CERTIK

AVA Foundation Audit

Security Assessment

CertiK Assessed on Nov 26th, 2025

CertiK Assessed on Nov 26th, 2025

AVA Foundation Audit

The security assessment was prepared by CertiK.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Staking	Ethereum (ETH)	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE
Solidity	Preliminary comments published on 11/17/2025 Final report published on 11/26/2025

Vulnerability Summary

13	8	3	1	1	0
Total Findings	Resolved	Multi-Sig	Partially Resolved	Acknowledged	Declined

3	Centralization	3 Multi-Sig	Centralization findings highlight privileged roles & functions and their capabilities, or instances where the project takes custody of users' assets.
0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
1	Major	1 Resolved	Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control.
2	Medium	1 Partially Resolved, 1 Acknowledged	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
2	Minor	2 Resolved	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
5	Informational	5 Resolved	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | AVA FOUNDATION AUDIT

CODEBASE | AVA FOUNDATION AUDIT

Repository

https://github.com/AVA-Foundation/ava-lockup-contracts

Commit

d807a5fbfab620e529e23c70bc60948f2f311dae

a1956a3eee93b6246fd4aaceaefc8c8db6863b9f

a25e0028d5ce293abc8c20e72add11de8d2ba0ba

AUDIT SCOPE | AVA FOUNDATION AUDIT

AVA-Foundation/ava-lockup-contracts
DailyEarnLockUp.sol
OwnPauseAuth.sol

APPROACH & METHODS | AVA FOUNDATION AUDIT

This audit was conducted for AVA Foundation to evaluate the security and correctness of the smart contracts associated with the AVA Foundation Audit project. The assessment included a comprehensive review of the in-scope smart contracts. The audit was performed using a combination of Formal Verification, Manual Review, and Static Analysis.

The review process emphasized the following areas:

- Architecture review and threat modeling to understand systemic risks and identify design-level flaws.
- Identification of vulnerabilities through both common and edge-case attack vectors.
- Manual verification of contract logic to ensure alignment with intended design and business requirements.
- Dynamic testing to validate runtime behavior and assess execution risks.
- Assessment of code quality and maintainability, including adherence to current best practices and industry standards.

The audit resulted in findings categorized across multiple severity levels, from informational to critical. To enhance the project's security and long-term robustness, we recommend addressing the identified issues and considering the following general improvements:

- Improve code readability and maintainability by adopting a clean architectural pattern and modular design.
- Strengthen testing coverage, including unit and integration tests for key functionalities and edge cases.
- Maintain meaningful inline comments and documentations.
- Implement clear and transparent documentation for privileged roles and sensitive protocol operations.
- Regularly review and simulate contract behavior against newly emerging attack vectors.

FINDINGS | AVA FOUNDATION AUDIT

13	0	3	1	2	2	5
Total Findings	Critical	Centralization	Major	Medium	Minor	Informational

This report has been prepared for AVA Foundation to identify potential vulnerabilities and security issues within the reviewed codebase. During the course of the audit, a total of 13 issues were identified. Leveraging a combination of Formal Verification, Manual Review & Static Analysis the following findings were uncovered:

ID	Title	Category	Severity	Status
AFA-02	**Centralization Related Risks**	**Centralization**	**Centralization**	**2/3 Multi-Sig**
AFA-03	**Centralized Control Of Token Withdrawal**	**Centralization**	**Centralization**	**2/3 Multi-Sig**
AFA-13	**Potential Risk Of User Fund Theft**	**Design Issue, Centralization**	**Centralization**	**2/3 Multi-Sig**
AFA-04	The `maxLockupAmountPerTotal` Can Be By-Passed When Updating The Users' Membership	Logical Issue	Major	Resolved
AFA-05	Missing Check If The User Id And Wallet Address Are Used	Logical Issue	Medium	Partially Resolved
AFA-06	Updating An Exist Membership Type's Lockup Condition Affects Users' Rewards	Volatile Code	Medium	Acknowledged
AFA-07	Missing Zero Address Check	Volatile Code	Minor	Resolved
AFA-08	Missing Unknown Membership Type Validation In `updateUserIdToMembershipType()` Function	Coding Issue	Minor	Resolved
AFA-01	Design Logic Of The Reward Resource	Design Issue	Informational	Resolved

ID	Title	Category	Severity	Status
AFA-09	The Comparison Operator Prefer To Use '>=' Instead Of '>'	Logical Issue	Informational	Resolved
AFA-10	The `amountNFTs` Only For The `SmartDiamond` Membership Type.	Volatile Code	Informational	Resolved
AFA-11	The Potential Fee-On-Transfer Token	Volatile Code	Informational	Resolved
AFA-12	Unused Function	Code Optimization	Informational	Resolved

AFA-02 | Centralization Related Risks

Category	Severity	Location	Status
Centralization	**Centralization**		**2/3 Multi-Sig**

Description

In the contract `OwnPauseAuth`, the role `owner` has authority over the following functions:

- `grantAuthorized()`
- `revokeAuthorized()`
- `pause()`
- `unpause()`

Any compromise to the `owner` account may allow an attacker to arbitrarily assign or strip privileged operators and globally pause or resume all inheriting protocol functionality, enabling censorship or shutdown.

In the contract `DailyEarnLockUp`, the role `owner` has authority over the following functions:

- `withdrawByAdmin()`
- `takeImmediateWithdrawalFeeCollected()`
- `setMembershipTypeToCondition()`
- `setImmediateWithdrawalFee()`
- `setCommunityWallet()`
- `setMaxTotalLockedAmount()`
- `updateCommonMinLockupAmount()`
- `updateCommonWithdrawPeriod()`

Any compromise to the `owner` account may allow an attacker to drain staked tokens, seize fee revenue, and arbitrarily rewrite core economic parameters (lockup conditions, fees, withdrawal caps, treasury recipient), letting them expropriate user funds or reconfigure the program at will.

In the contract `DailyEarnLockUp`, the role `authorized operator` has authority over the following functions:

- `deactivateUser()`
- `activateUser()`
- `initUserData()`
- `updateUserIdToWalletAddress()`
- `updateUserIdToAmountNFTs()`
- `updateUserIdToMembershipType()`

Any compromise to an `authorized operator` account may allow an attacker to block or restore users, rewrite wallet mappings, and reshuffle membership tiers and NFT-based bonuses, enabling targeted censorship, diversion of rewards, or forced forfeiture of withdrawal rights.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[AVA Foundation, 11/26/2025]: The team acknowledged the issue and adopted the multisign solution to ensure the private key management process at the current stage. The `DailyEarnLockUp` contract has transferred the ownership to a Gnosis Safe contract with 2/3 signers in the sensitive function signing process.

- Grant Role transaction hash for Gnosis Safe: 0x34595881f1d17f33e87c720c632cbd32fe021b0c15ab9f1ad5a9d54e9d0f56dd, Gnosis safe contract address: eth:0x58653987Ff3837ADBE6383F670f6935fcDE521b0
- The 3 multisign addresses:
    1. EOA:0xA5CbE8c764323f78c023F9342Dc867b10fb57C3f]
    2. EOA:0x9ea99109E1b1Aa7e83C028391FB2D038fa6a4174
    3. EOA:0x73524D7f64365a63Cd0F99edddAEa18370b83Dc7

The contract's current community wallet is also a Gnosis Safe contract with 2/3 signers, eth:0xE234857A497deCf6239911C8190c195a0eaBB638.

- The 3 multisign addresses:
    1. EOA:0xA5CbE8c764323f78c023F9342Dc867b10fb57C3f,
    2. EOA:0x9ea99109E1b1Aa7e83C028391FB2D038fa6a4174,
    3. EOA:0x73524D7f64365a63Cd0F99edddAEa18370b83Dc7.

[CertiK, 11/26/2025]: While this strategy has indeed reduced the risk, it's crucial to note that it has not completely eliminated it. CertiK strongly encourages the project team to periodically revisit the private key security management of all above-listed addresses.

AFA-03 | Centralized Control Of Token Withdrawal

Category	Severity	Location	Status
Centralization	**Centralization**	**DailyEarnLockUp.sol (pre): 774**	**2/3 Multi-Sig**

Description

The `withdrawByAdmin` function in the `DailyEarnLockUp` contract is considered a major centralization risk. Any compromise to the `owner` accounts may allow the hacker to take advantage of this authority and withdraw all user locked funds from the `DailyEarnLockUp` contract.

Additionally, addresses in the authorized list can change a user's membership type to `Unknown`. As a result, the user's asset withdrawal is blocked because the transaction reverts when validating the membership type during the withdrawal process.

```
294     function _validateWithdrawRequest(
295         uint256 withdrawnAmount,
296         string memory userId,
297         address walletAddress,
298         bool isImmediate
299      ) private view {
300         ....
301
302         MembershipType membershipType = userIdToMembershipType[userId];
303         require(
304 @>          membershipType != MembershipType.Unknown,
305             "userId does not have any associated membership type"
306         );
```

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;

  AND

- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[AVA Foundation, 11/26/2025]: The team acknowledged the issue and adopted the multisign solution to ensure the private key management process at the current stage. The `DailyEarnLockUp` contract has transferred the ownership to a Gnosis Safe contract with 2/3 signers in the sensitive function signing process.

- Grant Role transaction hash for Gnosis Safe: 0x34595881f1d17f33e87c720c632cbd32fe021b0c15ab9f1ad5a9d54e9d0f56dd, Gnosis safe contract address: eth:0x58653987Ff3837ADBE6383F670f6935fcDE521b0
- The 3 multisign addresses:
  1. EOA:0xA5CbE8c764323f78c023F9342Dc867b10fb57C3f]
  2. EOA:0x9ea99109E1b1Aa7e83C028391FB2D038fa6a4174
  3. EOA:0x73524D7f64365a63Cd0F99edddAEa18370b83Dc7

The contract's current community wallet is also a Gnosis Safe contract with 2/3 signers, eth:0xE234857A497deCf6239911C8190c195a0eaBB638.

- The 3 multisign addresses:
  1. EOA:0xA5CbE8c764323f78c023F9342Dc867b10fb57C3f,
  2. EOA:0x9ea99109E1b1Aa7e83C028391FB2D038fa6a4174,

3. EOA:0x73524D7f64365a63Cd0F99edddAEa18370b83Dc7.

[CertiK, 11/26/2025]: While this strategy has indeed reduced the risk, it's crucial to note that it has not completely eliminated it. CertiK strongly encourages the project team to periodically revisit the private key security management of all above-listed addresses.

AFA-13 | Potential Risk Of User Fund Theft

Category	Severity	Location	Status
Design Issue, Centralization	Centralization	DailyEarnLockUp.sol (fix-1119): 773	2/3 Multi-Sig

Description

The `updateUserIdToWalletAddress()` function assigns a new wallet address to an existing user ID associated with a valid user. Any compromise of the central authority addresses could allow an attacker to redirect wallet bindings and steal user funds.

```
773     function updateUserIdToWalletAddress(
774         string calldata userId,
775         address walletAddress
776     ) external isAuthorized {
777         require(walletAddress != address(0), "Invalid wallet address");
778         require(bytes(userId).length > 0, "Invalid userId");
779         require(
780             bytes(walletAddressToUserId[walletAddress]).length == 0,
781             "Wallet address in use"
782         );
783 
784         // If userId already has a wallet address, we need to clear it first.
785         if (userIdToWalletAddress[userId] != address(0)) {
786             walletAddressToUserId[userIdToWalletAddress[userId]] = "";
787         }
788 
789         userIdToWalletAddress[userId] = walletAddress;
790         walletAddressToUserId[walletAddress] = userId;
791 
792         emit EvtUpdateUserIdToWalletAddress(userId, walletAddress);
793     }
```

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[AVA Foundation, 11/26/2025]: The team acknowledged the issue and adopted the multisign solution to ensure the private key management process at the current stage. The `DailyEarnLockUp` contract has transferred the ownership to a Gnosis Safe contract with 2/3 signers in the sensitive function signing process.

- Grant Role transaction hash for Gnosis Safe: 0x34595881f1d17f33e87c720c632cbd32fe021b0c15ab9f1ad5a9d54e9d0f56dd, Gnosis safe contract address: eth:0x58653987Ff3837ADBE6383F670f6935fcDE521b0
- The 3 multisign addresses:
  1. EOA:0xA5CbE8c764323f78c023F9342Dc867b10fb57C3f]
  2. EOA:0x9ea99109E1b1Aa7e83C028391FB2D038fa6a4174
  3. EOA:0x73524D7f64365a63Cd0F99edddAEa18370b83Dc7

The contract's current community wallet is also a Gnosis Safe contract with 2/3 signers, eth:0xE234857A497deCf6239911C8190c195a0eaBB638.

- The 3 multisign addresses:
    1. EOA:0xA5CbE8c764323f78c023F9342Dc867b10fb57C3f,
    2. EOA:0x9ea99109E1b1Aa7e83C028391FB2D038fa6a4174,
    3. EOA:0x73524D7f64365a63Cd0F99edddAEa18370b83Dc7.

[CertiK, 11/26/2025]: While this strategy has indeed reduced the risk, it's crucial to note that it has not completely eliminated it. CertiK strongly encourages the project team to periodically revisit the private key security management of all above-listed addresses.

AFA-04 The `maxLockupAmountPerTotal` Can Be By-Passed When Updating The Users' Membership

Category	Severity	Location	Status
Logical Issue	Major	DailyEarnLockUp.sol (pre): 338, 738, 807	Resolved

Description

The `withdrawRequestSubmit` splits the user's principal into two parts, the `stats.lockedAmount` and the `stats.withdrawReqAmount`:

```
function withdrawRequestSubmit(
        //code snippet
        LockupStats storage stats = userIdToLockupStats[userId];
        stats.lockedAmount -= withdrawnAmount;
        stats.withdrawReqTimeStamp = block.timestamp;
        stats.withdrawReqAmount = withdrawnAmount;
        //code snippet
```

Then, the function `withdrawRequestCancel()` sum those two parts rewards, by indirectly calling `getEarnAmountFromLockupStats()` and the `getEarnAmountFromWithdrawRequest()` function:

```
    //withdrawRequestCancel->_claimReward->getEarnAmountFromLockupStats
    function getEarnAmountFromLockupStats(
        address walletAddress
    ) public view returns (uint256, uint256) {
        string memory userId = _getUserId(walletAddress);
        LockupStats memory stats = userIdToLockupStats[userId];

        return
            _getEarnAmount(
                walletAddress,
                stats.lockedAmount,
                stats.lastTimeStamp
            );
    }

    //withdrawRequestCancel->_claimRewardFromWithdrawRequest-
>getEarnAmountFromWithdrawRequest
    function getEarnAmountFromWithdrawRequest(
        address walletAddress
    ) public view returns (uint256, uint256) {
        string memory userId = _getUserId(walletAddress);
        LockupStats memory stats = userIdToLockupStats[userId];

        return
            _getEarnAmount(
                walletAddress,
                stats.withdrawReqAmount,
                stats.withdrawReqTimeStamp
            );
    }
```

Though, there is a check in the `_validateWithdrawRequest` function tries to ensure that the sum of `stats.lockedAmount` and the `stats.withdrawReqAmount` share the same cap:

```
function _validateWithdrawRequest(
    uint256 withdrawnAmount,
    string memory userId,
    address walletAddress,
    bool isImmediate
) private view {
    //codesnippet
    if (!isImmediate) {
        require(
            stats.withdrawReqTimeStamp == 0,
            "One withdrawal request already exists"
        );
    }
    //codesnippet
```

But, the functions(The `updateUserIdToMembershipType` and the `setMembershipTypeToCondition` function) that are able to update the membership can still lead to the cap `maxLockupAmountPerTotal` being by-passed.

Scenario

Exploit steps:

- Setup: Alices has `lockedAmount` L that is double of the current membership cap C = `membershipTypeToCondition[M].maxLockupAmountPerTotal` (like a membership downgrade or owner reducing C).
- Step 1 : Alice calls `withdrawRequestSubmit` with `withdrawnAmount` as C so that both parts(the `stats.lockedAmount` and the `stats.withdrawReqAmount`) are C.
- Step 2 : Alice waits t seconds to accrue rewards on both buckets.
- Step 3 : Alice calls `withdrawRequestCancel`. The function will:
    - Pay _claimReward on the remaining locked part (base min(remaining, C))
    - Pay _claimRewardFromWithdrawRequest on the pending part (base min(W, C))
    - Merge the two buckets back
- Step 4 : Repeats step 2 and step 3 get the extra rewards.

Example: Suppose C = 10,000 and the user's L = 20,000 after a legitimate membership downgrade or the owner reducing C. By submitting W = 10,000 and later cancelling, the user is paid interest over t seconds on 10,000 (remaining) + 10,000 (pending) = 20,000 instead of being capped at 10,000.

Proof of Concept

PoC:

```
function setUp() public {
    token = new ERC20Mock();
    lockUp = new DailyEarnLockUp(address(token), COMMUNITY);

    // register test user so it can call lockup/claim
    lockUp.grantAuthorized(address(this));
    lockUp.initUserData(
        USER_ID,
        DailyEarnLockUp.MembershipType.SmartSteel,
        USER,
        0
    );

    token.mint(USER, USER_BALANCE);
    vm.prank(USER);
    token.approve(address(lockUp), type(uint256).max);
}

function testMaxLockupCapBypassViaMembershipUpdate() public {
    uint256 initialLock = 20_000 ether;
    // SmartSteel maxLockupAmountPerTotal: 10_000 * 1e18
    uint256 downgradedCap = 10_000 ether;

    // provide reward tokens so the contract can pay out earns
    token.mint(address(lockUp), USER_BALANCE);

    // upgrade to a membership type that allows a large lockup
    // SmartSilver maxLockupAmountPerTotal: 50_000 * 1e18
    lockUp.updateUserIdToMembershipType(
        USER_ID,
        DailyEarnLockUp.MembershipType.SmartSilver,
        0
    );

    vm.prank(USER);
    lockUp.lockup(initialLock);// Double of the cap

    // downgrade back to a restrictive membership with a low cap

    lockUp.updateUserIdToMembershipType(
        USER_ID,
        DailyEarnLockUp.MembershipType.SmartSteel,
        0
    );

    vm.prank(USER);

    lockUp.withdrawRequestSubmit(downgradedCap);
```

```
        vm.warp(block.timestamp + 2 days);

        (uint256 earnLocked, ) = lockUp.getEarnAmountFromLockupStats(USER);
        (uint256 earnWithdrawReq, ) = lockUp.getEarnAmountFromWithdrawRequest(USER);
        // 2 days' earn for a max lockup, that equals to 2739726027397260273
        uint256 earn = (downgradedCap *
            lockUp.getTotalApr(USER)) *
            2 days / (1e3 * 365 days);

        // each part has a rewards
        assertEq(earnLocked, earn, "locked bucket should accrue rewards after
time");
        assertEq(
            earnWithdrawReq,
            earn,
            "withdraw request bucket should accrue rewards after time"
        );

        uint256 balanceBefore = token.balanceOf(USER);
        vm.prank(USER);
        lockUp.withdrawRequestCancel();
        uint256 balanceAfter = token.balanceOf(USER);
        // The reward exceeds the cap
        assertEq(
            balanceAfter - balanceBefore,
            earnLocked + earnWithdrawReq,
            "Cancel pays both locked and pending buckets"
        );
    }
```

```
Ran 1 test for test/DailyEarnLockUp.t.sol:DailyEarnLockUpTest
[PASS] testMaxLockupCapBypassViaMembershipUpdate() (gas: 299168)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 3.89ms (904.58µs CPU
time)
```

Recommendation

Consider refactoring the reward calculation logic to enforce the cap across the combined total of `lockedAmount` and `withdrawReqAmount`, rather than applying the cap independently to each bucket.

Alleviation

[AVA Foundation, 11/19/2025]: The team heeded the advice and resolved the issue by refactoring the reward calculation logic to enforce the cap across the combined total of `lockedAmount` and `withdrawReqAmount`, in commit 4fcd16c6eeb66c12cab71f248cbd434d5fddbf4f

AFA-05 | Missing Check If The User Id And Wallet Address Are Used

Category	Severity	Location	Status
Logical Issue	Medium	DailyEarnLockUp.sol (pre): 690	Partially Resolved

Description

The `updateUserIdToWalletAddress` function allows an authorized role to update the wallet address for a specific user ID, it neither checks if the `userId` is used nor checks if the `walletAddress` is used, leads to side effects.

Missing Check If User Id is Used

If the user id is used, the `updateUserIdToWalletAddress` function simply clear the `walletAddressToUserId[userIdToWalletAddress[userId]]`. As a result, the previous wallet associated to the user id loses the access of the wallet.

Exploit steps:

1. Ensure there is an existing mapping: userIdToWalletAddress["Alice"] = A and walletAddressToUserId[A] = "Alice".
2. Call the updateUserIdToWalletAddress("Alice", B)
3. Post-state:
    - walletAddressToUserId[A] = ""
    - walletAddressToUserId[B] = Alice, userIdToWalletAddress[Alice] = B

As a result, wallet A loses access to the user Alice, due to any authorized user can invoke the `updateUserIdToWalletAddress` function, it is a risk that users' wallet would lost access to their fund without any notification if any authorized invoke the function with the users' user id.

Missing Check If Wallet Address is Used

Exploit steps:

1. Ensure there is an existing mapping: userIdToWalletAddress["Alice"] = A and walletAddressToUserId[A] = "Alice".
2. Call updateUserIdToWalletAddress("Bob", A).
3. Post-state:
    - userIdToWalletAddress["Alice"] == A (unchanged; not cleaned)
    - walletAddressToUserId[A] == "Bob" (overwritten),
    - userIdToWalletAddress["Bob"] == A This leaves two userIds ("Bob" and "Alice") effectively pointing to the same wallet A through different directions of the mapping, results in Alice losing her funds.

it does not check if the wallet address(the `walletAddressToUserId[walletAddress]`) is already mapped to a previous user. If the same walletAddress is assigned to multiple user IDs, previous users linked to that address will lose access to their

funds.

Recommendation

Recommend refactoring the code to mitigate the potential risk of user fund loss.

Alleviation

[AVA Foundation, 11/19/2025]: The team heeded the advice and resolved one of the issues, Missing Check If Wallet Address is Used, by checking if the wallet address is used or not, in the commit 60b8ebba011b0187a4394ce90a502afa2133766b

[AVA Foundation, 11/20/2025]: The team acknowledged the finding of Missing Check If User Id is Used, and replied that's our design choice because we allow the currently-used "userId" to be updated with a new wallet address. Thus, the previous wallet address will be set to empty "userId" so that it losts any access. This is done by one of our authorized accounts.

AFA-06 Updating An Exist Membership Type's Lockup Condition Affects Users' Rewards

Category	Severity	Location	Status
Volatile Code	Medium	DailyEarnLockUp.sol (pre): 807	Acknowledged

Description

The `setMembershipTypeToCondition` function allows the contract owner to modify the lockup conditions for an existing membership type. Since the computation of user rewards in the `_getEarnAmount` function is based on parameters such as `lockedAmountChecked` and total APR, any change to the membership type's lockup conditions directly influences the value of rewards accrued by users sharing that membership type.

```
function _getEarnAmount(
    address walletAddress,
    uint256 lockedAmount,
    uint256 lastTimeStamp
) private view returns (uint256, uint256) {
    //code snippet

    uint256 lockedAmountChecked = _getLockedAmountChecked(
        walletAddress,
        lockedAmount
    );

    // Determine the time elapsed since the last claim.
    uint256 timeElapsedSinceLastClaim = block.timestamp - lastTimeStamp;

    uint256 earn = (lockedAmountChecked *
        getTotalApr(walletAddress) *
        timeElapsedSinceLastClaim) / (PERCENT_FACTOR * 365 days);

    //code snippet
}
```

Specifically, parameters like `maxLockupAmountPerTotal`, `apr`, `aprExtraPerNFT`, and `maxAllowedAmountNFTs`—all part of the lockup conditions—are used in the reward calculation logic as shown below:

```
    function _getLockedAmountChecked(
        address walletAddress,
        uint256 lockedAmount
    ) private view returns (uint256) {
        //code snippet
        LockupCondition
            memory membershipTypeCondition = membershipTypeToCondition[
                membershipType
            ];

        if (lockedAmount > membershipTypeCondition.maxLockupAmountPerTotal) {
            return membershipTypeCondition.maxLockupAmountPerTotal;
        }
        //code snippet
    }

function getTotalApr(address walletAddress) public view returns (uint256) {
        //code snippet
        uint256 extraEarnRate = 0;
        if (membershipType == MembershipType.SmartDiamond) {
            uint256 amountNFTs = userIdToAmountNFTs[userId];
            if (amountNFTs > membershipTypeCondition.maxAllowedAmountNFTs) {
                amountNFTs = membershipTypeCondition.maxAllowedAmountNFTs;
            }
            extraEarnRate = (membershipTypeCondition.aprExtraPerNFT *
                amountNFTs);
        }

        return (membershipTypeCondition.apr + extraEarnRate);
    }
```

For instance, if the owner reduces `maxLockupAmountPerTotal` from `200_000 * 1e18` to `100_000 * 1e18` for the `SmartDiamond` membership type, a user who previously locked `200_000 * 1e18` tokens would, after the update, only accrue further rewards as if they had locked half that amount. This may result in users receiving lower rewards than expected for the period in which they met previous criteria.

Without distributing pending rewards before altering these parameters, existing users may have their accrued rewards recalculated under the new, potentially less favorable conditions. This approach can lead to unexpected reward reductions for users who have met lockup requirements prior to the update.

Recommendation

It's recommended that before modifying the lockup conditions of an existing membership type, the contract should ensure all users with that membership type automatically receive any pending rewards accrued under the previous conditions. This prevents users from having their previously earned rewards negatively impacted by subsequent changes. Alternatively, the contract can restrict updates to lockup conditions for membership types with active users, or implement logic to track rewards based on the conditions in effect during the accrual period for each user.

Alleviation

[AVA Foundation, 11/19/2025]: We know and accept to live with this kind of risk.We will inform all the users to claim their rewards before we modify the membership type condition.

AFA-07 | Missing Zero Address Check

Category	Severity	Location	Status
Volatile Code	Minor	DailyEarnLockUp.sol (pre): 788	Resolved

Description

Address is not validated before token transfer, potentially allowing the use of zero addresses and leading to unexpected behavior. For example, transferring tokens to a zero address can result in a permanent loss of those tokens.

```
    function takeImmediateWithdrawalFeeCollected(
        address recipient,
        uint256 amount
    ) external isOwner {
        require(amount > 0, "Invalid amount");
        require(
            immediateWithdrawalFeeCollected >= amount,
            "Insufficient fee collected"
        );

        immediateWithdrawalFeeCollected -= amount;
        lockupToken.safeTransfer(recipient, amount);

        emit EvtTakeImmediateWithdrawalFeeCollected(recipient, amount);
    }
```

Recommendation

It is recommended to add a zero-check for the passed-in address value to prevent fund loss.

Alleviation

[AVA Foundation, 11/19/2025]: The team heeded the advice and resolved the issue by adding a zero-check for the passed-in address value in commit 1b6266813b40c250935f42730c2ca4d2446b6e90

AFA-08 Missing Unknown Membership Type Validation In `updateUserIdToMembershipType()` Function

Category	Severity	Location	Status
Coding Issue	Minor	DailyEarnLockUp.sol (pre): 740	Resolved

Description

The `updateUserIdToMembershipType()` function is intended to update a user's membership type. However, it currently lacks logic to check whether the provided `membershipType` argument is set to `Unknown`. As a result, the user's membership type cannot be updated, and they are also unable to withdraw assets if it is set to `Unknown`.

```
738     function updateUserIdToMembershipType(
739         string memory userId,
740         MembershipType membershipType,
741         uint256 amountNFTs
742     ) external isAuthorized {
743         require(bytes(userId).length > 0, "Invalid userId");
744         require(
745 @>          userIdToMembershipType[userId] != MembershipType.Unknown,
746             "User data not yet set. Pls use the function initUserData"
747         );
748         require(
749             userIdToMembershipType[userId] != membershipType,
750             "UserId already has this membership type"
751         );
752
753         ....
754
755
// If the currently-locked amount > max lockup cap, user will need to manually
submit withdrawal req

756
757         userIdToMembershipType[userId] = membershipType;
```

Notice: A similar issue also exists in `initUserData()` function.

Recommendation

Recommend implementing logic to prevent the user's membership type from being updated to `Unknown`.

Alleviation

[AVA Foundation, 11/19/2025]: The team heeded the advice and resolved the issue by adding `Unknown` membership type validation in commit a1956a3eee93b6246fd4aaceaefc8c8db6863b9f.

AFA-01 | Design Logic Of The Reward Resource

Category	Severity	Location	Status
Design Issue	Informational	DailyEarnLockUp.sol (pre): 597	Resolved

Description

The `DailyEarnLockUp` contract allows users to deposit funds and earn rewards. However, according to the contract logic, user deposits are the sole source of assets, which is insufficient to sustain the staking rewards.

```
587         (
588             uint256 earn,
589             uint256 timeElapsedSinceLastClaim
590         ) = getEarnAmountFromLockupStats(walletAddress);
591         if (earn > 0) {
592
// Update lastTimeStamp to the timestamp rounded up to the last full day.
593             LockupStats storage stats = userIdToLockupStats[userId];
594             stats.lastTimeStamp += timeElapsedSinceLastClaim;
595             stats.earnedAmount += earn;
596
597  @>         lockupToken.safeTransfer(walletAddress, earn);
598             emit EvtClaim(walletAddress, earn);
599             return earn;
600         } else {
```

Recommendation

The audit team would like to inquire with the AVA Foundation regarding the design logic of reward resource.

Alleviation

[AVA Foundation, 11/19/2025]: That's our design choice. We know and accept this risk.

We can monitor the AVA balance on the contract to ensure that it still has at least a certain minimum balance for daily rewards.

This minimum balance can be determined based on the totally-locked amount and the average APR per day.

AFA-09 | The Comparison Operator Prefer To Use '>=' Instead Of '>'

Category	Severity	Location	Status
Logical Issue	Informational	DailyEarnLockUp.sol (pre): 409	Resolved

Description

In the `withdraw` function, if the `isImmediate` is false, the function checks whether the minimum withdrawal period requirement is satisfied using the following condition::

```
require(
    block.timestamp >
        stats.withdrawReqTimeStamp + condition.withdrawPeriod,
    "Minimum withdraw period not met"
);
```

Using the `>` operator requires the caller to wait until the current block timestamp strictly exceeds the sum of `withdrawReqTimeStamp` and `withdrawPeriod`. This means users can only withdraw after the exact withdrawal period has passed, not at the precise moment it ends. Replacing `>` with `>=` aligns the logic with typical expectations, allowing withdrawals as soon as the period concludes and matching common time-based restriction patterns.

Recommendation

It is recommended that update the comparison operator to meet the design.

Alleviation

[AVA Foundation, 11/19/2025]: The team heeded the advice and resolved the issue by updating the comparison operator in commit 42956105d47a43130b898bb78ec92336776988a9

AFA-10 The `amountNFTs` Only For The `SmartDiamond` Membership Type.

Category	Severity	Location	Status
Volatile Code	Informational	DailyEarnLockUp.sol (pre): 658	Resolved

Description

The `initUserData` function sets user data including `userId`, `amountNFTs`, etc. However, the `amountNFTs` is only for the `SmartDiamond` membership type, but the `initUserData` does not check for it.

```
    // Mapping from userId to amount of NFTs owned only for SmartDiamond membership type.
    mapping(string => uint256) public userIdToAmountNFTs;

    function initUserData(
        string memory userId,
        MembershipType membershipType,
        address walletAddress,
        uint256 amountNFTs
    ) external isAuthorized {
        //code snippet
        userIdToAmountNFTs[userId] = amountNFTs;
        //code snippet
    }
```

Recommendation

It is recommended that only update the `amountNFTs` for the `SmartDiamond` membership type.

Alleviation

[AVA Foundation, 11/19/2025]: The team heeded the advice and resolved the issue by only updating the `amountNFTs` for the `SmartDiamond` membership type in commit a1956a3eee93b6246fd4aaceaefc8c8db6863b9f

AFA-11 | The Potential Fee-On-Transfer Token

Category	Severity	Location	Status
Volatile Code	Informational	DailyEarnLockUp.sol (pre): 136	Resolved

Description

In lockup, the contract trusts the `lockedAmount` parameter to update accounting, without verifying how many tokens were actually received. Specifically, it executes:

```
lockupToken.safeTransferFrom(msg.sender, address(this), lockedAmount);
userIdToLockupStats[userId].lockedAmount += lockedAmount;
totalLockedAmount += lockedAmount;
```

There is no balance delta check. If `lockupToken` is fee-on-transfer/deflationary token, the contract credits the user with a larger principal than it received. This over-credited principal is then used by `_claimReward` (called inside `lockup`) to compute and transfer rewards. As a result, a user can earn rewards on “phantom” tokens that were never deposited.

Recommendation

The audit team would like to confirm with the AVA Foundation that is the `lockupToken` an fee-on-tranfer Token

Alleviation

[AVA Foundation, 11/19/2025]: We ensure that the "lockupToken" is NOT fee-on-transfer/deflationary token.

AFA-12 | Unused Function

Category	Severity	Location	Status
Code Optimization	Informational	DailyEarnLockUp.sol (fix-1119): 357, 357	Resolved

Description

After the fix commit a1956a3eee93b6246fd4aaceaefc8c8db6863b9f, the private function `_claimRewardFromWithdrawRequest` is deprecated, it can be removed.

Recommendation

It is recommended that removing the deprecated function.

Alleviation

[AVA Foundation, 11/21/2025]: The team heeded the advice and resolved the issue by removing the deprecated function, in commit a25e0028d5ce293abc8c20e72add11de8d2ba0ba

APPENDIX | AVA FOUNDATION AUDIT

Finding Categories

Categories	Description
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Elevating Your Web3 Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is the largest blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.