

River4FUN

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES:

February 11th to February 18th, 2025

AUDITED BY:

fedebianu
oakcobalt

Contents

1	Introduction	2
1.1	About Zenith	3
1.2	Disclaimer	3
1.3	Risk Classification	3
<hr/>		
2	Executive Summary	3
2.1	About River4Fun	4
2.2	Scope	4
2.3	Audit Timeline	5
2.4	Issues Found	5
<hr/>		
3	Findings Summary	5
<hr/>		
4	Findings	6
4.1	Medium Risk	7
4.2	Low Risk	10
4.3	Informational	16

1

Introduction

1.1 About Zenith

Zenith assembles auditors with proven track records: finding critical vulnerabilities in public audit competitions.

Our audits are carried out by a curated team of the industry's top-performing security researchers, selected for your specific codebase, security needs, and budget.

Learn more about us at <https://zenith.security>.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2

Executive Summary

2.1 About River4Fun

River is building the circulatory system for crypto, connecting assets, liquidity, and yield across chains through satUSD, a cross-chain stablecoin infra powered by the omni-CDP

Users can deposit assets (ex. BTC, ETH, BNB) on any source chain and natively mint satUSD on any destination chain, earn, leverage, and scale — all without selling your assets.

River4FUN is the contribution layer of River. It rewards users for participation, content, and influence.

By staking tokens and engaging on X (Twitter), users earn River Pts that convert into \$RIVER at TGE. This aligns social activity with real protocol rewards.

2.2 Scope

The engagement involved a review of the following targets:

Target	satoshi-farm
Repository	https://github.com/Satoshi-Protocol/satoshi-farm/
Commit Hash	4685ceb41c671835f9a90a2882dad60f00a060e7
Files	satoshi-farm/src/core/*

2.3 Audit Timeline

February 11, 2025	Audit start
February 18, 2025	Audit end
February 26, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	2
Low Risk	4
Informational	3
Total Issues	9

3

Findings Summary

ID	Description	Status
M-1	Missing owner check in executeClaim enables short-term DoS attacks	Resolved
M-2	Excess msg.value not refunded in FarmManager	Resolved
L-1	Add _disableInitializers() call in the constructor	Resolved
L-2	Add additional checks when updating FarmConfig	Acknowledged
L-3	OFT dust removal causes permanent lzCompose failure in FarmManager	Resolved
L-4	Batch functions may revert if quote changed	Resolved
I-1	LzCompose() may revert because of pause mechanism	Acknowledged
I-2	Users will not be able to execute their claims when claim period ends	Resolved
I-3	Modify transferCallback() to call safeTransfer() when using self as sender	Acknowledged

4

Findings

4.1 Medium Risk

A total of 2 medium risk findings were identified.

[M-1] Missing owner check in `executeClaim` enables short-term DoS attacks

SEVERITY: Medium

IMPACT: Medium

STATUS: Resolved

LIKELIHOOD: Low

Target:

- [FarmManager.sol#L370](#)

Description:

The `FarmManager::executeClaim` function lacks validation that `msg.sender` is the owner of the claim being executed. This allows any user to front-run and execute pending claims belonging to other users, which can be used to DoS other claim functions.

Proof of Concept:

1. Alice calls `requestClaim` to initiate a claim for 100 tokens
2. Alice attempts to batch execute multiple claims via `stakePendingClaimCrossChain`
3. Bob front-runs Alice's batch transaction by calling `executeClaim` with Alice's claim parameters
4. Alice's batch transaction fails because her claim was already executed by Bob

This can be used to DoS the following operations:

- `stakePendingClaimCrossChain`
- `stakePendingClaim`
- `stakePendingClaimBatch`
- `stakePendingClaimCrossChainBatch`
- `executeClaimBatch`

Recommendations:

Consider allowing only the owner to execute its own claim.

River: Resolved with [@90d3d39906...](#) & [@9b73c34820...](#) - changed owner in `executeClaimBatch()` to `msg.sender`.

Zenith: Verified.

[M-2] Excess `msg.value` not refunded in `FarmManager`

SEVERITY: Medium

IMPACT: Medium

STATUS: Resolved

LIKELIHOOD: Medium

Target:

- [FarmManager.sol#L812](#)

Description:

The `FarmManager::claimAndStakeCrossChain` function accepts payments in native tokens for LayerZero fees but does not refund the excess `msg.value` sent by users. The function calls `quoteSend` to get the required fee and forwards exactly that amount to the `send` function, but any additional value sent is simply transferred to the `FarmManager` contract.

Proof of concept:

1. User calls `FarmManager::claimAndStakeCrossChain` with `msg.value` of 1 ETH
2. `quoteSend` determines actual required fee is 0.1 ETH
3. `send` function is called with exactly 0.1 ETH
4. The remaining 0.9 ETH becomes locked in the `FarmManager` contract with no way for the user to recover it

Recommendations:

Add a mechanism to refund the excess `msg.value` sent inside the `claimAndStakeCrossChain` function.

River: Resolved with [@77e3f760cf3...](#) & [@c3ada5934d...](#) - Added refund function

Zenith: Verified.

4.2 Low Risk

A total of 4 low risk findings were identified.

[L-1] Add `_disableInitializers()` call in the constructor

SEVERITY: Low

IMPACT: Low

STATUS: Resolved

LIKELIHOOD: Low

Target:

- [Farm.sol#L22](#)
- [FarmManager.sol#L42](#)

Description:

The Farm and FarmManager contracts inherit from `Initializable` but do not call `_disableInitializers()` in their constructors. This makes it possible to initialize the implementation.

Recommendations:

Add `_disableInitializers()` call in the constructor of the Farm and FarmManager contracts.

River: Resolved with [@8209c14cb8..](#) - added `_disableInitializers()` to both

Zenith: Verified.

[L-2] Add additional checks when updating FarmConfig

SEVERITY: Low

IMPACT: Low

STATUS: Acknowledged

LIKELIHOOD: Low

Target:

- [Farm.sol#L870](#)

Description:

While updating the FarmConfig there are a few additional sanity checks that could be added:

1. If the new rewardEndTime is less than or equal to lastUpdateTime, the computeInterval function will return 0 due to its internal check `endTime < lastUpdateTime`, effectively stopping reward accrual.
2. If the new rewardStartTime is set to after the lastUpdateTime, rewards will only start accruing from the new start time due to `currentTime - Math.max(lastUpdateTime, startTime)`; check, creating a gap in reward distribution.

Additionally, modifying depositStartTime and depositEndTime can break cross-chain functionality by making pending LayerZero messages unexecutable.

Recommendations:

Add validation check for rewardEndTime inside the `_checkFarmConfig` function.

```
function _checkFarmConfig(FarmConfig memory _farmConfig) internal view {
    if (_farmConfig.rewardEndTime < _farmConfig.rewardStartTime) {
        revert InvalidConfigRewardTime(_farmConfig.rewardStartTime,
            _farmConfig.rewardEndTime);
    }
    if (_farmConfig.rewardEndTime ≤ _lastUpdateTime) {
        revert InvalidRewardEndTime(_farmConfig.rewardEndTime, _lastUpdateTime);
    }
}
```

```
// ... rest of the checks  
}
```

Checking `rewardStartTime` is optional as some farms might want to set the `rewardStartTime` after the `lastUpdateTime`.

River: No change, this is the expected state.

Zenith: Acknowledged.

[L-3] OFT dust removal causes permanent 1zCompose failure in FarmManager

SEVERITY: Low

IMPACT: Low

STATUS: Resolved

LIKELIHOOD: Low

Target:

- [FarmManager.sol#L733](#)

Description:

The FarmManager contract's 1zCompose functionality will permanently fail when receiving OFT tokens due to dust removal in the LayerZero OFT implementation.

Proof of Concept:

1. User has 1 ether + 1 wei of reward token on one of the chains.
2. He calls OFT::send function directly and encodes the composeMsg the same way as done in the FarmManager contract.
3. Due to how OFT handles decimal conversion between chains, 1 wei gets [removed from the amount](#).
4. The deposit amount is different than the received amount.
5. The 1zReceive function will credit the reward tokens to the primary farm while 1zCompose will permanently revert.

If the user tries to do the same but through the FarmManager::claimAndStakeCrossChain function the SendParam sets both the amount and minAmount to the same value so if there is dust removal the claimAndStakeCrossChain function will revert, forcing the user to first execute the claim and then initiate the LayerZero message directly through the OFT::send function.

Recommendations:

The FarmManager::claimAndStakeCrossChain function luckily reverts if there is dust removal but it's not obvious that you can't request the claim for an amount that contains dust. Consider adding a check for the amount of the claim request to be a multiple of the [decimalConversionRate](#), i.e. it shouldn't contain any dust.

When it comes to the regular `OFT::send` function make sure that the users or integrators are aware of this behavior if they want to use the `1zCompose` functionality.

Alternatively, you can rely on the `uint256 _amountLD = OFTComposeMsgCodec.amountLD(_message);` and deposit it to the Farm, as this is guaranteed to be the amount transferred to the `FarmManager` contract.

River: Partially resolved with [@67131f28b92...](#).

We've decided to ignore these dust amounts for the following reasons:

- The amounts are extremely small ($> \text{decimalConversionRate } 10^{12} \text{ wei}$).
- Our reward token emission is relatively large, making the dust amounts insignificant in value.
- If we enforce that it must be a multiple of `decimalConversionRate`, then the request claim must also follow the same rule. Otherwise, there will still be issues when calling `stakePendingClaim`.
- On the client side, we will ensure that users input amounts in multiples of `decimalConversionRate`.
- Refunding excess dust amounts to users is not always the expected behavior. Depositing them into the reward farm on the destination chain is immediate, whereas a pure claim would need to wait for the delay time.
- Adding too much extra logic to handle or restrict these dust amounts may not be cost-effective.

Zenith: The resolved part has been verified. The change to use `_amountLD` resolves the issue with permanent `1zCompose` revert, but hardcoding the `minAmountOut` to zero inside the `SendParam` causes dust amounts to be lost for the user.

[L-4] Batch functions may revert if quote changed

SEVERITY: Low

IMPACT: Low

STATUS: Resolved

LIKELIHOOD: Medium

Target:

- [FarmManager.sol:532](#)
- [FarmManager.sol:452](#)

Description:

The function `stakePendingClaimCrossChainBatch()` and `claimAndStakeCrossChainBatch()` enforce the `msg.value` to be equal to the total amount returned by each LayerZero quotes in the batch.

Because quotes can change over time, if the user submitted the batch transaction before the quote changed then the transaction will revert as the total during execution will differ from the `msg.value` provided by the user even if the `msg.value` is greater than the quote.

Recommendations:

Consider modifying the condition inside the `_checkTotalAmount()` function, replace the `if (msgValue \neq totalAmount)` with `if (msgValue < totalAmount)`. Allowing users to send more than the quote total.

River: Resolved with [@d9252726f2...](#)

Zenith: Verified

4.3 Informational

A total of 3 informational findings were identified.

[I-1] LzCompose() may revert because of pause mechanism

SEVERITY: Informational

IMPACT: Informational

STATUS: Acknowledged

LIKELIHOOD: Low

Target:

- [FarmManager.sol](#)

Description:

When the FarmManager contract is paused, all the functions depositing, withdrawing and claiming from a farm are paused.

The LzCompose() function will revert as well if the contract is paused which means the user would not receive his tokens until the contract is unpaused.

Because of this, In the case of an emergency, if side farms are not paused prior to the pausing of the primary farm, some users may still trigger a crosschain staking but will not receive their tokens.

Recommendations:

- Ensure to always pause the farms on other chains prior to pausing the primary farm.
- Consider checking if the primary farm is paused inside LzCompose() and just transfer the reward tokens to the user in that case instead of reverting.

River: No change - Directly transferring to users is not the intended behavior. If the LzCompose operation fails to execute while in a paused state, anyone can still re-trigger the transaction once it is unpaused.

Zenith: Acknowledged

[I-2] Users will not be able to execute their claims when claim period ends

SEVERITY: Informational

IMPACT: Informational

STATUS: Resolved

LIKELIHOOD: Low

Target:

- [Farm.sol](#)

Description:

The function `executeClaim()` can be called after calling the function `requestClaim()` and waiting the `farmConfig.claimDelayTime`.

Both functions check if we currently are inside the claiming period, by calling `_checkIsClaimable()`. While it makes sense to allow claim requests only during the claiming period, it may not be expected for the claim execution.

A user may request his claim during the claim period but because of the `claimDelayTime`, he might be able to claim only after the period ended and thus won't be able to execute his claim.

Recommendations:

Consider removing `_checkIsClaimable()` from the `_beforeExecuteClaim()` internal function so users can execute their claims even after the period ended.

River: Resolved with [@59a2a2ce748...](#). Removed - `_checkIsClaimable()` in `_beforeExecuteClaim()`

Zenith: Verified.

[I-3] Modify `transferCallback()` to call `safeTransfer()` when using self as sender

SEVERITY: Informational	IMPACT: Informational
STATUS: Acknowledged	LIKELIHOOD: Low

Target:

- [FarmManager.sol:795](#)
- [FarmManager.sol:40](#)
- [FarmManager.sol:587](#)

Description:

The FarmManager contract approves itself inside `1zCompose()` and `_stake()` as the callback function `transferCallback()` always uses `transferFrom()` when being called even if the sender is the contract itself which is not ideal.

Recommendations:

Consider adding a condition to the `transferCallback()` function and call `token.safeTransfer()` instead of `token.safeTransferFrom()`. Then remove self approvals throughout the codebase.

River: Acknowledged.