// Security Assessment

12.19.2025 - 12.22.2025

Dollar EVM

Agora Finance

HALBORN

Dollar EVM - Agora Finance

Prepared by: HALBORN

Last Updated 01/16/2026

Date of Engagement: December 19th, 2025 - December 22nd, 2025

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
5	0	0	0	4	1

TABLE OF CONTENTS

1. INTRODUCTION

`Agora Finance` engaged Halborn to conduct a security assessment on their smart contracts beginning on December 19th, 2025 and ending on December 22nd, 2025. The security assessment was scoped to the smart contracts provided in the `agora-dollar-evm-dev` Github repository, provided to the Halborn team. Commit hash and further details can be found in the Scope section of this report.

The review is for the diff introduced in the Agora token upgrade, which adds a mint rate-limiting mechanism based on linear decay and integrates it into the existing minting flow. The updated design enforces per-minter throttling while preserving the protocol's role-based access control and upgrade mechanism.

2. ASSESSMENT SUMMARY

`Halborn` was provided with 2 days for this engagement and assigned a full-time security engineer to assess the security of the smart contracts in scope. The assigned engineer possess deep expertise in blockchain and smart contract security, including hands-on experience with multiple blockchain protocols.

The objective of this assessment is to:

- Identify potential security issues within the `Agora Dollar` protocol smart contracts.
- Ensure that smart contract of `Agora Dollar` protocol functions operate as intended.

In summary, `Halborn` identified several areas for improvement to reduce the likelihood and impact of security risks, which were mostly acknowledged by the `Agora Finance` team. The main recommendations were:

- `Adjusting the decay formula to minimize truncation effects.`
- `Restricting initialization to an authorized caller.`
- `Resetting minter's rate limit state when the minter role is revoked.`

3. CAVEATS

This assessment only covers the changes introduced between commit `89927a7` and commit `4a47d91`. All other parts of the codebase are treated as black boxes, and it is assumed that any required security mechanisms are properly implemented elsewhere in the system. This includes, but is not limited to, input validation, business logic enforcement, and access control.

Differential audits focus on identifying security risks introduced or modified in a specific set of changes. While some components added in the diff may interact with each other or with existing logic, these interactions are only reviewed when they are explicitly visible within the diff itself. Implicit assumptions or cross-component dependencies may fall outside the scope unless clearly surfaced through changes or supporting context.

For a more complete understanding of the protocol's security posture and the behavior of interconnected components, a full-scope assessment is recommended.

4. TEST APPROACH AND METHODOLOGY

`Halborn` performed a combination of manual review of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices.

The following phases and associated tools were used throughout the term of the assessment:

- Research into the architecture and purpose of the `Agora Dollar` protocol.
- Manual code review and walkthrough of the `Agora Dollar` in-scope contracts.
- Manual assessment of critical `Solidity` variables and functions to identify potential vulnerability classes.
- Manual testing using custom scripts.
- Static Analysis of security for scoped contract, and imported functions. (`Slither`).
- Local deployment and testing with (`Foundry`, `Remix IDE`).

5. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

5.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

5.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (C:N) Low (C:L) Medium (C:M) High (C:H) Critical (C:C)	0 0.25 0.5 0.75 1
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

5.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4

SEVERITY	SCORE VALUE RANGE
Informational	0 - 1.9

6. SCOPE

REPOSITORY

(a) Repository: agora-dollar-evm-dev

(b) Assessed Commit ID: 4a47d91

(c) Items in scope:

- src/contracts/AgoraDollar.sol
- src/contracts/AgoraDollarAccessControl.sol
- src/contracts/AgoraDollarCore.sol
- src/contracts/AgoraDollarMintRateLimit.sol
- src/contracts/Erc20Privileged.sol
- src/contracts/interfaces/IAgoraDollar.sol
- src/contracts/proxy/StorageLib.sol

Out-of-Scope: Third party dependencies and economic attacks. Furthermore, this assessment was limited to the differential changes between commit 89927a7 and commit 4a47d91. Components outside of the reviewed diff, pre-existing logic not modified by the changes, system-wide interactions not directly surfaced in the diff context, and external assumptions regarding input validation, business rules, and access controls were explicitly out of scope.

REMEDIATION COMMIT ID:

- fd017c1

Out-of-Scope: New features/implementations after the remediation commit IDs.

7. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	4	1

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
PRECISION LOSS IN LINEAR DECAY AFFECTS RATE-LIMIT GRANULARITY	LOW	RISK ACCEPTED - 01/14/2026
UNPROTECTED INITIALIZATION ENABLES ROLE ASSIGNMENT BY FIRST CALLER	LOW	RISK ACCEPTED - 01/14/2026
RATE-LIMIT STATE PERSISTS ACROSS MINTER ROLE REVOCATION AND RE-GRANT	LOW	RISK ACCEPTED - 01/14/2026
REMOVED MINTERS RETAIN MINTING ALLOWANCE IN RATE-LIMIT VIEW FUNCTION	LOW	SOLVED - 01/07/2026
RATE LIMITS CAN BE CONFIGURED FOR ADDRESSES WITHOUT MINTING AUTHORIZATION	INFORMATIONAL	ACKNOWLEDGED - 01/14/2026

8. FINDINGS & TECH DETAILS

8.1 PRECISION LOSS IN LINEAR DECAY AFFECTS RATE-LIMIT GRANULARITY

// LOW

Description

`AgoraDollarMintRateLimit._amountCanBeMinted()` relies on integer division when computing linear decay. Due to rounding down, small elapsed times or low limits result in zero decay, leaving `amountInFlight` unchanged even though time has passed. This allows minting capacity to replenish in coarse steps rather than smoothly, weakening the intended throttling behavior.

Copy Code

```
63  function _amountCanBeMinted(
64        uint256 _amountInFlight,
65        uint256 _lastUpdated,
66        uint256 _limit,
67        uint256 _window
68    ) internal view returns (uint256 currentAmountInFlight, uint256 amountCanBeMinted) {
69        //slither-disable-next-line timestamp
70        uint256 _timeSinceLastMint = block.timestamp - _lastUpdated;
71        // @dev Presumes linear decay.
72        uint256 _decay = (_limit * _timeSinceLastMint) / (_window > 0 ? _window : 1); // prevent di
73        currentAmountInFlight = _amountInFlight <= _decay ? 0 : _amountInFlight - _decay;
74        // @dev In the event the _limit is lowered, and the 'in-flight' amount is higher than the _
75        amountCanBeMinted = _limit <= currentAmountInFlight ? 0 : _limit - currentAmountInFlight;
76  }
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:L/I:L/D:N/Y:N (3.1)

Recommendation

Adjust the decay formula to minimize truncation effects (higher precision math).

Remediation Comment

RISK ACCEPTED: The **Agora Finance team** accepted the risk of this finding.

8.2 UNPROTECTED INITIALIZATION ENABLES ROLE ASSIGNMENT BY FIRST CALLER

// LOW

Description

The `AgoraDollarCore.initialize()` function is declared as `external reinitializer(4)` and does not enforce any access control. If initializer version-4 has not yet been consumed on a proxy, the first caller can invoke `initialize()` and arbitrarily set all privileged roles, including admin, minter, pauser, freezer, and rate-limit manager.

```
82 | function initialize(InitializeParams memory _params) external reinitializer(4) {...}
```

On chains where the stored initializer version is less than 3, the public `initialize()` function may be invoked by any address to pump the version and assign privileged roles. This exposure can be addressed by enforcing access control on the function. For deployments that have already been initialized up to version 3, the risk can be mitigated by performing the version-4 initialization atomically with the upgrade in a single transaction.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (2.5)

Recommendation

Initialization must be restricted to an authorized caller.

Remediation Comment

RISK ACCEPTED: The **Agora Finance team** accepted the risk of this finding.

8.3 RATE-LIMIT STATE PERSISTS ACROSS MINTER ROLE REVOCATION AND RE-GRANT

// LOW

Description

Upon revocation of the minter role, the associated rate-limit state fields (`amountInFlight`, `lastUpdated`, `limit`, `window`) are not reset. If the same address is later re-granted the minter role, minting is resumed using the previously stored rate-limit state. As a result, residual minting capacity can be retained by a previously removed minter after re-addition, allowing minting capability to be regained sooner than intended.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (2.5)

Recommendation

Rate-limit state for the minter must be coupled to the role lifecycle by resetting the minter's rate-limit state when the minter role is revoked.

Remediation Comment

RISK ACCEPTED: The **Agora Finance team** accepted the risk of this finding.

8.4 REMOVED MINTERS RETAIN MINTING ALLOWANCE IN RATE-LIMIT VIEW FUNCTION

// LOW

Description

When a minter role is revoked, the associated rate-limit state (`amountInFlight`, `limit`, `window`) remains unchanged. As a result, `AgoraDollarMintRateLimit.getAmountCanBeMinted()` may continue to report non-zero minting capacity for addresses that are no longer authorized minters. While actual minting is blocked by access control, this creates state inconsistency, relying on the rate-limit manager to manually reset the configuration.

Copy Code

```
102 function getAmountCanBeMinted(
103     address _minter
104 ) external view returns (uint256 currentAmountInFlight, uint256 amountCanBeMinted) {
105     StorageLib.RateLimit memory rateLimit = StorageLib.getPointerToMintRateLimitStorage().rateLim
106     return
107         _amountCanBeMinted({
108             _amountInFlight: rateLimit.amountInFlight,
109             _lastUpdated: rateLimit.lastUpdated,
110             _limit: rateLimit.limit,
111             _window: rateLimit.window
112         });
113 }
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (2.5)

Recommendation

It is recommended that the `AgoraDollarMintRateLimit.getAmountCanBeMinted()` function be modified to return zero for minters whose authorization has been revoked.

Remediation Comment

SOLVED: The **Agora Finance team** solved the issue in the specified commit id by following the mentioned recommendation.

Remediation Hash

https://github.com/amphora-atlas/agora-dollar-evm-dev/commit/fd017c1a215b18e4ac22463cb24b659d2233c460

8.5 RATE LIMITS CAN BE CONFIGURED FOR ADDRESSES WITHOUT MINTING AUTHORIZATION

// INFORMATIONAL

Description

The `AgoraDollarMintRateLimit.setMintRateLimit()` function allows configuring a rate limit for any arbitrary address without verifying that the address currently holds a minting role (`MINTER_ROLE` or `BRIDGE_MINTER_ROLE`). As a result, the rate-limit window and decay state may start accumulating before the address is actually authorized to mint.

Copy Code

```
42  function _setMintRateLimit(MintRateLimitConfig memory _rateLimitConfig) internal {
43        StorageLib.RateLimit storage rateLimit = StorageLib.getPointerToMintRateLimitStorage().rate
44            _rateLimitConfig.minter
45        ];
46
47        // @dev Ensure we checkpoint the existing rate limit as to not retroactively apply the new
48        _outflow({ _minter: _rateLimitConfig.minter, _amount: 0 });
49
50        // @dev Does NOT reset the amountInFlight/lastUpdated of an existing rate limit.
51        rateLimit.limit = _rateLimitConfig.limit;
52        rateLimit.window = _rateLimitConfig.window;
53        emit MintRateLimitChanged(_rateLimitConfig);
54    }
```

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.5)

Recommendation

Validation should be performed to ensure the target address holds the required minting role before the rate limit is configured. Alternatively, the minter's rate-limit state should be initialized at the moment the minting role is granted to guarantee consistent behavior.

Remediation Comment

ACKNOWLEDGED: The **Agora Finance team** accepted the risk of this finding.
