Alongside Finance

Index Smart Contracts

Security Assessment Report

Version: 2.1

September, 2022

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Alongside Finance index smart contracts. The review focused solely on the security aspects of the Solidity implementation of the contract, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the Alongside Finance index smart contracts contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the Alongside Finance smart contracts.

Overview

The Alongside Crypto Market Cap (AMKT) Token is an ERC20 token designed to be fully redeemable for its underlying collateral held in multiple cryptocurrencies and digital assets, serving as an index of the broader crypto market.

The AMKT contracts are forked from WBTC and modified to:

- upgrade contracts to solidity 0.8.7 and utilize OpenZeppelin upgradeable libraries
- support multiple underlying digital assets as collateral
- migrate custody from the protocol team to a third-party institutional custodian
- enable rebalancing of the index portfolio assets on a periodic basis
- generate protocol fees

The AMKT contracts provide a template for managing the minting, burning, and transfer operations of the token. However, significant portions of this activity occur off-chain: utilizing IPFS for data distribution and custodian APIs for the validation, transfer, and rebalancing of portfolio assets. While these off-chain processes were considered throughout the course of the review, they should not be considered within the scope of this security assessment.

Second Review Stage

The retesting and resolution stage of the review was merged into a review of a new version of the **OTC** contract. This second version is referred to in this document as **OTC.sol-2**, although the file name in the project remains **OTC.sol**.

The second miscellaneous findings section relates to this second review stage. The first miscellaneous findings section relates only to the first stage.

Security Assessment Summary

A first review was conducted on the files hosted on the Alongside-Finance index-contract repository and were assessed at commit 81f73cb.

A second subsequent review targetted commit 26be069.

Note: the OpenZeppelin libraries and dependencies were excluded from the scope of this assessment.

The manual code review section of the report is focused on identifying any and all issues/vulnerabilities associated with the business logic implementation of the contracts: specifically, their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout). Additionally, the manual review process focused on all known Solidity anti-patterns and attack vectors. These include, but are not limited to, the following vectors: re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers. For a more thorough, but non-exhaustive list of examined vectors, see [1, 2].

To support this review, the testing team used the following automated testing tools:

- Mythril: `https://github.com/ConsenSys/mythril`
- Slither: `https://github.com/trailofbits/slither`

Output for these automated tools is available upon request.

Findings Summary

The testing team identified a total of 19 issues during this assessment. Categorized by their severity:

- Critical: 1 issue.
- High: 1 issue.
- Medium: 6 issues.
- Low: 2 issues.
- Informational: 9 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Alongside Finance index smart contracts. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as "informational".

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
AMKT-01	Incorrect Requests Array Updated	Critical	Closed
AMKT-02	`owner` Can Block Users from Claiming their Tokens	High	Closed
AMKT-03	Lack of Usage of `safeTransferFrom` for External Token Interactions	Medium	Resolved
AMKT-04	Lack of Usage of `safeTransferFrom` for External Token Interactions (2)	Medium	Closed
AMKT-05	Suboptimal Token Distribution Model	Medium	Closed
AMKT-06	Compromised Validator Can Validate Unbalanced Burn Requests	Medium	Closed
AMKT-07	No Check on `finalizeRequest().toRefund` Value Could be Exploited by a Compromised Validator	Medium	Closed
AMKT-08	Duplicated Mint Requests May Be Confirmed	Medium	Closed
AMKT-09	Merchants Can Mint Requests Over the Supply Ceiling	Low	Closed
AMKT-10	OTC Mint Requests Over Merchant Limits Are Allowed	Low	Resolved
AMKT-11	Limits Only Enforced Per Mint or Burn	Informational	Resolved
AMKT-12	Inconsistent Indices in `IndexedMapping`	Informational	Closed
AMKT-13	Inflation Rate Varies With Frequency of Calls to `_mintToFeeReceiver()`	Informational	Resolved
AMKT-14	Enums With Consequential First Values Can Cause Unexpected Behaviour	Informational	Resolved
AMKT-15	Use Checks-Effects-Interactions Pattern to Reduce Reentrancy Risk	Informational	Closed
AMKT-16	OTC is Dependant on `merchantMintLimit` and `merchantBurnLimit` in `Factory.sol`	Informational	Closed
AMKT-17	Miscellaneous General Comments	Informational	Resolved
AMKT-18	Miscellaneous General Comments - Second Review Stage	Informational	Closed
AMKT-19	Impact of Key Compromise for Privileged Roles	Informational	Closed

AMKT-01	Incorrect Requests Array Updated		
Asset	`OTC-2.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

On line [**435**] of `otc.sol-2`, the state variable `requests[nonce].status` is updated. However, this is the mint requests array. The variable `burnRequests[nonce].status` is the appropriate variable.

This bug has the potential to allow multiple token claims for the same rejected burn request.

Recommendations

Change `requests[nonce].status` to `burnRequests[nonce].status` on line [**435**] of `otc.sol-2`.

Resolution

The affected contract is no longer in use.

AMKT-02	owner Can Block Users from Claiming their Tokens		
Asset	OTC-2.sol		
Status	**Closed:** See Resolution		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

The role `owner` has free access to add and remove addresses from the `users` indexed mapping. This indexed mapping is an access check for the functions that claim tokens from completed mint and burn processes: `otc.claimIndexToken()` for confirmed mints and `otc.refundIndexToken()` for rejected burns.

A malicious owner can call `otc.removeUser()` at a point where a mint or burn process has been started, but no call has been made to `otc.claimIndexToken()` or `otc.refundIndexToken()`. This will then result in the targetted user being unable to withdraw their tokens because of the `onlyUser` modifier checks on those two functions.

This attack is somewhat mitigated by the owner's inability to directly profit, and thereby having limited motivation to carry it out.

Recommendations

`otc.claimIndexToken()` and `otc.refundIndexToken()` already contain access control checks: they both `require` that `msg.sender` is equal to `request.requester`. This is a check which is unaffected by the actions of `owner` but is also highly restrictive: only a single address is allowed. It is therefore recommended to simply remove the `onlyUser` modifier from `otc.claimIndexToken()` and `otc.refundIndexToken()` (but to keep it in the other places where it is used).

Resolution

The affected contract is no longer in use.

AMKT-03	Lack of Usage of `safeTransferFrom` for External Token Interactions		
Asset	`OTC.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

There is a potentially unsafe call to `PaymentToken.transferFrom()` on line [**128**] of `OTC.sol`.

```
/// @notice Requests IndexToken for PaymentToken
/// @param paymentTokenAmount amount of PaymentToken to deposit
/// @param indexTokenAmount amount of IndexToken to receive
function requestIndexToken(uint256 paymentTokenAmount, uint256 indexTokenAmount)
  external
  onlyUser
  returns (uint256)
{
  PaymentToken.transferFrom(msg.sender, depositAddress, paymentTokenAmount); // reverts on failure
  requests.push(
    Request(msg.sender, paymentTokenAmount, indexTokenAmount, RequestStatus.PENDING, 2**256 - 1, depositAddress)
  );
  uint256 nonce = requests.length - 1;
  emit RequestIndexToken(nonce, msg.sender, depositAddress, paymentTokenAmount, indexTokenAmount);
  return nonce;
}
```

Although this call is expected to revert on failure, this cannot be assumed of an arbirary `paymentToken` which may not fully comply with `ERC20` specifications. As such, failed payments may manage to successfully initiate an index token request.

Recommendations

Use OpenZeppelin's `SafeERC20` wrapper function on line [**128**] of `OTC.sol`.

Alternatively, new `OTC` deployments for each `paymentToken` should be carefully reviewed and whitelisted by the protocol team. This reduces gas costs on transfers, but retains an avoidable long term risk in the contracts.

Resolution

OpenZeppelin's `SafeERC20` wrapper has been used.

✓ Resolved in commit [e988ab1]

AMKT-04	Lack of Usage of `safeTransferFrom` for External Token Interactions (2)		
Asset	`OTC-2.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

This issue is an extension of AMKT-03.

There are more instances of `PaymentToken.transfer()` or `PaymentToken.transferFrom()` on line [**174**], line [**293**] and line [**381**].

See AMKT-03 for more details.

Recommendations

Use OpenZeppelin's `SafeERC20` wrapper function for all transfers of `PaymentToken`.

Resolution

The affected contract is no longer in use.

AMKT-05	Suboptimal Token Distribution Model		
Asset	`OTC-2.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

`otc.sol-2` transfers tokens to a user in functions that can only be called by the validator on line [**293**] and line [**381**].

Where possible, it is preferable to use a pull model of token transfer (*ie. to have the tokens transferred inside function calls that are made by the users*).

There are multiple disadvantages to the push model (*ie. 'pushing' the tokens out in calls made by roles controlled by the protocol*):

1. **Passing off control -** Fundamentally, all issues stem from the fact that external calls to token contracts by protocol EOAs involve some risk of code running that is not fully controlled or anticipated by the protocol. In this case, where the payment token is expected to always be USDC, there is a low risk of unexpected code execution, but this cannot be guaranteed. It is possible that a different token might some day be used or that USDC might introdice modification to the token contract. For best security, external calls to contracts not controlled by the protocol should always be treated as uncontrolled.
2. **Runaway gas costs -** One issue this may cause is the unexpected execution of additional instructions, the gas costs of which would come from the validator account.
3. **Execution blocking -** If the external call reverts, the `otc.sol-2` contract would be blocked from progressing the relevant request. This is very low impact for this contract, as other requests would be unaffected.
4. **Hijacking of tx.origin -** If the validator account interacts with any code which checks `tx.origin` to control access, external calls to transfer certain tokens can be used to exploit that code.

Additionally, there is a transfer of the index token on line [**352**] which also fits this pattern. The index token is protocol controlled, and so does not strictly create this issue. The development team needs to be keenly aware, however, that this only remains the case whilst there is absolutely no way that transfers of the index token could execute uncontrolled code.

Recommendations

Create a mapping of user balances for the payment token (and possibly the index token, if desired), and convert the transfers mentioned above to modifications of this variable. Then add a withdraw function which allows users to withdraw their balance in payment tokens.

Resolution

The affected contract is no longer in use.

AMKT-06	Compromised Validator Can Validate Unbalanced Burn Requests		
Asset	`OTC-2.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

The development team requested in the documents that the scenario of a compromised validator be considered. The validator role is a powerful one with a lot of security implications. However, it is also a multisig protected role. This is one way that a malicious validator could steal funds. It requires collusion with any address listed in the `users` indexed mapping.

At any time when there are significant numbers of payment tokens present in the `otc.sol-2` contract, a user can call `otc.requestBurn()` to burn index tokens for a large number of payment tokens. The compromised/rogue validator can then call `processBurn()` and `finalizeBurn()`. When calling `finalizeBurn()`, the `actual` parameter can then be maximised to within the limits allowed by `maxDivergenceBP` on line [376]. Any difference in tokens between this value and the actual number of returned payment tokens will be taken from the balance of `otc.sol-2`, at the expense of other users.

Additionally, a compromised/rogue validator could block processing mint requests or finalising burn requests from other users, thereby trapping additional payment tokens in the contract to be harvested through this method.

Recommendations

It is a question for the development team whether it is desirable to design around this issue at all. If security controls on the validators' private keys are extensive, it may not be worth the extra complexity required to build a new version of `otc.sol-2` that is more resistant to exploitation by the validator.

The main vulnerability is that the value of `actual` is unchecked. If it were confirmed with `factory`, then a validator would not be able to submit arbitrary values.

It is difficult to see how a version of the contract could be built which does not allow a compromised/rogue validator to block the progression of mint and burn requests, however.

Resolution

The affected contract is no longer in use.

AMKT-07	No Check on `finalizeRequest().toRefund` Value Could be Exploited by a Compromised Validator		
Asset	`OTC.sol-2`		
Status	**Closed:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

The development team requested in the documents that the scenario of a compromised validator be considered. The validator role is a powerful one with extensive, critical privileges. However, it is also a multisig protected role. This is one way that a malicious validator could steal funds. It requires collusion with any address listed in the `users` indexed mapping.

Whilst the contract checks other token amount parameters, `finalizeRequest().toRefund` is not checked for consistency with the deviation values or, more importantly, to ensure that it is not larger than the value of the requester's initial payment.

If the validator submitted an oversized value for `finalizeRequest().toRefund` whilst calling `finalizeRequest()` to confirm a request, it can then call `OTC.openMintRefund()` and send that amount in payment tokens to the relevant merchant.

Recommendations

In the positive confirmation section of `OTC.finalizeRequest()`, check the value of `toRefund` for compliance with the deviation settings and also ensure that it is no larger than the initial payment token deposit.

Resolution

The affected contract is no longer in use.

AMKT-08	Duplicated Mint Requests May Be Confirmed		
Asset	`Factory.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

There are no requirements in the contracts that prevent a `Merchant` from successfully calling `Factory.addMintRequest()` multiple times for the same deposit. It is also possible for the `Issuer` to subsequently confirm the duplicated requests.

Two identical calls can be made by a `Merchant` to `addMintRequest()`, which are both accepted. The values of `nonce` will differ between the requests, and so will `requestHash`. The parameter `txid` is not used for validation, so there is nothing in the contracts to ensure the requests refer to separate transactions.

For the same reason, a `Merchant` may also call `Factory.addMintRequest()` to create new mint requests for previously rejected or cancelled mint requests.

The system therefore relies on the validators to pick up duplicate mint requests to prevent tokens being minted twice in these situations.

Recommendations

The protocol could be changed to require `txid` to be used, although this is a significant design change.

Alternatively, if the team wishes to keep the current design, they should remain keenly aware of this issue and be sure that the `Issuer` validators have a robust system to avoid minting `AMKT` from duplicate requests.

Resolution

The development team indicated that the off-chain processes in place have robust systems to avoid processing duplicate mint requests.

AMKT-09	Merchants Can Mint Requests Over the Supply Ceiling		
Asset	`Factory.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

Calls to `Factory.addMintRequest()` do not check that the mint amount is below the `IndexToken.supplyCeiling` threshold. Therefore, a mint request over this limit may be initiated by a `Merchant`.

The pending mint request will eventually fail when `Factory.confirmMintRequest()` is called due to the requirement on line [**92**] of `IndexToken.mint()`. However, this pattern in asyncronous operations results in a negative experience for the `Merchant` initiating a mint request, who should have already deposited the underlying assets.

The full resolution of the issue after this point is outside the scope of this assessment.

Recommendations

The issue could be mitigated by requiring that mint amounts are below the `IndexToken.supplyCeiling` threshold within the `Factory.addMintRequest()` function.

Alternatively, the team could simply stay alert to this issue and ensure that `IndexToken.supplyCeiling` is never within reach of allowable mint requests.

Resolution

The development team acknowledged the issue and decided not to remediate, but instead to remain aware of the issue and communicate to merchants about the supply limit so as to avoid problems.

AMKT-10	OTC Mint Requests Over Merchant Limits Are Allowed		
Asset	`OTC.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

Calls to `OTC.requestIndexToken()` do not check that the mint amount is below the `Factory.merchantMintLimit` threshold. Therefore, a mint request over this limit may be initiated by a `Merchant`.

Instead, the `merchantMintLimit` of the `OTC` contract will be applied for all merchants.

Recommendations

The issue could be mitigated by requiring that mint amounts are below the `Factory.merchantMintLimit` of the `OTC` contract within the `OTC.requestIndexToken()` function.

Resolution

As recommended, `Factory.merchantMintLimit` is being checked within the `OTC.requestIndexToken()` function.

✓ Resolved in commit [6121e56]

AMKT-11	Limits Only Enforced Per Mint or Burn
Asset	`Factory.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

`merchantMintLimit` and `merchantBurnLimit` are only enforced for each request, therefore a `Merchant` may exceed these limits by submitting multiple requests.

Recommendations

Make sure that this behaviour is understood and intended.

If a total mint limit is preferred, mappings could be created to track the total value in `AMKT` that is currently open per merchant for each request type. These values would need to be incremented and decremented as the processes complete, and they would be tested against `merchantMintLimit` and `merchantBurnLimit`, adding the new request during the test.

Resolution

`Factory.merchantMintLimit` and `Factory.merchantBurnLimit` have been converted to work cumulatively.

✓ Resolved in commit [22c4525]

AMKT-12	Inconsistent Indices in `IndexedMapping`
Asset	`Members.sol, OTC.sol, IndexedMapping.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

The `IndexedMapping` is implemented for `Members.merchants` and `OTC.users`. Both of these variables have external functions that retrieve entries by index: `Members.getMerchant()` and `OTC.getUser()`. They also have external functions that retrieve their values in order of index: `Members.getMerchants()` and `OTC.getUsers()`.

The implementation of `IndexedMapping` changes the index values for some entries when other entries are removed.

This behavior could potentially break external applications that might rely on these external functions, if they are unaware of the facts that:

1. the index of a given entry may change
2. the sort order of the returned values may change

Recommendations

`Members.getMerchants()` and `OTC.getUsers()` could be modified to return sorted results. `Members.getMerchant()` and `OTC.getUser()` could be removed entirely.

If these measures are not desired, then the protocol team should clarify in the related documentation that external applications should not rely on the indices and sort orders remaining consistent.

Resolution

The development team acknowledged the issue and decided not to remediate, on the basis that external applications will not rely on the ordering or indexes of these return values.

AMKT-13	Inflation Rate Varies With Frequency of Calls to `_mintToFeeReceiver()`
Asset	`IndexToken.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The fees collected by `IndexToken._mintToFeeReceiver()` are dependent on the total supply of the token, yet also increase the total supply. Therefore, the effective fee rate is compounded whenever `_mintToFeeReceiver()` is called. More frequent calls will increase the effective fee rate.

`_mintToFeeReceiver()` is called with every burn or mint operation of the token. It may also be called via an external function, which `feeReceiver` would be incentivized to call frequently.

Recommendations

Be aware of this issue, particularly when calculating an increase in the fee rate or describing the fee rate to users.

The effective fee rate will be greater than the parameter chosen for `feeRateAnnualBips` due to variable compounding.

Resolution

`feeRateAnnualBips` has been converted to a daily rate which is rigidly calculated on a daily basis, such that the interest will not be calculated if less than a day has passed.

✓ Resolved in commit [e94e58a]

AMKT-14	Enums With Consequential First Values Can Cause Unexpected Behaviour
Asset	`Factory.sol, OTC.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The user-defined data type `enum` defaults to its first value, so it is good practice to define its first value as a null value that is never used in the code, i.e. a value that will fail any `require` checks performed on the zero-initialized value.

`Factory.sol` and `OTC.sol` have (different) enums called `RequestStatus`, both of which default to a value of `PENDING`.

The testing team did not identify a scenario in the current implementation of the protocol where this could cause an issue.

However, `Factory.sol` is upgradeable and does make checks against `RequestStatus.PENDING` for a given `requestHash` in `validatePendingRequest()`.

```
function validatePendingRequest(Request memory request, bytes32 requestHash) internal pure {
  require(request.status == RequestStatus.PENDING, "request is not pending");
  require(requestHash == calcRequestHash(request), "given request hash does not match a pending request");
}
```

This check could be expected to pass for a zero-initialized `request` struct and its corresponding `requestHash`, which has not yet been added via `addMintRequest` and marked as `PENDING`.

Recommendations

Add a first value `NULL` to both `RequestStatus` enums.

If this is not desired, remain aware of this potential issue if the implementation contract of `Factory.sol` is ever upgraded.

Resolution

As recommended, a first value `NULL` has been added to both `RequestStatus` enums.

✓ Resolved in commit [d725b4c]

AMKT-15	Use Checks-Effects-Interactions Pattern to Reduce Reentrancy Risk
Asset	`OTC-2.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

There are multiple instances of essential state variables being updated after external contract calls. Generally, this is a pattern that is vulnerable to reentrancy.

1. line [174]-line [179]

```
PaymentToken.transfer(
    depositAddress,
    request.paymentTokenAmount + request.deviation
    );

requests[nonce].status = RequestStatus.PROCESSING;
```

2. line [210]-line [219]

```
uint256 factoryNonce = factory.addMintRequest(
    request.indexTokenAmount,
    txid,
    depositAddressString
);

requests[nonce].factoryNonce = factoryNonce;
requests[nonce].refunded = toRefund;

requests[nonce].status = RequestStatus.CONFIRMED;
```

3. line [332]-line [337]

```
uint256 factoryNonce = factory.burn(request.indexTokenAmount);
burnRequests[_nonce].factoryNonce = factoryNonce;

getBurnRequestFromFactoryNonce[factoryNonce] = _nonce;

burnRequests[_nonce].status = RequestStatus.PROCESSING;
```

4. line [381]-line [382]

```
PaymentToken.transfer(request.requester, actual);
burnRequests[_nonce].status = RequestStatus.CONFIRMED;
```

In all cases, the essential gatekeeping `burnRequests[_nonce].status` variable is updated after the external call, potentially allowing the caller to call back into the function and repeat the external call, thereby transferring tokens multiple times, or initiating multiple factory interactions.

`requests[nonce].refunded` should also be updated before the external call.

In all of these cases, the code is protected by the `onlyValidator` modifier, so the likelihood of malicious action is very low. However, the development team did ask to consider scenarios where validators are compromised.

In two of these cases, the external call is to the payment token, and in two it is to the factory. There are tokens that can allow callbacks and thereby enable reentrancy. The factory is less likely to be manipulated unless the contract was incorrectly configured.

Recommendations

Modify the code in question to update `burnRequests[_nonce].status` and `requests[nonce].refunded` before making external calls.

Because of the extensive requirements of an exploitation scenario, this issue could simply be considered a matter of best practice. Because it is easily implemented, the change is still recommended.

Resolution

The affected contract is no longer in use.

AMKT-16	OTC is Dependant on `merchantMintLimit` and `merchantBurnLimit` in `Factory.sol`
Asset	`OTC-2.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

As the `otc.sol-2` contract makes calls directly to the `factory` contract, it needs to be registered as a `merchant` on `factory`, and is therefore subject to the mint and burn limit mechanisms of `factory`.

`factory.addMintRequest()` is called in `otc.finalizeBurn()` (rejected) and `otc.finalizeRequest()` (confirmed). `factory.addMintRequest()` checks `factory.merchantMintLimit` and reverts if the added request exceeds that limit. In this case, these functions in `otc.sol-2` would therefore be blocked until `factory.merchantMintLimit` for `otc.sol-2` was increased.

Similarly, `factory.burn()` is called in `otc.processBurn()`. `factory.burn()` checks `factory.merchantBurnLimit` and reverts if the added request exceeds that limit. In this case, `otc.processBurn()` would therefore be blocked until `factory.merchantBurnLimit` for `otc.sol-2` is increased.

Recommendations

This issue has been discussed with the development team, who expressed their intention to stay aware of the issue and monitor the relevant limits and increase them as necessary. The development team also noted that this behaviour can act as a mechanism whereby a `otc.sol-2` contract can be paused or shut down on the factory side: a desirable functionality.

Resolution

The affected contract is no longer in use.

AMKT-17	Miscellaneous General Comments
Asset	`contracts/*`
Status	**Resolved:** See Resolutions
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team:

1. **Frontrunning Rebalance:** As the protocol grows and the size of the assets in custody increases, it may be possible to monitor calls to `IndexToken.setMethodology()` and thereby predict what the rebalancing trades will be. If those trades are very large in volume, it is conceivable they might affect some asset prices and thus it may be possible to frontrun them for profit.
2. **Meta Transaction Support:** `transfer` and `transferFrom` are not implementing OpenZeppelin's `ContextUpgradeable` (as the orignal functions do) so support for meta transactions would require an implementation upgrade of this contract.
3. **Ownership Transfer:** OpenZeppelin's `Ownable` and `OwnableUpgradeable` allow ownership transfer with a single transaction, which opens up a risk of human error losing owner control of a vital contract. Propose/accept structures of ownership transfer can be used to avoid this issue.
4. **Token Name:** `DeployAMKT.sol` line [43] deploys the AMKT token with the name "Test".
5. **Issuer/Custodian Role Ambiguity:** The confusion over the `Issuer` role being known by the name of another role (`Custodian`) within the contracts seems unnecessarily ambiguous and could potentially cause issues in future through confusion. If the goal is to minimise differences with `WBTC`, one solution would be to refer to the role as (`Custodian`) in the documentation.
6. **Supply Ceiling and Inflation:** The Alongside documentation says, *"The supply ceiling ... MUST not apply to inflation."* Two interpretations of this should be distinguished. Supply ceiling does not block inflation from occurring in the instance where inflation will take the token supply above the supply ceiling. However, the supply ceiling does apply to inflation in the sense that all tokens previously created through inflation are counted towards the supply ceiling.
7. **Unused txid Parameter:** `Factory.addMintRequest()` has the parameter `txid` that is used in the OTC flow, but not in the standard mint flow. This effectively means that merchants can deposit any value they wish into the contract's storage. It is also used in determining `requestHash` values. This is not a desirable state of affairs from a security perspective. It is suggested that a mechanism be added to force `txid` to be empty, unless the sender is on a whitelist, which the `OTC` contract could be added to.
8. **Token Balance:** `OTC.requestIndexToken()` could, for added certainty, check its payment token balance before and after the payment token transfer.
9. **OTC and Shifting Asset Prices:** Unless the OTC system has a way of locking rates on the API side, it seems that a request could potentially be marked `PROCESSING`, and then be unable to pay for all its asset tokens (or have an excess) before ' `finalizeRequest` is called.
10. **Issues in Comments:**
    - `Controller.sol` line [56] and line [65]: technically, the "implementation" contract is the code address, not the proxy addess, and these are both the proxy addresses

- `Controller.sol` line [92] and line [103]: typo "funciton"
- `Factory.sol` line [172]: This comment is wrong. The function sets the address that assets from a burn should be moved to (as per the comment on line [110]).
- `Factory.sol` line [240]: typo "merchnat"
- Many comments say, "@notice Explain to an end user what this does" or, "@dev Explain to a developer any extra details". These should be replaced with the requested explanations.

11. **Issues in Error Messages:**

    - `OTC.sol` line [112]: There is only one validator address so this error message is a little confusing.
    - `OTC.sol` line [201]: This error message should probably be, "Mint request is not approved".

12. **Ignored Return Values:** It is best practice to check return values when they are provided by a function.

    - `DeployAMKT.constructor()` line [33] ignores the return value by `members.setCustodian()`.
    - `DeployAMKTPartTwo.completeDeployment()` line [35] ignores the return value by `controller.setMembers()`.
    - `DeployAMKTPartTwo.completeDeployment()` line [36] ignores the return value by `controller.setFactory()`.

13. **Minimise Function Access:** It is best practice to use the least public permission for a function.

    - `Controller.factoryPause()` should be declared external.
    - `DeployAMKTPartTwo.completeDeployment()` should be declared external.

14. **Zero Address Checks:** Consider adding checks to ensure submitted values are not `address(0)` in the following places:

    - `DeployAMKT.sol` and `DeployAMKTPartTwo.sol` A zero address in any of their function parameters could be a costly mistake in terms of gas.
    - `OTC.sol`: the constructor has no zero address checks.

15. **Contract Check:** The address parameters to `DeployAMKTPartTwo.completeDeployment()` could be checked to ensure that they are smart contracts.

16. **Interactions between Rebalancing and Inflation:** As the `AMKT` token will inflate between rebalances, the numbers in the methodology file will go out of date. Two methods were discussed for addressing this:

    - The systems will take account of the fee mint rate so that all functions dynamically modify the methodology when minting or burning. One possible issue here is that there could be a source of error if the fee rate changes between rebalances.
    - The idea was mentioned of inflation not being applied on mint and burn rates until rebalancing. If this was the case, it might lead to heavy burning just before rebalancing and then heavy minting right after.

17. **Possible Gas Optimisations:**

    - `DeployAMKT.sol` line [43] / line [72] a constant could be used for the supply ceiling. This would also allow for it to be updated in one place.
    - `DeployAMKT.sol` and `DeployAMKTPartTwo.sol` have multiple state variables that seem to be acting only as a form of logging. The events emitted by the contracts could be used instead for this purpose.
    - `Factory.burn()` line [323] is creating a memory variable `txid` that remain an empty `string`. This value literal could be used directly on line [329].
    - `OTC.finalizeRequest()` line [161] reads a variable from storage that is read again in storage on the next line. The memory variable `request` on line [163] could be used for this check. This will be more expensive if many transactions are submitted with the wrong `RequestStatus`, but this presumably will not be the case.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolutions

1. **Frontrunning Rebalance:** The development team acknowledged the issue.
2. **Meta Transaction Support:** Pending
3. **Ownership Transfer:** The development team have implemented a propose-accept system with the new contracts `ProposableOwnable` and `ProposableOwnableUpgradeable`.
4. **Token Name:** The token name has been changed to "Alongside Crypto Market Index"
5. **Issuer/Custodian Role Ambiguity:** The role name `Custodian` has been changed to `Issuer` in the Solidity code, thus significantly improving its readability.
6. **Supply Ceiling and Inflation:** The development team acknowledged the issue.
7. **Unused txid Parameter:** The `txid` parameter has been removed. ✓ Resolved in commit [ef0f764]
8. **Token Balance:** The recommended check has been added. ✓ Resolved in commit [5d00244]
9. **OTC and Shifting Asset Prices:** The development team acknowledged the observation.
10. **Issues in Comments:** The recommended changes have been made. ✓ Resolved in commit [68aaa98]
11. **Issues in Error Messages:** The recommended changes have been made. ✓ Resolved in commit [8dd8776]
12. **Ignored Return Values:** The recommended checks have been added. ✓ Resolved in commit [e51e84b]
13. **Minimise Function Access:** The recommended permissions have been implemented. ✓ Resolved in commit [f156eb4]
14. **Zero Address Checks:** The recommended checks have been added. ✓ Resolved in commit [c3770f8]
15. **Contract Check:** The development team acknowledged the suggestion and decided not to implment.
16. **Interactions between Rebalancing and Inflation:** The development team acknowledged the observation.
17. **Possible Gas Optimisations:** The development team acknowledged the first two suggestions and decided not to implment. The third suggestion was rendered void by the removal of `txid`. The fourth suggestion was implemented as recommended. ✓ Resolved in commit [7e26eb9]

AMKT-18	Miscellaneous General Comments - Second Review Stage
Asset	`contracts/*`
Status	**Closed:** See Resolution
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team during the second round of the review. In this section, all references to "OTC" are to `OTC.sol-2`:

1. **Unused State Variable:** `DeployAMKTPartTwo._partOneDeploymentAddress` is not set or read in the code.
2. **Event Key Mismatch:** The event `OTC.ProcessingBurnRequest` has a key `indexTokenAmount` but on line [342] its value is emitted as the variable `factoryNonce`.
3. **Payment Token Amounts in `OTC.finalizeRequest()` Events:** `OTC.finalizeRequest()` line [224] and line [242] are not representative of the amount of payment tokens processed in either situation.

   The event `ConfirmMintRequest` emits values that in no way take account of `toRefund`, and thus the actual payment tokens paid for the emitted number of index tokens.

   The event `RejectMintRequest` emits values that in no way take account of `request.deviation`, and thus the actual amount of payment tokens being returned.

   In either or both cases, the development team may wish to preserve the current situation if it has the desired functionality. There is no security issue with these values; they simply seemed possibly unintuitive.
4. **Zero Refund:** `OTC.openMintRefund()` does not check whether the request has any value to refund.
5. **Issues in Error Messages:**
   - `ProposableOwnable.sol` and `ProposableOwnableUpgradeable.sol` have revert messages that begin "`Ownable:`". This is a little misleading and might steer towards the parent contract. These messages could instead start with "`ProposableOwnable:`".
   - `Members.sol` line [57]: This error message should probably be, "invalid factory admin address".
   - `OTC.sol` line [127], line [305]: This error message says "less than", but the test allows equality.
   - `OTC.sol` line [371]: This error message says the request is already finalized, but this could be inaccurate as the test is only testing whether or not the status is equal to `RequestStatus.PROCESSING`.
6. **Ignored Return Values:** It is best practice to check return values when they are provided by a function. Note that some of these function calls are recommended to be replaced by their equivalents from `SafeERC20` in AMKT-15.
   - `DeployAMKT.constructor()` line [41] ignores the return value by `members.setFactoryAdmin()`.
   - `OTC.processRequest()` line [174] ignores the return value by `PaymentToken.transfer()`.
   - `OTC.claimIndexToken()` line [276] ignores the return value by `IndexToken.transfer()`.
   - `OTC.openMintRefund()` line [293] ignores the return value by `PaymentToken.transfer()`.
   - `OTC.requestBurn()` line [307] ignores the return value by `IndexToken.transferFrom()`.
   - `OTC.cancelBurn()` line [352] ignores the return value by `IndexToken.transfer()`.

- `OTC.finalizeBurn()` line [381] ignores the return value by `PaymentToken.transfer()`.
- `OTC.refundIndexToken()` line [436] ignores the return value by `IndexToken.transfer()`.
- `OTC.setFactoryDepositAddress()` line [452] ignores the return value by `factory.setMerchantDepositAddress()`.

7. **Unnecessary Function:** `OTC.setFactoryDepositAddress()` could be integrated into the constructor as it needs to be run once for the contract to be initialised.
8. **Variable Name Shadowing:** `OTC.constructor._owner` shadows `Ownable._owner`. These variables share the same name. Although the contract functions with this name collision, it would help avoid possible confusion or collisions to rename `OTC.constructor._owner`, perhaps to `OTC.constructor.owner_`.
9. **Comment Typo:** `Factory.sol` line [253], line [397]: typo "merchnat"

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

The affected contract is no longer in use for the majority of these points.

For **Comment Typo** and the first **Issues in Error Messages**:

✓ Resolved in commit [3d9fc3c]

AMKT-19	Impact of Key Compromise for Privileged Roles
Asset	`contracts/*`
Status	**Closed:** See Resolution
Rating	Informational

Description

The AMKT contracts require several privileged roles to facilitate the minting and burning of AMKT. The testing team recommends reassessing and documenting the abilities and trust assumptions of each of these roles to avoid confusion and bring clarity between specifications, implementation, and documentation.

For example:

- The role name `Custodian` was retained in code from legacy WBTC contracts, while referred as `Issuer` in documentation
- The `Issuer` in documentation seems to have the abilities of several roles in code: `Admin`, `Methodologist`, and `Custodian`
- In the `OTC` contract, it is unclear whether `Validator` is expected to be from the same `Validators` controlling `Issuer`, whether `User` in this contract is expected to be a `Merchant`, and the `OTC` contract must also be a `Merchant`

For the purposes of this review, the `Merchant` role is assumed to be malicious, and the implications of key compromise of other trusted roles is briefly considered here. These findings are not exhaustive and should be considered an overview of the topic:

1. **Issuer**

   The `Issuer` is intended to be set to a multi-sig wallet controlled by a set of validators who observe deposits and withdrawals of underlying assets to trigger minting and burning events.

   A compromise of the `Issuer` private key could allow an attacker to:

   - set a false `custodianDepositAddress`
   - collude with a `Merchant` to approve mint requests when assets have not been deposited
   - collude with a `Merchant` to approve a mint request for assets from another merchant (using `setCustodianDepositAddress()` to reassign the addresses to validate the malicious mint request and invalidate the real one, or simply reject the real request)
   - maliciously reject valid requests to mint or burn
   - set limits to allow excessive arbitrage or to maliciously block some merchants entirely

2. **OTC Validator**

   The OTC `Validator` can process, reject, or confirm a mint request through the OTC contract. The flow of approval of requests must be co-administered with the `Issuer`.

   If the OTC `Validator` is compromised, it could reject valid requests.

3. **Methodologist**

   The `Methodologist` can set the contents of the methodology file, which defines the underlying composition of 1 AMKT.

   A compromised `Methodologist` could grieve `Merchants` by front-running a burn request and setting AMKT value very low just before a burn.

   If a malicious `Merchant` compromises the `Methodologist` then the `Merchant` could:

   - redefine the nominal amount of 1 AMKT to equal most of the value in the vault, and attempt to claim those assets by burning 1 AMKT
   - set a very low AMKT value, mint AMKT at a discount, reset to original methodology, and submit a burn request

4. **Admin**

   `Admin` sets all of the other roles, therefore a compromise of these keys opens all of the previously noted attacks and more:

   - set `feeReceiver` to itself with a high inflation rate, add itself as a `Merchant`, and submit a burn request.
   - upgrade any of the upgradable contracts to behave maliciously. For example, it could add an unrestricted mint method to `IndexToken`.

Recommendations

Ensure that the comments are understood and acknowledged.

Resolution

The development team acknowledged the comments.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are given along with this document. The `brownie` framework was used to perform these tests and the output is given below.

```
test_config                      PASSED       [1%]
test_mint                        PASSED       [2%]
test_burn                        PASSED       [4%]
test_factoryPause                PASSED       [5%]
test_addMintRequest              PASSED       [6%]
test_cancelMintRequest           PASSED       [8%]
test_confirmMintRequest          PASSED       [9%]
test_rejectMintRequest           PASSED       [10%]
test_burn                        PASSED       [12%]
test_confirmBurnRequest          PASSED       [13%]
test_pause                       PASSED       [14%]
test_unpause                     PASSED       [16%]
test_setCustodianDepositAddress  PASSED       [17%]
test_setMerchantDepositAddress   PASSED       [18%]
test_setMerchantMintLimit        PASSED       [20%]
test_setMerchantBurnLimit        PASSED       [21%]
test_setCustodian                PASSED       [22%]
test_addMerchant                 PASSED       [24%]
test_removeMerchant              PASSED       [25%]
test_deployment                  PASSED       [27%]
test_requestIndexToken           PASSED       [28%]
test_processRequest              PASSED       [29%]
test_finalizeRequest_confirmed   PASSED       [31%]
test_claimIndexToken             PASSED       [32%]
test_addUser                     PASSED       [33%]
test_removeUser                  PASSED       [35%]
test_finalizeRequest_rejected    PASSED       [36%]
test_toggleRestriction           PASSED       [37%]
test_mint                        PASSED       [39%]
test_burn                        PASSED       [40%]
test_mintToFeeReceiver           PASSED       [41%]
test_pause                       PASSED       [43%]
test_unpause                     PASSED       [44%]
test_transfer                    PASSED       [45%]
test_transferFrom                PASSED       [47%]
test_setMethodologist            PASSED       [48%]
test_setMethodology              PASSED       [50%]
test_setFeeRate                  PASSED       [51%]
test_setFeeReceiver              PASSED       [52%]
test_setMinter                   PASSED       [54%]
test_setSupplyCeiling            PASSED       [55%]
test_config                      PASSED       [56%]
test_mint                        PASSED       [58%]
test_burn                        PASSED       [59%]
test_factoryPause                PASSED       [60%]
test_addMintRequest              PASSED       [62%]
test_cancelMintRequest           PASSED       [63%]
test_confirmMintRequest          PASSED       [64%]
test_rejectMintRequest           PASSED       [66%]
test_burn                        PASSED       [67%]
test_confirmBurnRequest          PASSED       [68%]
test_pause                       PASSED       [70%]
test_unpause                     PASSED       [71%]
test_setCustodianDepositAddress  PASSED       [72%]
test_setMerchantDepositAddress   PASSED       [74%]
test_setMerchantMintLimit        PASSED       [75%]
test_setMerchantBurnLimit        PASSED       [77%]
test_toggleRestriction           PASSED       [78%]
test_mint                        PASSED       [79%]
test_burn                        PASSED       [81%]
test_mintToFeeReceiver           PASSED       [82%]
```

```
test_pause                      PASSED       [83%]
test_unpause                    PASSED       [85%]
test_transfer                   PASSED       [86%]
test_transferFrom               PASSED       [87%]
test_setMethodologist           PASSED       [89%]
test_setMethodology             PASSED       [90%]
test_setFeeRate                 PASSED       [91%]
test_setFeeReceiver             PASSED       [93%]
test_setMinter                  PASSED       [94%]
test_setSupplyCeiling           PASSED       [95%]
test_duplicateMintRequest       XFAIL        [97%]
test_mintToFeeReceiver_rate     XFAIL        [98%]
test_requestIndexToken          XFAIL   (unsaf...)     [100%]
```

This is the test output from the second part of the review.

```
test_controller.py::test_config PASSED                                                     [  1%]
test_controller.py::test_mint PASSED                                                       [  2%]
test_controller.py::test_burn PASSED                                                       [  3%]
test_controller.py::test_factoryPause PASSED                                               [  4%]
test_factory.py::test_addMintRequest PASSED                                                [  5%]
test_factory.py::test_cancelMintRequest PASSED                                             [  6%]
test_factory.py::test_confirmMintRequest PASSED                                            [  7%]
test_factory.py::test_rejectMintRequest PASSED                                             [  8%]
test_factory.py::test_burn PASSED                                                          [  9%]
test_factory.py::test_confirmBurnRequest PASSED                                            [ 10%]
test_factory.py::test_pause PASSED                                                         [ 11%]
test_factory.py::test_unpause PASSED                                                       [ 12%]
test_factory.py::test_setIssuerDepositAddress PASSED                                       [ 13%]
test_factory.py::test_setMerchantDepositAddress PASSED                                     [ 15%]
test_factory.py::test_setMerchantMintLimit PASSED                                          [ 16%]
test_factory.py::test_setMerchantBurnLimit PASSED                                          [ 17%]
test_members.py::test_setIssuer PASSED                                                     [ 18%]
test_members.py::test_setFactoryAdmin PASSED                                               [ 19%]
test_members.py::test_addMerchant PASSED                                                   [ 20%]
test_members.py::test_removeMerchant PASSED                                                [ 21%]
test_otc.py::test_deployment PASSED                                                        [ 22%]
test_otc.py::test_requestIndexToken PASSED                                                 [ 23%]
test_otc.py::test_processRequest PASSED                                                    [ 24%]
test_otc.py::test_finalizeRequest_confirmed PASSED                                         [ 25%]
test_otc.py::test_claimIndexToken PASSED                                                   [ 26%]
test_otc.py::test_openMintRefund PASSED                                                    [ 27%]
test_otc.py::test_addUser PASSED                                                           [ 29%]
test_otc.py::test_removeUser PASSED                                                        [ 30%]
test_otc.py::test_requestBurn PASSED                                                       [ 31%]
test_otc.py::test_processBurn PASSED                                                       [ 32%]
test_otc.py::test_finalizeBurn_confirm PASSED                                              [ 33%]
test_otc.py::test_cancelBurn PASSED                                                        [ 34%]
test_otc.py::test_finalizeBurn_reject PASSED                                               [ 35%]
test_otc.py::test_refundIndexToken XFAIL (code bug on 435: requests/burnRequests)          [ 36%]
test_otc.py::test_setMaxDivergence PASSED                                                  [ 37%]
test_otc.py::test_getUser PASSED                                                           [ 38%]
test_otc.py::test_getUsers PASSED                                                          [ 39%]
test_otc.py::test_usersLength PASSED                                                       [ 40%]
test_otc.py::test_finalizeRequest_rejected PASSED                                          [ 41%]
test_otc_owner_lock.py::test_owner_lock_mint PASSED                                        [ 43%]
test_otc_owner_lock.py::test_owner_lock_burn XFAIL (code bug on 435: requests/burnRequests)    [ 44%]
test_otc_validator_theft.py::test_do_not_process PASSED                                    [ 45%]
test_otc_validator_to_refund.py::test_to_refund PASSED                                     [ 46%]
test_proposable_ownable.py::test_propose_owner PASSED                                      [ 47%]
test_proposable_ownable.py::test_transfer_ownership PASSED                                 [ 48%]
test_proposable_ownable_upgradeable.py::test_propose_owner PASSED                          [ 49%]
test_proposable_ownable_upgradeable.py::test_transfer_ownership PASSED                     [ 50%]
test_token.py::test_toggleRestriction PASSED                                               [ 51%]
test_token.py::test_mint PASSED                                                            [ 52%]
test_token.py::test_burn PASSED                                                            [ 53%]
test_token.py::test_mintToFeeReceiver PASSED                                               [ 54%]
test_token.py::test_pause PASSED                                                           [ 55%]
test_token.py::test_unpause PASSED                                                         [ 56%]
```

```
test_token.py::test_transfer PASSED                                         [ 58%]
test_token.py::test_transferFrom PASSED                                     [ 59%]
test_token.py::test_setMethodologist PASSED                                 [ 60%]
test_token.py::test_setMethodology PASSED                                   [ 61%]
test_token.py::test_setFeeRate PASSED                                       [ 62%]
test_token.py::test_setFeeReceiver PASSED                                   [ 63%]
test_token.py::test_setMinter PASSED                                        [ 64%]
test_token.py::test_setSupplyCeiling PASSED                                 [ 65%]
test_token_more.py::test_mintIgnoresShortLivedTokens PASSED                 [ 66%]
test_token_more.py::test_long_mint PASSED                                   [ 67%]
test_upgrade_controller.py::test_config PASSED                              [ 68%]
test_upgrade_controller.py::test_mint PASSED                                [ 69%]
test_upgrade_controller.py::test_burn PASSED                                [ 70%]
test_upgrade_controller.py::test_factoryPause PASSED                        [ 72%]
test_upgrade_factory.py::test_addMintRequest PASSED                         [ 73%]
test_upgrade_factory.py::test_cancelMintRequest PASSED                      [ 74%]
test_upgrade_factory.py::test_confirmMintRequest PASSED                     [ 75%]
test_upgrade_factory.py::test_rejectMintRequest PASSED                      [ 76%]
test_upgrade_factory.py::test_burn PASSED                                   [ 77%]
test_upgrade_factory.py::test_confirmBurnRequest PASSED                     [ 78%]
test_upgrade_factory.py::test_pause PASSED                                  [ 79%]
test_upgrade_factory.py::test_unpause PASSED                                [ 80%]
test_upgrade_factory.py::test_setIssuerDepositAddress PASSED                [ 81%]
test_upgrade_factory.py::test_setMerchantDepositAddress PASSED              [ 82%]
test_upgrade_factory.py::test_setMerchantMintLimit PASSED                   [ 83%]
test_upgrade_factory.py::test_setMerchantBurnLimit PASSED                   [ 84%]
test_upgrade_token.py::test_toggleRestriction PASSED                        [ 86%]
test_upgrade_token.py::test_mint PASSED                                     [ 87%]
test_upgrade_token.py::test_burn PASSED                                     [ 88%]
test_upgrade_token.py::test_mintToFeeReceiver PASSED                        [ 89%]
test_upgrade_token.py::test_pause PASSED                                    [ 90%]
test_upgrade_token.py::test_unpause PASSED                                  [ 91%]
test_upgrade_token.py::test_transfer PASSED                                 [ 92%]
test_upgrade_token.py::test_transferFrom PASSED                             [ 93%]
test_upgrade_token.py::test_setMethodologist PASSED                         [ 94%]
test_upgrade_token.py::test_setMethodology PASSED                           [ 95%]
test_upgrade_token.py::test_setFeeRate PASSED                               [ 96%]
test_upgrade_token.py::test_setFeeReceiver PASSED                           [ 97%]
test_upgrade_token.py::test_setMinter PASSED                                [ 98%]
test_upgrade_token.py::test_setSupplyCeiling PASSED                         [100%]
```

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

Impact \ Likelihood	Low	Medium	High
High	Medium	High	Critical
Medium	Low	Medium	High
Low	Low	Low	Medium

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] Sigma Prime. Solidity Security. Blog, 2018, Available: https://blog.sigmaprime.io/solidity-security.html. [Accessed 2018].

[2] NCC Group. DASP - Top 10. Website, 2018, Available: http://www.dasp.co/. [Accessed 2018].

σ'