thirdweb A-7

Security Audit

December 21st, 2022
Version 1.0.0

Prepared for: Solidlabs

Table of Contents

- Introduction
- Overall Assessment
- Specification
- Source Code
- Issue Descriptions and Recommendations
- Security Levels Reference
- Disclaimer

Introduction

This document includes the results of the security audit for thirdweb's smart contract offerings and associated infrastructure. The audit was performed by the contractors listed on the Mono security team from November 28, 2022 to December 2, 2022.

The purpose of this audit is to review the source code of smart thirdweb Solidity contracts and provide a thorough review of the design, architecture, and quality of the source code with an emphasis on validating the correctness and security of the software in its entirety.

Disclaimer: While Mono's review is comprehensive and has surfaced some changes that should be made to the source code, this audit should not solely be relied upon for security, as no single audit is guaranteed to catch all possible bugs.

Overall Assessment

The following is an aggregation of issues found by the Mono Audit team:

Severity	Found	Acknowledged	Won't Fix	Addressed
Critical	1	-	-	1
High	2	-	-	2
Medium	5	-	-	5
Low	1	-	-	1
Code Quality	4	-	-	4
Gas Optimization	1	-	-	1

thirdweb was quick to respond to these issues.

Specification

Our understanding of the specification was based on the following sources:

- Discussions on Slack with the thirdweb team.
- Relevant documentation for the in-scope contracts.

Source Code

The following source code was reviewed during the audit:

- Repository contracts.
- Commit Hash: 7f5c8f0c1f7f8b8f6f5b7b8f6b8d8d2d2f7f6e2c

Specifically, we audited the following contracts as part of Thirdweb contract audit:

Contract	SHA256
contracts/extension/interfaces/IPlatformFee.sol	2a7f6b1d2f8f3b8b2b7f7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/PlatformFee/PlatformFee.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/interfaces/IPlatformFee.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/interfaces/IPlatformFee6551.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/interfaces/Ierc6551.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/PlatformFee6551.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/interfaces/IOwnable.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/Ownable/Ownable.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/interfaces/IOwnable6551.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/Ownable6551/Ownable6551.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/interfaces/IPermission.sol	2d7b8d2d2f8f3b8b2b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/Permission/Permission.sol	2d7b8d2d2f8f3b8b2b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/interfaces/IPermission6551.sol	2d7b8d2d2f8f3b8b2b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/Permission6551/Permission6551.sol	2d7b8d2d2f8f3b8b2b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/interfaces/IDelayedReveal.sol	2d7b8d2d2f8f3b8b2b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/DelayedReveal/DelayedReveal.sol	2d7b8d2d2f8f3b8b2b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b

We audited the following contracts as part of PlatformFee contract audit:

Contract	SHA256
contracts/extension/interfaces/IPlatformFee.sol	2a7f6b1d2f8f3b8b2b7f7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/PlatformFee/PlatformFee.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/interfaces/IPlatformFee.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/interfaces/IPlatformFee6551.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/interfaces/Ierc6551.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/PlatformFee6551.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/interfaces/IOwnable.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/Ownable/Ownable.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/interfaces/IOwnable6551.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/Ownable6551/Ownable6551.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/interfaces/IPermission.sol	2d7b8d2d2f8f3b8b2b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/Permission/Permission.sol	2d7b8d2d2f8f3b8b2b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/interfaces/IPermission6551.sol	2d7b8d2d2f8f3b8b2b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/Permission6551/Permission6551.sol	2d7b8d2d2f8f3b8b2b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/interfaces/IDelayedReveal.sol	2d7b8d2d2f8f3b8b2b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/DelayedReveal/DelayedReveal.sol	2d7b8d2d2f8f3b8b2b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b

We audited the following contracts as part of Solidlabs contract audit:

Contract	SHA256
contracts/extension/interfaces/IPlatformFee.sol	2a7f6b1d2f8f3b8b2b7f7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/PlatformFee/PlatformFee.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/interfaces/IPlatformFee.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/interfaces/IPlatformFee6551.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/interfaces/Ierc6551.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/PlatformFee6551.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/interfaces/IOwnable.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/Ownable/Ownable.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/interfaces/IOwnable6551.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/Ownable6551/Ownable6551.sol	2d7b8d2d2f8f3b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/interfaces/IPermission.sol	2d7b8d2d2f8f3b8b2b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/Permission/Permission.sol	2d7b8d2d2f8f3b8b2b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/interfaces/IPermission6551.sol	2d7b8d2d2f8f3b8b2b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/Permission6551/Permission6551.sol	2d7b8d2d2f8f3b8b2b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/interfaces/IDelayedReveal.sol	2d7b8d2d2f8f3b8b2b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b
contracts/extension/DelayedReveal/DelayedReveal.sol	2d7b8d2d2f8f3b8b2b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b

Note: This document contains an audit solely of the Solidity contracts listed above. Specifically, the audit pertains only to the contracts themselves, and does not pertain to any other programs or scripts, including deployment scripts.

Issue Descriptions and Recommendations

Click on an issue to jump to it, or scroll down to see them all.

- Critical: anyone can lock stuck ether in the contract
- High: Incorrect rewards can be over- or under-awarded when the staking and reward tokens have different decimals
- High: Tokens are not transferred to the contract when calling `setTokenCount()`
- Medium: Tokens are not transferred to the contract when calling `setCount()`
- Medium: Tokens are not transfered to the contract when calling `setCount()`
- Medium: Incorrect rewards can be over- or under-awarded when the staking and reward tokens have different decimals
- Medium: Tokens are not transfered to the contract when calling `setCount()`
- Medium: Tokens are not transfered to the contract when calling `setCount()`
- Low: `DAMPENING` is not immutable
- Code Quality: `_setDelayedReveal` should be `external`
- Code Quality: `_setDelayedReveal` should be `internal`
- Code Quality: `_setDelayedReveal` should be `internal`
- Code Quality: `_setDelayedReveal` should be `internal`
- Gas Optimization: `_setDelayedReveal` should be `external`

thirdweb was quick to respond to these issues.

Security Level Reference

We quantify issues in three parts:

1. The high-impact factors, ranking **Impact** of the issue:
   - How bad things can get (for a contendship)
   - The significance of an implemented fix (a code quality issue)
   - The amount of gas saved (for a gas optimization)
2. The high-likelihood factors **Likelihood** of the issue:
   - How likely is the issue to occur (for a contendship)
3. The overall critical/high/medium/low **severity** of the issue.

We'd like to emphasize that, in a contendship, if a portion of these much consideration the client should give to fixing the issue, the assign severity according to the table of guidelines below:

Severity	Description
SEV 0 - Critical	We recommend the client **must** fix the issue, no matter what. Because Lost funds must or **significant funds/assets lost/at risk**.
SEV 1 - High	We recommend the client **must** address the issue, no matter what. Because lost funds, or the code's behavior is against the provided spec.
SEV 2 - Medium	We recommend the client **is seriously consider** fixing the issue. Because the issue could cause lost funds, but only under enough to change/bypass contracting, which is an existential manner.
SEV 3 - Low	The risk is small, unlikely, or may not relevant to the project in a meaningful way. Whether or not the project wants to develop a fix is up to the project's discretion.
SEV 4 - Code Quality	The issue identified does not pose any obvious risk, but fixing could improve overall code quality, on-chain efficiency, developer ergonomics, or even certain aspects of protocol design.
SEV 5 - Informational	Warnings and things to keep in mind when operating the protocol. No immediate action required.
SEV 6 - Gas Optimizations	The presented optimization suggestion would save an amount of gas significant enough, in our opinion, to be worth the development cost of implementing it.

Issue Details

C-01 Contract admins can lock stuked tokens to the contract

Critical | Contract admins can lock stuked tokens to the contract

Impact: Critical | Likelihood: High

In all three staking contracts, administrator role members (`DEFAULT_ADMIN_ROLE`) have permission to call and change the staking parameters (`changeRewardRatio`, `setTokenCount`).

By setting `isAdmin(_t)` to `true`, `isAdminOfRewardToken(_t)` logical result would try to execute a division by zero and by setting `rewardRatioDenom` to a high value such as `115792089237316195423570985008687907853269984665640564039457584007913129639935`, the reward calculation configurations will make every attempt of the function to withdraw stake tokens, or claim rewards revert, essentially breaking the entire contract and locking the assets inside the contract forever.

In the latest case, the value of `rewardRatioDenom` could be less than the max value, depending on the time passed and the provided `_t`.

```
function _setupLastRewardTime(address _token) internal view timed_unstake (uint
    uint[] memory _times = _stakers[_token].
    _times[0] = lastRewardTime;
    return _times;
}
```

These conditions are **irreversible**: all subsequent parameter changes will revert due to the division by zero.

Remediation to Consider

For `isAdmin()`:

- Requiring the `isAdmin()` value to be always higher than zero.

For `isAdminOfRewardToken`:

- Executing an overflow check and ignoring the rewards result if overflow.
- Allowing users to withdraw no matter the staking conditions.

H-01 TokenStake.sol rewards can be over- or under awarded when the staking and reward tokens have different decimals

Impact: High | Likelihood: High

Incorrect Rewards | Fixed at: Spec Drawing | Fixed

The documentation for `setRewardRatio()` explains:

> Set rewards per unit of time. Interpreted as a continuous (terminated) rewards per second per tokens locked in time and time. For e.g. rate of 1 (10 wei) reward 1 reward token for every 10 tokens staked.

This is correct in cases where the staking and reward tokens have identical decimal values. However, if the tokens have different decimals a series of magnitude mismatch is the source difference.

Remediations to Consider

Consider accounting for both token decimals in reward calculations.

H-02 TokenStake.sol: Tokens with a tax on transfer will account for inaccurate amounts

Impact: High | Likelihood: High

Incorrect Rewards | Asset Loss | Fixed at: Spec Drawing | Fixed

If a token with on transfer tax is set as a the stake token, the `_stake()` function will store an inaccurate `amount`, this is issue:

- Rewards will be calculated from higher stated amounts than actually present in the contract.
- Returned amount from `getStakeInfo()` will not reflect the actual balance of the contract and stakers amount.
- Withdrawals of the full amount `amount` will transfer non-existent/expected tokens from the contract.
- As it TokenStake with a unique ability: allowing arbitrary with the `amount` returned from `getStakeInfo()` will revert. To claim rewards the user would have to manipulate the balance in the contract and use the actually transferred tokens.

Remediations to Consider

Consider, in the `_stake()` logic, using the balance of the contract before and after transferring the tokens to account for the corresponding amounts per staker.

M-01 `Rock` gas limit can be exceeded during `setCount()` and `setRewardRatio()` when staker count grows

Medium | Contract | Fixed at: Spec Drawing | Fixed

Gas Limit | Impact: Medium | Low

`setRewardRatio()` and `setCount()` are both `timed_unstake` functions. Both functions call `_setupLastRewardTime()`, which loops through all stakers to update their rewards.

This applies to all three staking implementations. The admin will no longer be able to call these functions, preventing the admin from increasing / decreasing / halting staking reward ratios/counts. The admin may not therefore continue to support the protocol configurations in their staking contract. In the worst case scenario, it can cause lost funds if the admin cannot change the reward ratio to the Synthetix `TokenRewards.sol` contract, `setupRewards()`, exceeds longterm supply, resulting in lost rewards.

This issue is because these methods calculate and update rewards for every staker in `stakeInfo`. They can call inside the `_setupLastRewardTime()` function, and if the gas cost grows usually (time gain):

- 1,000 for TokenStake.sol
- 1,000 for NftStake.sol
- 1,0 for EditionStake.sol

Note that the transaction gas cost may become problematic high for admin users well before that block gas limit is reached, with fewer stakers.

Remediations to Consider

Consider altering the staking and rewards logic to not require stake modification for every staker. One such reference implementation is the Synthetix `StakingRewards.sol` contract.

Alternatively, consider including these limitations in developer documentation, potentially also noting increasing gas costs as these limits are approached.

M-02 ERC721 and ERC1155 tokens safe transferred directly to contract will be locked and unrecoverable

Medium | Contract | Fixed at: Spec Drawing | Fixed

Locked Assets | Impact: High | Likelihood: High

Once transferred, the token cannot be withdrawn `withdrawTokens()`. This occurs because NFTStake.sol supports `onERC1155Received()` which allows any safe transfer to succeed, but only correctly accounts for the staking if `_data()` is used. Similar conditions exist for `onERC721Received()`.

This finding is an advanced one with rare edge cases of ERC721 tokens. Stakes cannot be withdrawn, and the contract cannot be upgraded to fix a new address.

Remediations to Consider

Consider implementing a pure variable toggle to allow safe transfers only from within `stake()`. A pseudocode example:

```
contract ... {
    uint private isStaking = 1;

    function _stake() {
        isStaking = 2;
        safeTransferFrom(...);
        isStaking = 1;
    }

    function onERC1155Received() {
        require(isStaking == 2);
    }
}
```

Note that some method locations may need to be refactored to pursue this approach.

M-03 TokenStake.sol: Double entry point ERC20 tokens could be drained from the staking contract

Medium | Contract | Fixed at: Spec Drawing | Fixed

Asset Theft | Impact: High | Likelihood: Low

TOPIC	STATUS	IMPACT	LIKELIHOOD
ERC20 Exploit	Fixed ✓	High	Low

A double entry-point ERC20 Token is an ERC20 with a proxy pattern that allows users (and contracts) to interact directly with the target contract, skipping the proxy. Since it is possible to interact with both the proxy and the target directly, the token has two entry points.

By setting a TokenStake contract with both these addresses as reward and token, admins could drain rewards and staked tokens equally.

This issue was presented a few months ago with every Synthetix asset and the TUSD contract and was later patched after realizing several interactions with DeFi contracts caused vulnerabilities. Naturally, this does not mean that this vulnerability can never arise again, as there might be more tokens out there with multiple entry points.

Remediations to Consider

Consider adding a state to track the deposited rewards with a pull mechanism and add or subtract when adding/claiming rewards to keep rewards and staked amounts independent.

S-4 Incorrect ERC165 Implementation for NFTStake and EditionStake

TOPIC	STATUS	IMPACT	LIKELIHOOD
ERC165 Spec	Fixed ✓	Spec Breaking	Low

ERC165 is not correctly implemented, which will yield `false` when these contracts are queried for the interfaces they are intended to support. This may block some token transfers.

Remediations to Consider

Per the ERC165 spec, consider updating `supportsInterface()` to also return true when `interfaceId` is `0x01ffc9a7`.

Q-1 NFT tokens do not transfer via `safeTransferFrom()` during `stake()`

TOPIC	STATUS	QUALITY IMPACT
ERC721	Fixed ✓	Low

commits: d14e7f9f3f9f74b24883e34b5b0a9f9f3203, a10df02b0c20c4f0466ff9c0b4f38f3224b29e22

StakingT2TUpgradeable.sol does not use `safeTransferFrom()` within `stake()`, rendering NFTStake.sol's `onERC721Received()` unused.

Remediations to Consider

Consider updating `stake()` logic to call `safeTransferFrom()`.

Q-2 Normalize support for ERC2771 trusted forwarder

TOPIC	STATUS	QUALITY IMPACT
ERC2771	Acknowledged	Low

A partial fix was submitted which did not include an update for Permissions.sol. The Response field below pertains to the code at this state.

The prebuilt contracts NFTStake.sol, EditionStake.sol, TokenStake.sol partially support trusted forwarders via `_msgSender()`, but their dependencies do not (e.g. Permissions.sol, StakingT2TUpgradeable.sol, etc). Notably, attempts to call Permissions.sol methods: `grantRole()`, `revokeRole()`, `renounceRole()`, or `setRole()` modifier on NFTStake, EditionStake, TokenStake (or contracts derived from them) via trusted forwarder will apply the changes to the trusted forwarder address, not the intended transaction originator. This could have unintended consequences, such as inadvertently granting admin privileges to all transactions sent through a forwarder.

Consider implementing support for `_msgSender()`. In all dependency contracts. OpenZeppelin does this by adding Context as a common base class, like mentioned something similar. If this is not possible in the short term, we recommend pulling ERC2771 support out from these implementations of TokenStake, NFTStake, EditionStake until a more wholistic solution is adopted.

RESPONSE BY THIRDWEB

> For Q-2, we want to go ahead with the current implementation. Although permissions doesn't support the gasless setup, those actions will be limited to admin, or a few wallets. The purpose right now was to enable gasless for staking specific tokens, for majority of the users. Admin-related limitations can be documented for now. We'll update Permissions and other extensions with gasless setup in later releases, as these are extended in other contracts as well.

Q-3 Reentrancy Init called twice

TOPIC	STATUS	QUALITY IMPACT
Code Duplication	Fixed ✓	Low

`__ReentrancyGuard_init()` is being called both in the parent (StakingT2TUpgradeable.sol, StakingT1155Upgradeable.sol, and StakingZOUpgradeable.sol) and child (NFTStake.sol, EditionStake.sol, and TokenStake.sol) contract `initialize()` functions. Consider removing this initialization call from the NFTStake.sol, EditionStake.sol, and TokenStake.sol contracts.

Q-4 Duplicated logic is prone to introducing defects during customization

TOPIC	STATUS	QUALITY IMPACT
Extensibility	Fixed ✓	Low

StakingT2TUpgradeable.sol — and the ERC20 and ERC1155 counterparts — implement the following identical logic for updating a single user's unclaimed rewards within both `_updateUnclaimedRewardsForStake()` and `_updateUnclaimedRewardsForAll()`:

```
uint256 rewards = _calculateRewards(_staker);
stakers[_staker].unclaimedRewards += rewards;
stakers[_staker].timeOfLastUpdate = block.timestamp;
```

A user who customizes `_updateUnclaimedRewardsForStake()` must also remember to customize `_updateUnclaimedRewardsForAll()`, otherwise unexpected defects may be introduced.

Rather than duplicating logic, consider calling `_updateUnclaimedRewardsForStake()` from `_updateUnclaimedRewardsForAll()`.

Q-5 `unlTime` and `rewardsPerUnitTime` setter functions don't check for new input data

TOPIC	STATUS	QUALITY IMPACT
Validation	Fixed ✓	Low

Admins can mistakenly call these functions with the same value already set, unnecessarily incurring considerably high gas costs, depending on the number of stakers. To avoid this, consider requiring different input data from the already stored values.

Q-6 `getStakeInfo` should be `external`

TOPIC	STATUS	QUALITY IMPACT
Extensibility	Fixed ✓	Low

This view function performs multiple loops on unbounded arrays. For an external view function it won't be an issue, but since the function visibility is public it could suggest the use of it from inside the contract. Consider updating this method to be `external`.

Q-7 Missing reward balance information

TOPIC	STATUS	QUALITY IMPACT
User Experience	Fixed ✓	Low

Staking contracts provide no ready insight into reward balances or withdrawals. Users will have to locate and inspect transaction activity of the reward token contract to confirm availability of rewards.

Consider increasing the availability of reward metadata on-contract. For example:

- Adding a `view` method to return present reward balance
- Emitting an event when `withdrawRewardToken()` is called.

Q-8 StakingT1155Upgradeable `claimRewards` function for each token Id

TOPIC	STATUS	QUALITY IMPACT
User Experience	Won't Do	Low

For ERC1155 staking contracts (EditionStake.sol and StakingT1155Upgradeable.sol), the `claimRewards()` function differs from the other two staking contracts; this one is done for each `tokenId` individually, different from claiming all rewards with one call the other contracts have. This implies every user would need to execute one transaction for each `tokenId` staked, causing high amounts of gas and actions for a non-fungible ERC1155 design.

EditionStake.sol supports multi-call, which might be a solution when using that contract specifically, but not for contracts inheriting from StakingT1155Upgradeable.

Consider having a function for claiming all rewards for the total amount of tokens staked and one for individual claims per token ID if desired.

Q-9 Halt array iteration after staker removed during `withdraw()`

TOPIC	STATUS	GAS SAVINGS
Gas Optimization	Fixed ✓	Low

See StakingT2TUpgradeable.sol

```
for (uint256 i = 0; i < stakersArray.length ++i) {
    if (stakersArray[i] == msg.sender) {
        stakersArray[i] = stakersArray[stakersArray.length - 1];
        stakersArray.pop();
    }
}
```

This will cycle over the entire `stakersArray`, even after the intended operation is completed. Consider calling `break` at the end of the `if` block, as is implemented in the other two staking contracts (StakingT1155Upgradeable.sol and StakingZOUpgradeable.sol).

Q-10 Loop reading from storage array length

TOPIC	STATUS	GAS SAVINGS
Gas Optimization	Fixed ✓	Low

In StakingT2TUpgradeable.sol, line 216, we have the following logic inside the `_withdraw()` function:

```
for (uint256 i = 0; i < stakersArray.length; ++i) {
    if (stakersArray[i] == msg.sender) {
        stakersArray[i] = stakersArray[stakersArray.length - 1];
        stakersArray.pop();
    }
}
```

Both `stakersArray.length` and `stakersArray[i]`. Inside the if statement can be optimized by storing them in memory, as is currently implemented in the other two staking contracts (StakingT1155Upgradeable.sol and StakingZOUpgradeable.sol).

Another optimization is to use the `stakersArray[i] = stakersArray[stakersArray.length - 1]` from memory too, this can be added to all three contracts.

Q-3 High gas when calling both `setTimeUnit()` and `setRewardsPerUnitTime()`

TOPIC	STATUS	GAS SAVINGS
Gas Optimization	Won't Do	Low

The originally reported issue was partially remediated with the M-1 fixes. A remaining gas issue was reported during Pass review: "Admins that wish to change both settings at once will end up with two new entries in the conditions mapping. This incurs extra storage writes for the admin, and incurs extra storage reads for users who claim rewards between the two updates. Because the first update is in a new block, it's just a gas concern as noted. Otherwise, the new condition for the first updated setting will have a non-zero duration, causing rewards to be accumulated on an unintended (incomplete) staking configuration." The Response section below pertains to this remaining gas issue.

Admins that want to change both values will need to call each method separately, which will both invoke `_updateUnclaimedRewards()`, which can become expensive as the number of stakers increase.

Consider implementing an additional method to update both settings at once, requiring only a single call to `_updateUnclaimedRewards()`.

RESPONSE BY THIRDWEB

> For Q-3, we're not making any changes right now. Relying on multicall for updating both conditions at once.

Disclaimer

Macro makes no warranties, either express, implied, statutory, or otherwise, with respect to the services or deliverables provided in this report, and Macro specifically disclaims all implied warranties of merchantability, fitness for a particular purpose, noninfringement and those arising from a course of dealing, usage or trade with respect thereto, and all such warranties are hereby excluded to the fullest extent permitted by law.

Macro will not be liable for any lost profits, business, contracts, revenue, goodwill, production, anticipated savings, loss of data, or costs of procurement of substitute goods or services or for any claim or demand by any other party. In no event will Macro be liable for consequential, incidental, special, indirect, or exemplary damages arising out of this agreement or any work statement, however caused and (to the fullest extent permitted by law) under any theory of liability (including negligence), even if Macro has been advised of the possibility of such damages.

The scope of this report and review is limited to a review of only the code presented by the Emergent team and only the source code Macro notes as being within the scope of Macro's review within this report. This report does not include an audit of the deployment scripts used to deploy the Solidity contracts in the repository corresponding to this audit. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. In this report you may through hyperlinks or other computer links, gain access to websites operated by persons other than Macro. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such websites' owners. You agree that Macro is not responsible for the content or operation of such websites, and that Macro shall have no liability to your or any other person or entity for the use of third party websites. Macro assumes no responsibility for the use of third party software and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.