Agora dollar: EVM rc

Security Review

Cantina Managed review by:
Optimum, Lead Security Researcher
0xRajeev, Lead Security Researcher
Kaden, Security Researcher
Víctor martínez, Associate Security Researcher
Delvir0, Junior Security Researcher

October 23, 2024

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	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
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. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Agora provides an instant, low cost gateway to the global economy for those who need it most, bringing the US dollar to individuals and businesses across the world.

From Jun 17th to Jun 25th the Cantina team conducted a review of agora-dollar-evm-rc on commit hash f3fbde2f. The team identified a total of **14** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 0
- Low Risk: 5
- Gas Optimizations: 4
- Informational: 5

3 Findings

3.1 Low Risk

3.1.1 Tokens can be minted to frozen addresses

Severity: Low Risk

Context: Erc20Privileged.sol#L35-L56

Description: Frozen account checks are performed on receiver addresses in `transfer()`, `transferFrom()`, `transferWithAuthorization()` and `receiveWithAuthorization()`. However, such a check is missing in `batchMint()` and `batchBurnFrom()`.

While skipping frozen address check in `batchBurnFrom()` is supposedly an intentional feature (as communicated) for claiming back any seized tokens from frozen accounts, skipping it in `batchMint()` may allow minters to accidentally mint tokens to frozen accounts.

Impact: Even though such minted tokens cannot be transferred and potentially burned to reclaim from frozen accounts, this may have regulatory implications for minters if those frozen accounts happen to belong to sanctioned entities.

Likelihood: Minters are protocol-trusted roles who can be reasonably expected to check for frozen accounts in the protocol context and therefore not mint tokens to them. However, given that freezer roles may be intentionally distinct and therefore independently operating from minter roles, there may be a scenario where minter accidentally mints to accounts that were frozen right after minter checked for the same (i.e. a classic TOCTOU race-condition), which is possible if the check and mint do not happen atomically in the same transaction.

Recommendation: Consider applying a frozen account check in `batchMint()`.

Agora: Acknowledged.

Spearbit: Acknowledged.

3.1.2 Tokens mistakenly sent to contract address may get stuck

Severity: Low Risk

Context: Erc20Core.sol#L33-L74, USDC, USDT

Description: ERC-20 tokens are sometimes mistakenly sent by their senders to token contract addresses instead of receiver addresses. Such tokens get locked up in the contract if it does not have a mitigation mechanism to address this scenario.

Related projects USDC and USDT have many such tokens (both their own as well as others) stuck in their contracts, which are worth more than 800K USD at the time of this review.

The `_transfer()` function does not enforce a `require(_to != address(this));` (as recommended here), which is presumably to avoid the extra check's gas cost on every transfer. USDC's design has a "rescuer" who can transfer stuck ERC20 tokens out of the contract, but its actual usage is not clear.

Impact: Agora and other ERC-20 tokens may get locked in the contract leading to loss of funds for the senders.

Likelihood: This user error may be expected to be handled at the wallet UI, for e.g. see how Metamask handles this scenario. Mistakenly sent Agora tokens be burned and reissued to senders by the authorized roles, but other ERC-20 tokens may remain stuck.

Recommendation: Consider if this UX risk is worth:

1. Introducing an authorized rescuer role with the trade-off being additional complexity and centralization.
2. Introducing a `require(_to != address(this));` check in `_transfer()` with the trade-off being additional gas cost introduced for every transfer.

Agora: Acknowledged.

Spearbit: Acknowledged.

3.1.3 `isReceiveWithAuthorizationUpgraded()` and `IS_SIGNATURE_VERIFICATION_PAUSED_BIT_POSITION()` functions are missing

Severity: Low Risk

Context: AgoraDollar.sol#L187-L189, AgoraDollar.sol#L231-L233

Description: In `AgoraDollar`, we have `view` and `pure` functions responsible for returning proxy state and bit mask positions, respectively. Of those, we're missing both `isReceiveWithAuthorizationUpgraded` and `IS_SIGNATURE_VERIFICATION_PAUSED_BIT_POSITION`, which should be included to maintain consistency with interfaces provided for similar logic.

While the functions are not intended to be called within the codebase itself, they may be intended to be called by external contracts at some point or otherwise just for users to be able to easily retrieve state.

Recommendation: Include these functions in `AgoraDollar.sol`

```
function isReceiveWithAuthorizationUpgraded() external view returns (bool) {
    return StorageLib.sloadImplementationSlotDataAsUint256().isReceiveWithAuthorizationUpgraded();
}
```

```
function IS_SIGNATURE_VERIFICATION_PAUSED_BIT_POSITION() external pure returns (uint256) {
    return StorageLib.IS_SIGNATURE_VERIFICATION_PAUSED_BIT_POSITION_;
}
```

Agora: Fixed in commit 58e54ab8.

Spearbit: Reviewed that commit 58e54ab8 adds the two missing functions as recommended.

3.1.4 Project may fail deployment on chains not compatible with Shanghai hardfork

Severity: Low Risk

Context: Global scope

Description: The Ethereum hardfork known as Shanghai introduced several new features to the chain, including the `PUSH0` opcode. By default, Solidity contracts with a version >= 0.8.20 are compiled for the `shanghai` EVM version thus incorporating this new opcode into the bytecode.

However, the `PUSH0` opcode is not fully supported by all EVM chains, such as Linea (see evm diff for an incomplete list of chains). The current codebase compiler version is `0.8.21` which renders it incompatible to be deployed on this set of chains.

Impact: The current setting may produce incompatible bytecode with some of the chains targetted by the protocol.

Recommendation: Consider either downgrading the Solidity version to <0.8.20 or setting the EVM version of the compiler to `paris`.

Agora: Acknowledged.

Spearbit: Acknowledged.

3.1.5 ERC-2612 Permit call can be frontrun to cause a temporary denial of service (DoS)

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: `Erc2612.sol` implements `permit` following the EIP-2612 specification to process signed approvals.

However, this call can be frontrun by an attacker leading to DoS for the subsequent call using the same `nonce`. This breaks the desired idempotent property of permit calls, where repeated calls with the same parameters should yield the same result without adverse effects.

Consider the following scenario where Agora Dollar is integrated on a protocol that uses signed deposits for vaults:

- The deposit action of the protocol (`depositWithPermit`) involves two steps:

1. Call Agora Dollar's `permit`.
2. Call `transferFrom` to transfer said allowed tokens into the vault.

- A user signs the permit object for a nonce `n` and includes it as a parameter for the `depositWithPermit` call which is then submitted to the mempool.
- A malicious party that is monitoring the mempool detects a valid permit signature, extracts it from the `depositWithPermit` user transaction , and frontruns it with a direct `permit` call with the valid call-data. This performs the allowance change and renders the nonce `n` of the permit signature invalid.
- The original `depositWithPermit` transaction is processed afterwards, but it reverts since the nonce `n` is not valid anymore. Thus `transferFrom` is not reached.

In few words when `permit` is a standalone (external) TX, not much can go wrong. However when consider a situation where `permit` is integrated in a contract call which depends on it being successful the end result is harmful, since the user loses the functionality that follows the `permit`.

Impact: The primary impact is griefing, where calls to `permit` can be DoS'd by an attacker to disrupt the signed approval flow for token integrations.

Likelihood: The likelihood of this issue occurring is moderate because it requires an attacker to actively monitor the mempool for `permit` transactions and front-run them for griefing. However, given the open and transparent nature of mempools, this attack vector is feasible for anyone.

Recommendation: Consider implementing a check to not revert the transaction when `allowance == expected` even if the nonce was already used. Furthermore, this added functionality should be clearly documented for integrators to make them aware that in this specific case the call will not revert.

Agora: Acknowledged.

Spearbit: Acknowledged.

3.1.6 Burning of tokens cannot be paused

Severity: Low Risk

Context: Erc20Privileged.sol#L67-L81

Description: Pausing the functionality of minting & transfer of tokens, freezing and signature verification is supported in the protocol. However, pausing is not supported for burning of tokens in `batchBurnFrom()`.

Impact: This will prevent a global pause enforcement of all protocol functionality (mint + burn + transfer + freeze + signature verification), which may be desirable in an emergency situation.

Likelihood: Burning of tokens is access controlled by `BURNER_ROLE`, which are protocol trusted addresses that can be expected to not burn during any emergency situation.

Recommendation: Consider adding a pause functionality for`batchBurnFrom()`.

Agora: Fixed in commit 6dba8e04.

Spearbit: Reviewed that commit 6dba8e04 adds support for pausing `batchBurnFrom()` functionality. We note that the bit positions in the "*Contract Access Control masks*" have been updated.

3.2 Gas Optimization

3.2.1 `_isMsgSenderFrozenCheckEnabled` check in `receiveWithAuthorization()` is redundant

Severity: Gas Optimization

Context: AgoraDollarErc1967Proxy.sol#L256-L261, Erc20Core.sol#L62-L63

Description: ERC-3009 `receiveWithAuthorization()` allows only the receiver to self-authorize a transfer to itself enforced with the `if (_to != msg.sender) revert InvalidPayee(msg.sender, _to);` check. Given that it thereafter calls `_transfer()` which unconditionally enforces a freeze check on `_to`, the `_isMsgSenderFrozenCheckEnabled` check enforced on `msg.sender` in `receiveWithAuthorization()` is redundant because `msg.sender == _to`.

Recommendation: Consider removing the redundant `_isMsgSenderFrozenCheckEnabled` check enforced on `msg.sender` in `receiveWithAuthorization()` to save gas.

Agora: Fixed in commit 7f2d7b81.

Spearbit: Reviewed that commit 7f2d7b81 removes the redundant `_isMsgSenderFrozenCheckEnabled` check enforced on `msg.sender` in `receiveWithAuthorization()`.

We note that commit 7f2d7b81 also refactors enforcement of `isTransferPaused()` and `isSignatureVerificationPaused()` by hoisting them from `_transfer()`, `_transferWithAuthorization()` and `_receiveWithAuthorization()` to their call sites in `AgoraDollarErc1967Proxy.sol`.

3.2.2 `totalSupply` may be decremented inside an `unchecked` block

Severity: Gas Optimization

Context: Erc20Privileged.sol#L74

Description: During the execution of `batchBurnFrom`, the `totalSupply` decrement will never underflow because, in ERC20 tokens, the following invariant always holds: *Any user balance is never larger than the total supply*.

Therefore, the following decrement can be executed inside an unchecked block:

```
StorageLib.getPointerToErc20CoreStorage().totalSupply -= _value248;
```

This is because the decrement on the user balance is enough to check for underflow:

```
StorageLib.getPointerToErc20CoreStorage().accountData[_burns[i].burnFromAddress].balance -= _value248;
```

Recommendation: Change the order of operations for `totalSupply` and user balance decrements and add an unchecked block for the `totalSupply` decrement.

```
  for (uint256 i = 0; i < _burns.length; i++) {

      // Effects: subtract from totalSupply and account balance

      uint248 _value248 = _burns[i].value.toUint248();

-     StorageLib.getPointerToErc20CoreStorage().totalSupply -= _value248;

      StorageLib.getPointerToErc20CoreStorage().accountData[_burns[i].burnFromAddress].balance -= _value248;
+     unchecked {
+         StorageLib.getPointerToErc20CoreStorage().totalSupply -= _value248;
+     }
```

Agora: Acknowledged.

Spearbit: Acknowledged.

3.2.3 Caching of EIP-712 immutables is redundant

Severity: Gas Optimization

Context: Eip712.sol#L81

Description: In `Eip712`, during deployment we cache some immutables: `_cachedChainId`, `_cachedDomainSeparator`, and `_cachedThis`. Under most circumstances, this would be useful because we can avoid computing the domain separator every time we reference it, instead referencing a cached value as long as the contract address or `chainid` hasn't changed since deployment:

```
function _domainSeparatorV4() internal view returns (bytes32) {
    if (address(this) == _cachedThis && block.chainid == _cachedChainId) return _cachedDomainSeparator;
    else return _buildDomainSeparator();
}
```

However, since this contract is interacted with via a proxy, `_cachedThis` will reference the implementation contract address while `address(this)` references the proxy contract address. As a result, we must always rebuild the domain separator, making the cached immutables redundant.

Recommendation: Remove the cached immutables and run `_buildDomainSeparator` each time `_domainSeparatorV4` is called:

```
- bytes32 private immutable _cachedDomainSeparator;
- uint256 private immutable _cachedChainId;
- address private immutable _cachedThis;
  // ...

  constructor(string memory name, string memory version) {
      _name = name.toShortString();
      _version = version.toShortString();
      _hashedName = keccak256(bytes(name));
      _hashedVersion = keccak256(bytes(version));

-     _cachedChainId = block.chainid;
-     _cachedDomainSeparator = _buildDomainSeparator();
-     _cachedThis = address(this);
  }

  // ...

  function _domainSeparatorV4() internal view returns (bytes32) {
-     if (address(this) == _cachedThis && block.chainid == _cachedChainId) return _cachedDomainSeparator;
      else return _buildDomainSeparator();
  }
```

Alternatively, it may be more gas efficient to incorporate an `initialize` function which can be called by the proxy to cache the same values in storage on the proxy contract rather than as immutables. Note that this approach would require carefully applying storage using ERC7201, as well as carefully implementing the `initialize` function such that it can't be re-initialized and that the implementation itself cannot be initialized. The OpenZeppelin EIP712Upgradeable implementation could be a valuable reference.

Agora: Fixed in commit 831c0a27.

Spearbit: Reviewed that commit 831c0a27 resolves this finding by caching a domain separator with the intended proxy address, which prevents the need to rebuild the domain separator each time.

3.2.4 Constants can be precomputed

Severity: Gas Optimization

Context: Erc2612.sol#L25-L27, StorageLib.sol#L219-L232

Description: In `Erc2612`, we have a constant `PERMIT_TYPEHASH` which contains the EIP-712 typehash for the `permit` signature:

```
/// @notice The ```PERMIT_TYPEHASH``` stores keccak256("Permit(address owner,address spender,uint256
↪  value,uint256 nonce,uint256 deadline)")
bytes32 public constant PERMIT_TYPEHASH =
    keccak256("Permit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)");
```

This is computed during deployment, but since it's a constant value, we can precompute the typehash off-chain and provide the hash directly to save gas during deployment. Additionally, in `StorageLib`, we compute bit positions for various access control masks:

```
// Contract Access Control masks
uint256 internal constant IS_MSG_SENDER_FROZEN_CHECK_ENABLED_BIT_POSITION_ = 1 << (255 - 0);
uint256 internal constant IS_MINT_PAUSED_BIT_POSITION_ = 1 << (255 - 1);
uint256 internal constant IS_FREEZING_PAUSED_BIT_POSITION_ = 1 << (255 - 2);
uint256 internal constant IS_TRANSFER_PAUSED_BIT_POSITION_ = 1 << (255 - 3);
uint256 internal constant IS_SIGNATURE_VERIFICATION_PAUSED_BIT_POSITION_ = 1 << (255 - 4);

// internal function upgrade masks
// Erc20
uint256 internal constant IS_TRANSFER_UPGRADED_BIT_POSITION_ = 1 << (255 - 10);
uint256 internal constant IS_TRANSFER_FROM_UPGRADED_BIT_POSITION_ = 1 << (255 - 11);

// Eip 3009
uint256 internal constant IS_TRANSFER_WITH_AUTHORIZATION_UPGRADED_BIT_POSITION_ = 1 << (255 - 12);
uint256 internal constant IS_RECEIVE_WITH_AUTHORIZATION_UPGRADED_BIT_POSITION_ = 1 << (255 - 13);
```

These values can also be precomputed off-chain, assigning the result directly to the constant to save gas on deployment.

Recommendation: Precompute the `PERMIT_TYPEHASH` and provide it directly to the constant:

```
  bytes32 public constant PERMIT_TYPEHASH =
-     keccak256("Permit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)");
+     0x6e71edae12b1b97f4d1f60370fef10105fa2faae0126114a169c64845d6126c9;
```

Additionally, precompute the bit positions of the `StorageLib` access control masks, e.g.:

```
- uint256 internal constant IS_MSG_SENDER_FROZEN_CHECK_ENABLED_BIT_POSITION_ = 1 << (255 - 0);
+ uint256 internal constant IS_MSG_SENDER_FROZEN_CHECK_ENABLED_BIT_POSITION_ =
↪  0x8000000000000000000000000000000000000000000000000000000000000000; // 1 << (255 - 0)
```

Agora: Acknowledged.

Spearbit: Acknowledged.

3.3 Informational

3.3.1 `setIsTransferUpgraded` and similar purpose functions should be called as part of the `_upgradeToAndCall` flow upon upgrading

Severity: Informational

Context: AgoraDollarErc1967Proxy.sol#L89

Description: As opposed to the well known proxy pattern where the underlying logic resides only in the implementation contracts, in this case the project team decided to implement some of the functions inside the proxy itself for gas efficiency. To make sure these functions can be also upgraded functions like `setIsTransferUpgraded` where introduced. The purpose of these functions is to flag whether an upgrade to the `transfer` function has occurred and these are later used to target the call to the correct and most updated implementation.

Assuming one of the proxy functions need to be fixed the sequence of calls is supposed to be:

1. Pause the flawed proxy function (`AgoraDollarErc1967Proxy.transfer` in our example).
2. Deploy the fixed implementation contract.
3. Call `_upgradeToAndCall`.
4. Call `setIsTransferUpgraded`.
5. Unpause the `transfer` function.

However, during the process the dev team should make sure that `_upgradeToAndCall` includes a call to `setIsTransferUpgraded` so that both operations will happen atomically and there will be no chance for a user transfer to accidentally use the old implementation just because the `setIsTransferUpgraded` was wrongly called after the unpausing of the `transfer` function.

It is important to note that this issue is relevant for `setIsTransferUpgraded`, `setIsTransferFromUpgraded`, `setIsTransferWithAuthorizationUpgraded`, `setIsReceiveWithAuthorizationUpgraded`.

Agora: Acknowledged.

Spearbit: Acknowledged.

3.3.2 ERC-3009 `transferWithAuthorization()` can be front-run to cause unexpected behavior

Severity: Informational

Context: AgoraDollarErc1967Proxy.sol#L154, ERC-3009

Description: ERC-3009 `transferWithAuthorization()` has a front-running risk if called from smart contracts as noted in its security consideration:

> It is possible for an attacker watching the transaction pool to extract the transfer authorization and front-run the transferWithAuthorization call to execute the transfer without invoking the wrapper function. This could potentially result in unprocessed, locked up deposits. receiveWithAuthorization prevents this by performing an additional check that ensures that the caller is the payee.

Recommendation: Consider documenting the above risk for users and integrators.

Agora: Acknowledged.

Spearbit: Acknowledged.

3.3.3 Consistency and readability of code and comments could be improved

Severity: Informational

Context: Global scope

Description: While a majority of the code and comments adhere to a consistent convention and enhanced readability, there are a few places where it could be improved. Some such examples are highlighted below:

1. Missing NatSpec for functions in `AgoraPrivilegedRole` and few functions in other contracts.
2. The convention of specifying events as, for e.g., `emit Transfer({ from: _from, to: _to, value: _transferValue })`, and reverts as, for e.g., `revert AccountIsFrozen({ frozenAccount: _to })` is missing in a few places such as Erc20Privileged.sol#L69, OwnableAccessControl.sol#L99, OwnableAccessControl.sol#L116, Eip3009.sol#L115, Erc20Core.sol#81 and Erc20Privileged.sol#L45.
3. The `@notice` for `AgoraOwnable2Step` says it is an `abstract contract` but it is declared as `contract`.
4. `error AddressIsNotOwner` has `@notice Emitted when owner is transferred`, which is inaccurate because this error may be thrown even during `setMinterThrottleInfo()` or granting/revoking roles.
5. `OwnableAccessControl._requireIsRole()` can be moved down to the section with other internal functions.
6. `/// @notice The ```_requirePendingOwner``` function` in the NatSpec for `AgoraOwnable2Step._requireSenderIsPendingOwner()` should say `_requireSenderIsPendingOwner` instead of `_requirePendingOwner`.
7. `/// @return Whether or not msg.sender is current owner address` in the NatSpec for `AgoraOwnable2Step._isOwner()` should say `_address` instead of `msg.sender`.
8. `AgoraDollarAccessControl.transferRole()` uses the conditional check `if (!(_isRole({ _role: _role, _address: msg.sender }) || _isRole({ _role: ADMIN_ROLE, _address: msg.sender })))`, while `_requireSenderIsRole()` is an abstraction created specifically for such `msg.sender` role checks as used in `AgoraDollarCore`. Consider using `_requireSenderIsRole()` in the conditional for consistency.

Recommendation: Consider improving the consistency and readability of code and comments for aspects highlighted above.

Agora: Fixed in commit 0a8e13ce

Spearbit: Reviewed that commit 0a8e13ce partially fixes some of the issues highlighted in (2).

3.3.4 Implementation contract can be initialized by anyone

Severity: Informational

Context: AgoraDollarCore.sol#L43

Description: `AgoraDollarCore` contains an `initialize` function which gets executed via the proxy during deployment, setting the initial access control admin. Since `initialize` is called only via the proxy contract, we do not block initialize from being called in the context of the implementation contract. As a result, it's possible for anyone to call `initialize` to set an arbitrary address as the access control admin of the implementation contract.

This does not seem to have any effects on the proxy contract since the initialization doesn't set any immutables and the implementation contract cannot execute arbitrary calls or unsafe operations like `selfdestruct`. However, this is not intended behavior and may still have some unforeseen consequences and thus should be prevented.

Recommendation: In the `AgoraDollarCore` constructor, call `_disableInitializers` to prevent the implementation from being initialized.

```
  constructor(ConstructorParams memory _params) Eip712(_params.eip712Name, _params.eip712Version) {
      _name = _params.name.toShortString();
      _symbol = _params.symbol.toShortString();
+     _disableInitializers();
  }
```

Agora: Fixed in commit b4209044

Spearbit: Reviewed that commit b4209044 adds `_disableInitializers();` to the constructor as recommended.

3.3.5 Centralization risks should be appropriately documented and managed

Severity: Informational

Context: Global scope

Description: The protocol, by design, has several aspects of centralization such as the ability to batch mint/burn tokens, freeze addresses, pause contracts and upgrade functionality as managed by role-based access control. Such centralization aspects could potentially be compromised or their failure modes accidentally triggered.

Recommendation: Consider documenting/highlighting centralization aspects and managing operational risks by following best-practices of key storage and usage. This should be complemented with active monitoring and incident response preparedness.

Agora: Acknowledged.

Spearbit: Acknowledged.