing the pull request, we do not have any recommendations.

Conclusions

We audited 7 pull requests that make local, isolated, changes to contracts that had been audited in previous phases of our engagement with the UMA team. We suggested changes to reduce the code's attack surface and additional modifications to favor overall code quality. All suggested changes are already being addressed by the UMA team.

Related Posts

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users'

OpenBrush Contracts Library Security Review

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-

Security Audits

Security Audits

Security Audits

OpenZeppelin

Defender Platform

Secure Code & Audit
Secure Deploy
Threat Monitoring
Incident Response
Operation and Automation

Services

Smart Contract Security Audit
Incident Response
Zero Knowledge Proof Practice

Learn

Docs
Ethernaut CTF
Blog

Company

About us
Jobs
Blog

Contracts Library

Docs

Privacy | Terms of Use