Alongside

Security Review

Cantina Managed review by:

0xRajeev, Lead Security Researcher

0xleastwood, Lead Security Researcher

Defsec, Security Researcher

September 29, 2023

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Directly* exploitable security vulnerabilities that need to be fixed.
High	Security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All high issues should be addressed.
Medium	Objective in nature but are not security vulnerabilities. Should be addressed unless there is a clear reason not to.
Low	Subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. When determining the severity one first needs to determine whether the finding is subjective or objective. All subjective findings are considered of Minor severity.

Next it is determined whether the finding can be regarded as a security vulnerability. Some findings might be objective improvements that need to be fixed, but do not impact the project's security overall (Medium).

Finally, objective findings of security vulnerabilities are classified as either critical or major. Critical findings should be directly vulnerable and have a high likelihood of being exploited. Major findings on the other hand may require specific conditions that need to be met before the vulnerability becomes exploitable.

2 Security Review Summary

AMKT is a fully backed market index, providing exposure to a market-cap weighted basket of assets, to be reconstituted quarterly.

From September 11th to September 15th the Cantina team conducted a review of index-system-v2 on commit hash c1afc5a6. The team identified a total of **43** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 1
- Medium Risk: 8
- Low Risk: 10
- Gas Optimizations: 3
- Informational: 21

3 Findings

3.1 High Risk

3.1.1 `invariantCheck` calculates expected amount using `realUnits`

Severity: High Risk

Context: Vault.sol#L288-L302

Description: The `invariantCheck()` function is used to enforce sufficient vault balances after any function which may invoke a token mint, burn or rebalance. `expectedAmount` is calculated by querying `realUnits()` which effectively calculates what each index token is worth after fees have been applied. This value decreases linearly as the fee is applied daily but `currentMultiplier` is calculated as if the fee is charged each second.

If `invariantCheck()` were to be called in the last block before the daily fee was applied, then `token[i].units` will take into consideration this fee as if new index tokens have already been minted, but `totalSupply` will not reflect this until an entire day has passed. As a result, there is room to skim up to but strictly less than the daily fee by interacting with any function that enforces the protocol invariant through this function.

Recommendation: Consider including any un-minted index tokens in `totalSupply` when calculating the `expectedAmount` used to enforce the vault invariant.

3.2 Medium Risk

3.2.1 Tokens with callbacks can circumvent the supply check in `fulfillBounty`

Severity: Medium Risk

Context: Bounty.sol#L115-L130

Description: If a token with callback was added into the protocol, the original `indexToken.totalSupply() != startingSupply` check would be circumvented, allowing the rebalancer to deposit tokens into the protocol and counteract the rebalance action. As a result, the new nominal amounts for each token would not be properly enforced.

Recommendation: Consider adding a supply check after transferring in all tokens from `msg.sender`, this serves to ensure the rebalancer did not attempt to mint or redeem tokens in a way that would impact token nominal amounts.

3.2.2 Doubling `AMKT` as a governance token may facilitate governance attacks

Severity: Medium Risk

Context: Governor.sol

Description: The protocol plans to introduce governance in this upgrade by introducing v2 of `AMKT` token which not only serves as their index token but also doubles as their governance token. As described in their post, this is meant to be a differentiator over some of their competitors which have a governance token separate from the index token:

> « AMKT is also managed through a DAO, which controls the management of the index's token hierarchy. Community governance decisions include, but not limited to, contract changes relating to the DAO tokens or their issuance, whitelisting of custodians and merchants through a Governance-controlled multi-signature contract, and index weighting and methodology changes. Unlike Index Coop's INDEX token, AMKT tokens double as governance tokens – in addition to being fully-backed index tokens. This means AMKT holders immediately gain direct governance power in proportion to how much they have purchased, without needing to acquire a separate governance token. »

`AMKT` being an index token has no supply cap and can be issued/redeemed at will by anyone with appropriate quantities of the underlying tokens. While the risk from capital-based token-weighted governance exists with all such governed protocols, doubling AMKT as the governance token introduces a different risk in that, being an index token, it can be minted/burned at will without much token-holder risk and in arbitrary quantities (limited by the availability of the underlying tokens of course) without there being a supply cap.

In effect, sufficiently funded actors could takeover governance by minting large quantities of `AMKT` with underlyings, attacking governance and then redeeming them back to safer underlyings. While this is generally possible with any governance token, the dual-purpose `AMKT` token makes this less riskier and easier to exploit.

The planned mitigation is for the protocol MultiSig to monitor any such governance attacks using malicious proposals, veto them and trigger emergency mode when necessary.

Recommendation: Reconsider the design choice of using the `AMKT` index token to double as the governance token.

3.2.3 A compromised MultiSig can affect key protocol operations

Severity: Medium Risk

Context: MultiSig

Description: The protocol has a deployed MultiSig which is currently a 4/8 threshold with plans to upgrade to a 5/9. This MultiSig plays a critical role during the planned protocol upgrade and is planned to retain the Timelock `CANCELLER_ROLE` which can veto malicious governance proposals and the `emergencyResponder` role which can trigger emergency mode to prevent issuance and rebalancing of index tokens.

A compromised MultiSig can veto any governance proposal to take away the emergency responder role from itself while griefing the protocol to prevent issuance and rebalancing by triggering emergency mode. It may also arbitrarily veto any governance proposal.

A compromised MultiSig can also renounce its Timelock `CANCELLER_ROLE` which will make it impossible to veto any future malicious proposals that can harm protocol funds/operations.

During the upgrade, the MultiSig is made the vault owner and it then proposes the TimeLockController to take over the ownership. The timelock has to accept the ownership via a governance proposal which has to be executed as the first proposal. During this time, the MultiSig retains ownership to the vault which has all the user tokens. Any compromise here can drain the vault.

It is therefore a point of key centralization risk.

Recommendation: Upgrade the MultiSig to 5/9 as planned. Ensure individual owners are well-known hardware wallets operated by different and isolated entities. Enforce wallet best-practices. Document an incident response plan for a compromised MultiSig.

3.2.4 Fees are retroactively applied to users when `feeScaled` is changed

Severity: Medium Risk

Context: Vault.sol#L123-L146, Multiplier.sol#L65-L72

Description: `tryInflation()` captures fees on a daily basis by minting index tokens to the `feeRecipient` address. The vault contract keeps track of _two_important variables for this, `trackedMultiplier` which represents fees already charged and `currentMultiplier` which accounts for any fees yet to be claimed.

In `computeMultiplier()`, when `dT != 0`, `feePerSecondScaled` is used to apply the expected fee, however, if at any point, `setFeeScaled()` is called to update this fee, then `currentMultiplier` would no longer be a linear increase. This proves unexpected for users who choose not to redeem their index tokens and then reenter after the fee update.

Recommendation: Ensure this is documented and understood.

3.2.5 Index token v2 is not compatible with `EIP-712` due to an improper storage layout

Severity: Medium Risk

Context: IndexToken.sol

Description: The index token migration process completely changes the storage layout of original contract. Fortunately, most of it's prior state is preserved correctly as we can see in the *two* following snippets which outline the storage layout of index token's v1 and v2 implementations, we can see that everything is correct except for *two* storage slots; namely, `_HASHED_NAME` clashes with `_owner` and `_HASHED_VERSION` clashes with `__gap`.

```
 cast storage 0x88f84864fd0839a7753199b01acb89c4714319f2 --rpc-url https://eth.llamarpc.com/
[] Compiling...
No files changed, compilation skipped
No matching artifacts found, fetching source code from Etherscan...
```

Name	Type	Slot	Offset	Bytes	Value
_initialized	uint8	0	0	1	1
_initializing	bool	0	1	1	1
__gap	uint256[50]	1	0	1600	0
_balances	mapping(address => uint256)	51	0	32	0
_allowances	mapping(address => mapping(address => uint256))	52	0	32	0
_totalSupply	uint256	53	0	32	0
_name	string	54	0	32	4768621134925043971832549256884602404311035415051812575159338651313762205698 8
_symbol	string	55	0	32	4768621134925043971832549256884602404311035415051812575159338651313762205698 8
__gap	uint256[45]	56	0	1440	0
_owner	address	101	0	20	833204653418720119172618286816296123974065860883
__gap	uint256[49]	102	0	1568	0
proposedOwner	address	151	0	20	0
__gap	uint256[50]	152	0	1600	0
_paused	bool	202	0	1	0
__gap	uint256[49]	203	0	1568	0
feeRatePerDayScaled	uint256	252	0	32	0
feeTimestamp	uint256	253	0	32	1665698027
feeReceiver	address	254	0	20	1
methodologist	address	255	0	20	0
minter	address	256	0	20	0
methodology	string	257	0	32	0
supplyCeiling	uint256	258	0	32	0
isRestricted	mapping(address => bool)	259	0	32	0

(The `Contract` column was removed due to redundancy).

```
forge inspect IndexToken storage-layout --pretty
```

Name	Type	Slot	Offset	Bytes
_initialized	uint8	0	0	1

Name	Type	Slot	Offset	Bytes
_initializing	bool	0	1	1
__gap	uint256[50]	1	0	1600
_balances	mapping(address => uint256)	51	0	32
_allowances	mapping(address => mapping(address => uint256))	52	0	32
_totalSupply	uint256	53	0	32
_name	string	54	0	32
_symbol	string	55	0	32
__gap	uint256[45]	56	0	1440
_HASHED_NAME	bytes32	101	0	32
_HASHED_VERSION	bytes32	102	0	32
__gap	uint256[50]	103	0	1600
_nonces	mapping(address => struct CountersUpgradeable.Counter)	153	0	32
_PERMIT_TYPEHASH_DEPRECATED_SLOT	bytes32	154	0	32
__gap	uint256[49]	155	0	1568
_delegates	mapping(address => address)	204	0	32
_checkpoints	mapping(address => struct ERC20VotesUpgradeable .Checkpoint[])	205	0	32
_totalSupplyCheckpoints	struct ERC20VotesUpgradeable .Checkpoint[]	206	0	32
__gap	uint256[47]	207	0	1504

(The `Contract` column was removed due to redundancy).

The `permit()` function is impacted as users are forced to sign data according to an incorrectly implemented EIP-712 signature scheme. The domain separator is calculated using incorrect name and version values.

Recommendation: Calculate the hashed name and version values and store them in the expected storage slots.

3.2.6 Non-standard tokens can lead to silent failures

Severity: Medium Risk

Context: Vault.sol#L324, Bounty.sol#L125, Issuance.sol#L36

Description: Currently, the codebase does not adequately handle atypical ERC20 tokens when checking for invariant compliance. According to the `ERC20` specification, tokens should return "false" when a transfer fails, but it does not guarantee that the function will revert. This discrepancy could potentially lead to silent failures, making it difficult to detect issues when they occur.

In the configuration file, LDO (Lido DAO) token is mentioned.

From the code snippet below, It can be seen that LDO does not revert if the user balance is not enough for the transfer:

```
// ...
var previousBalanceFrom = balanceOfAt(_from, block.number);
if (previousBalanceFrom < _amount) {
    return false;
}
// ...
```

If there are ever rounding issues in the protocol, funds will be unrecoverable because the transfer failure is not properly handled. In these cases, enforcing transfers to revert upon failure ensures users are protected as they can simply redeem less index tokens in the next call.

Recommendation: To ensure proper handling of atypical tokens and compliance with ERC20 standards, it is advisable to incorporate OpenZeppelin's `SafeTransferLib`. This library is specifically designed to handle edge cases and provide a consistent behavior for token transfers.

3.2.7 Redemption failure risk due to token blacklisting

Severity: Medium Risk

Context: Issuance.sol#L79

Description: The `redeem` function in the contract allows users to redeem an amount of the underlying `ERC20` tokens stored in the vault. However, the current implementation does not account for the possibility that some of these `ERC20` tokens may have a blacklist mechanism. If a user is on the blacklist of any of these tokens, the redemption will fail, affecting the user's ability to redeem any token from the vault.

- Example token list:
  - USDC
  - Tether

```
function redeem(uint256 amount) external invariantCheck {
    vault.tryInflation();
    TokenInfo[] memory tokens = vault.realUnits();

    require(tokens.length > 0, "No tokens in vault");

    IVault.InvokeERC20Args[] memory args = new IVault.InvokeERC20Args[](
        tokens.length
    );

    for (uint256 i; i < tokens.length; ) {
        uint256 underlyingAmount = fmul(tokens[i].units, amount);

        args[i] = IVault.InvokeERC20Args({
            token: tokens[i].token,
            to: msg.sender,
            amount: underlyingAmount
        });

        unchecked {
            ++i;
        }
    }

    vault.invokeBurn(msg.sender, amount);

    vault.invokeERC20(args);
}
```

Recommendation: Modify the `redeem` function to ensure that if a user is blacklisted by one of the `ERC20` tokens, their ability to redeem other tokens is not compromised.

3.2.8 Low governance proposal threshold creates potential for malicious spam attacks

Severity: Medium Risk

Context: Config.sol#L18

Description: The current governance contract's `PROPOSAL_THRESHOLD` is set at a low value of `100`, making the system highly susceptible to spamming attacks. Malicious actors can easily meet this low threshold to create numerous disruptive proposals. An attacker with tokens just meeting the `PROPOSAL_THRESHOLD` can continually submit malicious or nonsensical proposals, disrupting the governance process.

- With the current low vote requirement, someone with bad intentions can easily create fake or harmful proposals. This can mess up the system and cause confusion.
- With lots of fake proposals, it's harder to pay attention to the relevant ones that can actually make a difference.
- If people see lots of fake or harmful proposals, they may lose trust in the system.

```
// governor measured in blocks
uint256 constant AVG_BLOCK_TIME = 12; // seconds
uint256 constant VOTE_DELAY = 1 hours / AVG_BLOCK_TIME; // 10 minutes
uint256 constant VOTE_PERIOD = 4 days / AVG_BLOCK_TIME; // 20 minutes
uint256 constant PROPOSAL_THRESHOLD = 100; // Number of votes required to create a proposal
uint256 constant GOVERNOR_NUMERATOR = 10;
```

Recommendation: It is recommended to increase the proposal threshold to a higher number to add a layer of protection against spamming and other types of abuse.

3.3 Low Risk

3.3.1 Users can avoid rebalance fees by sandwiching bounty fulfillments

Severity: Low Risk

Context: Bounty.sol#L83-L136

Description: To incentivize bounty fulfillments, the bounty hash must set nominals to a value slightly less than the intended amount. Because this rebalance does not happen often and the incentive will most likely exceed at least a day of fees, sophisticated users can monitor for these events and redeem their tokens right before a rebalance. After the rebalance has happened, they will be able to issue new tokens at a slightly better rate.

Recommendation: Adding lockout periods for when index tokens can be redeemed or issued only creates potential for DoS attacks. Alternatively, this can be acknowledged and monitored and the `Bounty` contract can be re-designed if there is sufficient evidence to suggest that this is being done at a large scale.

3.3.2 Nominal units for underlying tokens need to be updated before first bounty proposal

Severity: Low Risk

Context: Config.sol#L67-L88

Description: `Config.sol` has a list of 15 underlying tokens meant to be updated via the first bounty proposal during the upgrade. The nominal amounts for all of them are currently set to `1` as a placeholder with the following comment:

```
// The amounts will be determined shortly before the bounty is proposed.
// The goal is to have the bounty be equivalent the net asset value of AMKT at the time of proposal.
// 15 assets to be included in the index
```

Not updating these values during migration will not make the underlyings reflect the NAV of `AMKT` and effectively cause a loss of funds to the holders or protocol.

Recommendation: Ensure that these amounts are accurately determined and updated before the bounty is proposed during migration.

3.3.3 Migration will fail if `tryInflation` attempts to mint tokens

Severity: Low Risk

Context: __1__MultisigStep1.s.sol#L29, __2__MultisigStep2.s.sol#L19-L23, Bounty.sol#L96

Description: If during migration there is any accrued fee between contract deployment and bounty fulfilment, the call to `fulfillBounty()` will fail as the vault attempts to mint index tokens for which it is not yet the minter.

Currently, `feeScaled` will be set to `FEE_SCALED` as defined by `Config.sol`. As a result, inflation will immediately be accounted for even before the initial bounty to migrate assets over has been fulfilled.

Recommendation: Ensure that `feeScaled` is set to zero upon vault deployment and do not set the fee until the index token has been upgraded to v2.

3.3.4 `CoreDeployScript` executed by a non-privileged deployer address is risky

Severity: Low Risk

Context: __0__CoreDeploy.s.sol

Description: The protocol migration uses three scripts: `CoreDeployScript`, `MultisigStep1Script` and `MultisigStep2Script`. Of these, `CoreDeployScript` is allowed to be executed by any non-privileged deployer address via forge script while the other two are meant to be executed by the protocol MultiSig via Gnosis Safe transaction bundle. Moreover, the various actions in `CoreDeployScript` are not expected to be atomic given the usage.

The non-privileged deployer executing `CoreDeployScript` has temporary ownership rights on the newly created, but empty, vault whose ownership is then proposed to the MultiSig. It also has the `TIMELOCK_ADMIN_ROLE` which is correctly revoked.

The rationale for the `CoreDeployScript` to be executed by any non-privileged deployer address is to prefer the use of forge script over the MultiSig which may be error-prone.

Recommendation: Given the risks involved in the upgrade, the one-time nature of this migration and that the later two scripts require the use of MultiSig anyway, it may be worthwhile to reconsider this trade-off and require MultiSig for `CoreDeployScript` too.

3.3.5 Token holders can race to pass malicious proposals during protocol upgrade

Severity: Low Risk

Context: Governor.sol

Description: With the protocol upgrade from v1 to v2 of index token, existing token holders get voting powers on governance because v2 index token is `ERC20VotesUpgradeable` which keeps a history with checkpoints of each account's voting power. As mentioned in the OZ documentation:

> « By default, token balance does not account for voting power. This makes transfers cheaper. The downside is that it requires users to delegate to themselves in order to activate checkpoints and have their voting power tracked. »

This means that existing token holders who race to self-delegate their tokens to activate checkpoints may end up with majority voting power to propose and vote on potentially malicious proposals, which will require the protocol MultiSig to detect and veto such proposals.

Recommendation: Consider setting `VOTE_DELAY` to a much greater value, e.g. `1 weeks`, than the current `uint256 constant VOTE_DELAY = 1 hours` for the initial upgrade period to allow v1 token holders time to delegate. After that, governance can reset this to `1 hours`. Review the trade-off with any initial governance proposals, e.g. acceptance of vault ownership by governance from MultiSig, that could be affected by this long delay.

3.3.6 Issue/redeem functions are missing `ReentrancyGuard`

Severity: Low Risk

Context: Issuance.sol

Description: It may be possible to bypass `invariantCheck()` by taking advantage of any token callbacks from `Bounty.fulfillBounty()` or `issue()/redeem()` to increase `IERC20(tokens[i].token).balanceOf(address(this))` which is used to calculate `expectedAmount`. Because tokens are not minted or burnt until after the transfer has been made, the `invariantCheck()` may be unpredictable at enforcing the vault invariant.

Recommendation: Consider implementing a shared reentrancy guard that would prevent cross-contract calls between the `Issuance` and `Bounty` contracts.

3.3.7 Reimplementing widely-used and battle-tested libraries is risky

Severity: Low Risk

Context: ProposableOwnable.sol, M-03

Description: `ProposableOwnable.sol` aims to reimplement the widely-used `Ownable2Step.sol` library from OpenZeppelin to add two-step ownership transfer capability to `Ownable`. However, there are a few unnecessary checks, a missing event and a missing address reset which was identified and thereafter fixed from a previous security review. Some of these issues were raised in the previous review as well and are highlighted below:

- There is no need for `newOwner` parameter because this is required to be the `msg.sender`.

```
function transferOwnership(address newOwner) public virtual override {
```

- There is no need for this check because `msg.sender` can never be zero address.

```
require(
    newOwner != address(0),
    "ProposableOwnable: new owner is the zero address"
);
```

- These checks can effectively be replaced

```
require(
    newOwner == proposedOwner,
    "ProposableOwnable: new owner is not proposed owner"
);
require(
    newOwner == msg.sender,
    "ProposableOwnable: this call must be made by the new owner"
);
```

with the code snippet below using the custom error `OwnableUnauthorizedAccount` which is already defined in `Ownable`:

```
 if (proposedOwner != msg.sender) {
    revert OwnableUnauthorizedAccount(msg.sender);
 }
```

- Missing `OwnershipTransferStarted` event emission in `proposeOwner()` as done in Ownable2Step. This is recommended given the criticality of ownership transfer.

Recommendation: Consider replacing `ProposableOwnable.sol` with the use of `Ownable2Step.sol`.

3.3.8 `AMKT` minter role can be taken over during token migration

Severity: Low Risk

Context: __2__MultisigStep2.s.sol#L19-L23, IndexToken.sol#L38-L44

Description: The protocol attempts to migrate assets in two steps; firstly, core contracts are deployed (i.e. the vault, issuance, bounty and governance contracts) and then assets are transferred into the vault via bounty fulfilment for which the proxy contract representing the `AMKT` token is also upgraded to v2.

During this last step, the proxy admin contracts sets the new index token implementation contract and then initializes the vault as the sole minter of index token v2. However, it is possible for anyone to front-run the `IndexToken(_amkt).initialize(_vault)` call, takeover the minter role, burn all existing `AMKT` tokens and mint their own to drain the vault.

It is also worth noting that the implementation contract can have it's `initialize()` function front-run, making it vulnerable to unauthorized users taking over the minter role for this contract. However, because the minter role is unable to brick the implementation contract, it is in no way exploitable.

Recommendation: Update step 2 of the migration script to call `upgradeAndCall()` instead of `upgrade()`. This will ensure the v2 token is initialized before atomically. However, the Alongside team intends to make use of Gnosis safe's transaction builder feature which allows for transactions to be batched and executed atomically.

It is also recommended to call `_disableInitializers()` within the constructor to prevent any calls to initialize the implementation contract.

3.3.9 Gas limit exceedance risk due to large token array processing

Severity: Low Risk

Context: Bounty.sol#L22-L27

Description: In the Bounty contract, there is a potential gas limit exceedance risk associated with processing a large token array. The contract defines several data structures and functions to fulfill bounties, and it handles multiple tokens within a given bounty. When a bounty includes a substantial number of tokens, the gas required to process them all may exceed the gas limit.

Due to the gas limitations on the network, when the number of tokens in the bounty is sufficiently high, the transaction may fail to execute due to exceeding the gas limit.

```
function redeem(uint256 amount) external invariantCheck {
    vault.tryInflation();
    TokenInfo[] memory tokens = vault.realUnits();

    require(tokens.length > 0, "No tokens in vault");

    IVault.InvokeERC20Args[] memory args = new IVault.InvokeERC20Args[](
        tokens.length
    );

    for (uint256 i; i < tokens.length; ) {
        uint256 underlyingAmount = fmul(tokens[i].units, amount);

        args[i] = IVault.InvokeERC20Args({
            token: tokens[i].token,
            to: msg.sender,
            amount: underlyingAmount
        });

        unchecked {
            ++i;
        }
    }

    vault.invokeBurn(msg.sender, amount);

    vault.invokeERC20(args);
}
```

Recommendation: To mitigate the gas limit exceedance risk when handling large token arrays, consider implementing the following:

- Divide the processing of tokens into smaller, manageable batches to reduce the gas consumption in a single transaction. This approach allows for gradual processing of tokens without hitting the gas limit.
- Limit the token array size.

3.3.10 Transfer limitations due to 96-bit data type in some ERC20 tokens may affect interoperability

Severity: Low Risk

Context: Vault.sol#L324

Description: Some tokens, such as UNI, use `uint96` for balances and transfer amounts rather than the more standard `uint256`. This poses an issue for contracts that interact with such tokens and may assume that `uint256` is always safe to use. For instance, the `_transferTokens` function in UNI's contract reverts if the value passed for the amount is larger than ‘uint96‘. If a contract uses `uint256` for handling the transfer amount and it exceeds the `uint96` limit, then this will cause the transaction to fail.

```
function _transferTokens(address src, address dst, uint96 amount) internal {
    require(src != address(0), "Uni::_transferTokens: cannot transfer from the zero address");
    require(dst != address(0), "Uni::_transferTokens: cannot transfer to the zero address");

    balances[src] = sub96(balances[src], amount, "Uni::_transferTokens: transfer amount exceeds balance");
    balances[dst] = add96(balances[dst], amount, "Uni::_transferTokens: transfer amount overflows");
    emit Transfer(src, dst, amount);

    _moveDelegates(delegates[src], delegates[dst], amount);
}
```

Recommendation: Before making `approve` or `transfer` calls to such tokens, validate that the amount does not exceed the maximum value for `uint96`.

3.4 Gas Optimization

3.4.1 Adding a sanity check of requiring `amount > 0` for issue/redeem will avoid unnecessary execution

Severity: Gas Optimization

Context: Issuance.sol#L27-L45, Issuance.sol#L53-L80

Description: If a user accidentally attempts to issue/redeem zero amounts of index token then the implementation does not have a sanity check to revert early in this scenario, which leads to unnecessary execution of mint/burn logic. Issual even transfers a single token of each underlying from the user.

Recommendation: Adding a sanity check of requiring `amount > 0` for issue/redeem will avoid unnecessary execution and save gas.

3.4.2 Increment for loop variable can be placed in an `unchecked` block

Severity: Gas Optimization

Context: Bounty.sol#L124, Bounty.sol#L155, Bounty.sol#L185, Bounty.sol#L264, Vault.sol#L153, Vault.sol#L189, Vault.sol#L222, Vault.sol#L247, Vault.sol#L291, Issuance.sol#L33, Issuance.sol#L63, Issuance.sol#L85, VArray.sol#L75

Description: Increment for loop variables can be placed in an `unchecked` block to save gas by avoiding Solidity's (0.8+) default overflow check for unsigned integer arithmetic. While the codebase uses this optimization in some places, there are other loops where this is missing.

`i++` involves checked arithmetic, which is not required. This is because the value of `i` is always strictly less than `length <= 2**256 - 1`. Therefore, the theoretical maximum value of `i` to enter the for-loop body is `2**256 - 2`. This means that the `i++` in the for loop can never overflow. Regardless, the overflow checks are performed by the compiler. Using the `unchecked` block avoids this unnecessary check and saves some gas.

One can manually do this by:

```
for (uint i = 0; i < length; ) {
    // do something that doesn't change the value of i
    unchecked {
        ++i;
    }
}
```

Recommendation: Consider incrementing the `for` loop variable in an `unchecked` block like in other places for consistency and saving gas.

3.4.3 Loop iteration length can be cached

Severity: Gas Optimization

Context: Bounty.sol#L124, Bounty.sol#L155, Bounty.sol#L185, Bounty.sol#L264, Vault.sol#L153, Vault.sol#L291, Issuance.sol#L33, Issuance.sol#L63, Issuance.sol#L85

Description: Caching the length of memory arrays that are iterated in `for` loops will save gas by avoiding repetitive `mloads`. For example, in the below toy code snippet:

```
pragma solidity 0.8.19;
contract Test {
    uint256 public test;

    function increment() public {
        uint[] memory a = new uint[](500);
        // uint256 len = a.length;
        // for (uint i; i < len; i++) {
        for (uint i; i < a.length; i++) {
            test++;
        }
    }
}
```

caching `a.length` in a stack variable `len` and using that in the `for` loop check yields some gas savings as shown below:

```
with caching:
gas    360293 gas
transaction cost    313298 gas
execution cost    292234 gas

without caching:
gas    362003 gas
transaction cost    314785 gas
execution cost    293721 gas
```

Recommendation: Cache the length of memory arrays in a variable and use that in the conditional check of `for` loops.

3.5 Informational

3.5.1 Missing call to `__ERC20_init()` in `IndexToken.initialize()`

Severity: Informational

Context: IndexToken.sol#L38-L44

Description: `IndexToken.sol` derives from `ERC20VotesUpgradeable` but is missing a call to `__ERC20_init()` in `initialize()`.

Recommendation: Consider initializing via `__ERC20_init(string memory name_, string memory symbol_)`.

3.5.2 Differentiating `invokeSetNominal()` versions will improve readability

Severity: Informational

Context: Vault.sol#L150-L162

Description: There are two versions of `invokeSetNominal()` where one takes an array of `SetNominalArgs` and another takes a single `SetNominalArgs`. Both versions having the same name is confusing. Differentiating `invokeSetNominal()` versions will improve readability.

Recommendation: Consider renaming the version which takes an array to `invokeSetNominals()`.

3.5.3 Missing zero address checks for critical addresses

Severity: Informational

Context: Vault.sol#L76-L80, Vault.sol#L89-L105, Bounty.sol#L67-L69, ActiveBounty.sol#L11

Description: Critical protocol addresses are missing zero-address checks in constructors and setters. Accidental use of zero addresses will cause operational disruptions.

Recommendation: Consider adding zero-address checks for critical addresses.

3.5.4 `VaultInvariant()` can be moved to `IVault` for consistency

Severity: Informational

Context: Vault.sol#L15, IVault.sol#L6-L10

Description: Custom errors for Vault are declared inside `IVault.sol` except for `VaultInvariant()` which is in `Vault.sol`.

Recommendation: `error VaultInvariant()` can be moved to `IVault` for consistency.

3.5.5 `indexToken.totalSupply()` invariant check can be made specific to callback scenario

Severity: Informational

Context: Bounty.sol#L115-L121, H-01

Description: `indexToken.totalSupply()` invariant check was added in PR#45 per recommendation for H-01 from the previous security review.

However, the callback has since been made an opt-in via parameterization of `fulfillBounty()`. Therefore, the invariant check can be made specific to the callback present scenario.

Recommendation: Move the invariant check inside the conditional for callback:

```
if (callback) {
    Rebalancer(msg.sender).rebalanceCallback(ins, intoTokenInfo(outs));
    if (indexToken.totalSupply() != startingSupply) {
        revert BountyAMKTSupplyChange();
    }
}
```

3.5.6 Duplication of code affects readability

Severity: Informational

Context: Bounty.sol#L154-L157, Bounty.sol#L259-L267

Description: `intoTokenInfo()` is a helper function which converts `InvokeERC20Args` to `TokenInfo` format. However, `quoteBounty()` duplicates this conversion logic which is unnecessary and affects readability.

Recommendation: Replace duplicated code with a call to `intoTokenInfo()`.

3.5.7 Missing/incomplete Natspec can affect readability and UX

Severity: Informational

Context: Vault.sol#L67, Vault.sol#L89, Vault.sol#L93, Vault.sol#L97, Vault.sol#L101, Vault.sol#L107, Vault.sol#L116, Vault.sol#L123, Vault.sol#L150, Vault.sol#L158, Vault.sol#L164, Vault.sol#L172, Vault.sol#L179, Vault.sol#L185-L198, Vault.sol#L202, Vault.sol#L232, Issuance.sol#L82, Bounty.sol#L138

Description: Several functions have missing/incomplete Natspec which affect readability and UX.

Recommendation: Consider adding Natspec to the functions.

3.5.8 BOUNTY_DEADLINE is set to an unnecessarily high value

Severity: Informational

Context: Config.sol#L10, __1__MultisigStep1.s.sol#L18-L29

Description: Config.sol sets the BOUNTY_DEADLINE value to 14 days which is not required because this bounty is set and immediately fulfilled as part of __1__MultisigStep1.

Recommendation: While this does not introduce any immediate risk, if there are any future changes to __1__MultisigStep1 then this may allow the bounty fulfilment to not happen immediately and open the protocol to risk from the upgraded index token not reflecting the new underlyings and their nominals accurately.

3.5.9 Use custom errors instead of revert strings

Severity: Informational

Context: Issuance.sol#L31

Description: To save some gas the use of custom errors leads to cheaper deploy time cost and run time cost. The run time cost is only relevant when the revert condition is met.

Recommendation: Consider using custom errors instead of revert strings.

3.5.10 VaultInvariant can be moved to IVault for consistency

Severity: Informational

Context: Vault.sol#L15, IVault.sol#L7-L10

Description: Vault related custom errors are declared in IVault except VaultInvariant which is in Vault.sol.

Recommendation: VaultInvariant can be moved to IVault for consistency.

3.5.11 Stale comments in Config.sol affect readability

Severity: Informational

Context: Config.sol#L16-L17

Description:

```
uint256 constant VOTE_DELAY = 1 hours / AVG_BLOCK_TIME; // 10 minutes
uint256 constant VOTE_PERIOD = 4 days / AVG_BLOCK_TIME; // 20 minutes
```

The comment about 10 and 20 minutes do not correspond to values set for VOTE_DELAY and VOTE_PERIOD.

Recommendation: Remove stale comments.

3.5.12 Unused function overrides carried over from legacy requirements

Severity: Informational

Context: IndexToken.sol#L76-L105

Description: Previous version of Index token had transfer restriction logic which necessitated the overriding of their transfer functions. However, this has since been removed and the overridden functions now are the same as those in ERC20Upgradeable making these redundant.

Recommendation: Remove the redundant overridden functions to increase readability and avoid any unintended logic mismatch.

3.5.13 Missing events in privileged functions

Severity: Informational

Context: IndexToken.sol#L38-L44, ActiveBounty.sol#L14-L17, Vault.sol#L89-L118, Vault.sol#L150-L168, Vault.sol#L123-L146, Bounty.sol#L83-L136

Description: Several privileged functions that change critical protocol addresses/parameters or implement key functionality are missing event emissions. This will affect offchain monitoring/tooling capability.

Recommendation: Add event emissions to all privileged functions that change critical protocol addresses/parameters or implement key functionality.

3.5.14 Unused imports carried over from legacy requirements

Severity: Informational

Context: IndexToken.sol#L4-L5

Description: There are imports carried over from legacy requirements which are no longer required.

Recommendation: Remove unused imports.

3.5.15 Reevaluate salt generation method in the deployment script

Severity: Informational

Context: __1__MultisigStep1.s.sol#L20

Description: In the deployment script, the salt for the `Bounty` is generated using `keccak256(abi.encode(block.timestamp))`. While this approach is commonly used for salt generation, it's essential to assess whether it's the most suitable method for this specific context.

Using `block.timestamp` as a source of randomness for salt generation might be predictable in certain situations.

Recommendation: Consider exploring alternative salt generation methods that provide better unpredictability and uniqueness.

3.5.16 Typographical errors in the codebase affect readability

Severity: Informational

Context: Bounty.sol#L12, Bounty.sol#L73-L78, Governor.sol#L26, Issuance.sol#L26

Description: Across the codebase, there are typographical errors in the comments:

- `recieved` should be `received`
- `were` should be `we're`
- `becasue` should be `because`
- `smae` should `same`
- `requies` should be `requires`
- `Alongside Governer` to `Alongside Governor`

Recommendation: Consider correcting the errors and spellcheck the codebase in future to improve code readability.

3.5.17 Switch to Openzeppelin's IERC20 interface for standardization

Severity: Informational

Context: Bounty.sol#L3

Description: The current `Bounty` contract imports the *IERC20* interface from a custom path `"forge-std/interfaces/IERC20.sol"`.

Recommendation: Replace the custom `IERC20` import with OpenZeppelin's `IERC20` interface.

3.5.18 Self-delegation for Erc20VotesUpgradeable tokens

Severity: Informational

Context: IndexToken.sol#L13

Description: The OpenZeppelin's ERC20VotesUpgradeable contract does not automatically account for voting power when tokens are transferred or minted. By default, users must manually delegate their tokens to themselves to activate voting capabilities, which adds an extra step in the process and may hinder user experience.

According to the OpenZeppelin documentation:

> « By default, token balance does not account for voting power. This makes transfers cheaper. The downside is that it requires users to delegate to themselves in order to activate checkpoints and have their voting power tracked. »

Recommendation: To streamline the user experience and encourage participation in governance activities, it is proposed that automatic self-delegation should be implemented whenever tokens are transferred to a new address for the first time. This can be done by overriding the `_afterTokenTransfer` function within the `ERC20VotesUpgradeable` contract as follows:

```
function _afterTokenTransfer(address from, address to, uint256 amount) internal override {
    super._afterTokenTransfer(from, to, amount);

    // Automatically turn on delegation on mint/transfer but only for the first time.
    if (to != address(0) && numCheckpoints(to) == 0 && delegates(to) == address(0)) {
        _delegate(to, to);
    }
}
```

Nevertheless, while this approach makes it easier for users by removing the need to manually delegate tokens for voting, it will increase the base gas cost for first-time token transfers. This trade-off should be evaluated based on the specific requirements and user behavior associated with the token in question.

3.5.19 Discrepancy in pragma versioning is risky

Severity: Informational

Context: Governor.sol#L2

Description: The use of different pragma versions in the contracts can present several implications, with potential risks and compliance concerns that need to be addressed to maintain robust and compliant contracts.

Recommendation: Consider locking pragma like other contracts.

3.5.20 Redundant events on the interface affect readability

Severity: Informational

Context: IIndexToken.sol#L7-L15

Description: The new implementation of the `IIndexToken` interface in the IndexToken contract does not make use of event emissions as specified in the interface. Events like `MinterSet`, `SupplyCeilingSet`, `MintFeeToReceiver`, and `ToggledRestricted` are declared in the interface but are not emitted in the associated functions in the contract.

```
// ...
event MinterSet(address indexed minter);
event SupplyCeilingSet(uint256 supplyCeiling);
event MintFeeToReceiver(
    address feeReceiver,
    uint256 timestamp,
    uint256 totalSupply,
    uint256 amount
);
event ToggledRestricted(address indexed account, bool isRestricted);
// ...
```

Recommendation: Consider deleting redundant events to improve readability.

3.5.21 Risk of bounty hash collision in multichain environments

Severity: Informational

Context: Bounty.sol#L246

Description: The current implementation of the `hashBounty()` function is not designed to handle a multichain environment. This may open the door to hash collision attacks. As the protocol may evolve to operate on multiple chains, it is prudent to prepare for such a scenario in advance.

Recommendation: It is advisable to include chain-specific identifiers in the hash generation process.