Three Sigma Labs

Code Audit

D1

DistrictOne Social Space with Money Games

Disclaimer

Code Audit

DistrictOne Social Space with Money Games

Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

Table of Contents

Code Audit

DistrictOne Social Space with Money Games

Table of Contents

Summary

Code Audit

DistrictOne Social Space with Money Games

Summary

Three Sigma Labs audited DistrictOne in a 2 days engagement. The audit was conducted from 5-2-2024 to 6-2-2024.

Protocol Description

DistrictOne (D1) merges the excitement of money games with social interaction on Blast L2. It features five engaging activities: Linkup for reward earning and networking, Space Sprint for competitive visibility and earnings, Daily Rally to enhance engagement through gem collection and lotteries, SpaceShare for investing in spaces' success while earning from transactions, and the upcoming Battle Mode for head-to-head competition. D1 offers influencers and projects a dynamic platform for growth without entry barriers, leveraging gamified elements for enhanced community traction and engagement.

Scope

Code Audit

DistrictOne Social Space with Money Games

Scope

SpaceShare

Assumptions

Erc20Utils and ReentrancyGuard are secure.

Methodology

Code Audit

DistrictOne Social Space with Money Games

Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

Taxonomy

In this audit we report our findings using as a guideline Immunefi's vulnerability taxonomy, which can be found at immunefi.com/severity-updated/. The final classification takes into account the severity, according to the previous link, and likelihood of the exploit. The following table summarizes the general expected classification according to severity and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Severity / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Code Audit

DistrictOne Social Space with Money Games

Project Dashboard

Application Summary

Name	DistrictOne
Commit	078c88f9de9f1255753e16fe5bc91bc92b78a2ab
Language	Solidity
Platform	Blast

Engagement Summary

Timeline	5-2-2024 to 6-2-2024
№ of Auditors	2
Review Time	2 days

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	1	1	0
High	2	2	0
Medium	1	1	0
Low	3	2	1

None	1	1	0

Category Breakdown

Suggestion	0
Documentation	0
Bug	7
Optimization	0
Good Code Practices	1

Code Maturity Evaluation

Code Audit

DistrictOne Social Space with Money Games

Code Maturity Evaluation

Code Maturity Evaluation Guidelines

Category	Evaluation
Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system.
Arithmetic	The proper use of mathematical operations and semantics.
Centralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Code Stability	The extent to which the code was altered during the audit.
Upgradeability	The presence of parameterizations of the system that allow modifications after deployment.
Function Composition	The functions are generally small and have clear purposes.
Front-Running	The system's resistance to front-running attacks.
Monitoring	All operations that change the state of the system emit events, making it simple to monitor the state of the system. These events need to be correctly emitted.
Specification	The presence of comprehensive and readable codebase documentation.
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage.

Code Maturity Evaluation Results

Category	Evaluation
Access Controls	Satisfactory. All functions had proper access control.
Arithmetic	Satisfactory. No rounding errors were found.
Centralization	Satisfactory. The owner could increase the fees but users can freely opt out.
Code Stability	Satisfactory. The code was stable throughout the audit.
Upgradeability	Weak. The contracts are not upgradeable.
Function Composition	Satisfactory. Functionality was well split into helpers.
Front-Running	Satisfactory. The code has slippage protection.
Monitoring	Moderate. Some events were missing.
Specification	Satisfactory. The code reflected the specification.
Testing and Verification	Moderate. Some bugs should have been found by tests.

Findings

Code Audit

DistrictOne Social Space with Money Games

Findings

3S-D1-C01

Drained contract if holder fees were non zero and are zero now due to early return

Id	3S-D1-C01
Classification	Critical
Severity	Critical
Likelihood	Medium
Category	Bug
Status	Addressed in 9f1e2b1.

Description

`_updateSharesReward()` does an early return if the **`newRewards`** are **`0`**. **`newRewards`** might be 0, but a user may still increase its shares balance, but due to the early return, its **`holderSharesReward[spaceId][holder].rewardPerSharePaid`** will not be updated. Thus, if a user had pending rewards to claim but the owner set **`holderFeePercent`** to **`0`**, it could increase its balance by calling **`buyShares()`** without updating the pending rewards due to **`0 newRewards`**, then claim rewards, getting much more in return and then selling the shares. Increase the amount bought in the poc to verify how the fees claimed via **`withdrawRewards()`** increase (when they should not as holder fees were set to 0).

Recommendation

_updateSharesReward() should skip the rewardPerShareStored update if **`newReward`** is **`0`**, but it should never skip _updateHolderReward(), as this must be called whenever a user changes its shares balance (before the change).

3S-D1-H01

`withdrawRewards()` does not accumulate the rewards for all the space ids, leading to lost rewards

Id	3S-D1-H01
Classification	High
Severity	High
Likelihood	High
Category	Bug
Status	Addressed in 9f1e2b1.

Description

`withdrawRewards()` is missing the **+** sign in reward = _withdrawRewards(spaceIds[i]), leading to lost rewards.

Recommendation

Replace **=** by **+=**.

3S-D1-H02

Lost rewards due to updating shares in **`_sellShares()`** before calculating rewards

Id	3S-D1-H02
Classification	High
Severity	High
Likelihood	High
Category	Bug
Status	Addressed in 2327513.

Description

The **`SpaceShare`** contract implements MasterChef style rewards by using a rewards accumulator. However, in **`_sellShares()`**, the user shares balance (and the total supply) are updated before calculating the user rewards from the previous period, leading to the user losing these rewards. See the poc here for confirmation.

Recommendation

Update the rewards before changing the user shares and total supply.

```
function _sellShares(uint256 spaceId, uint256 shares, uint256 minOutAmount)
internal returns (uint256 outAmount) {
    ...
    _updateSharesReward(spaceId, holderFee, trader); //@audit place here the
rewards update and remove below
    uint256 totalSupply;
    unchecked {
        sharesBalance[spaceId][trader] -= shares;
        totalSupply = supply - shares;
    }
    sharesSupply[spaceId] = totalSupply;
    ...
}
```

3S-D1-M01

Any signature is valid if the signer is address(0)

Id	3S-D1-M01
Classification	Medium
Severity	High
Likelihood	Low
Category	Bug
Status	Addressed in 9f1e2b1.

Description

`SignatureLib:recoverSigner()` uses **`ecrecover()`** directly which returns **`address(0)`** on an invalid signature. If the signer is not set in the constructor, this would allow anyone to buy shares, as the address condition is true.

Here is a poc.

Recommendation

Use Openzeppelin's wrapper ECDSA, which checks for **`address(0)`** explicitly.

3S-D1-L01

Last user can not exit space

Id	3S-D1-L01
Classification	Low
Severity	Low
Likelihood	High
Category	Bug
Status	Acknowledged

Description

The code requires at least 1 share in each created **`spaceId`**. Thus, the last user can not call exitSpace(), as it will revert trying to delete all the shares. However, it can still manually sell all but 1 share and withdraw rewards after, so there are no funds lost.

3S-D1-L02

Missing events

Id	3S-D1-L02
Classification	Low
Severity	Low
Likelihood	High
Category	Bug
Status	Addressed in 9f1e2b1.

Description

Some state changes are missing events:

https://github.com/OpenLeverageDev/districtone-contracts-3sigma-audit/blob/main/contracts/share/SpaceShare.sol#L59-L65

https://github.com/OpenLeverageDev/districtone-contracts-3sigma-audit/blob/main/contracts/share/SpaceShare.sol#L144

https://github.com/OpenLeverageDev/districtone-contracts-3sigma-audit/blob/main/contracts/share/SpaceShare.sol#L150-L151

https://github.com/OpenLeverageDev/districtone-contracts-3sigma-audit/blob/main/contracts/share/SpaceShare.sol#L157-L158

https://github.com/OpenLeverageDev/districtone-contracts-3sigma-audit/blob/main/contracts/share/SpaceShare.sol#L250

https://github.com/OpenLeverageDev/districtone-contracts-3sigma-audit/blob/main/contracts/share/SpaceShare.sol#L255-L256

3S-D1-L03

Solidity **`0.8.21`** might not be yet supported on Blast due to **`PUSH0`**

Id	3S-D1-L03
Classification	Low
Severity	Low
Likelihood	High
Category	Bug
Status	Addressed in 9f1e2b1.

Description

The protocol is using solidity 0.8.21, which might not be supported on Blast (could not find official confirmation but many L2s don't support it).

Recommendation

Use **`0.8.19`** for safety.

3S-D1-N01

Protocol and holder fees are in basis points not percentages

Id	3S-D1-N01
Classification	None
Category	Good Code Practices
Status	Addressed in 9f1e2b1.

Description

Protocol and holder fees are divided by the numerator FEE_DENOMINATOR which is **`10000`**, so the correct denomination is basis points or **`bps`**.