SMART CONTRACT AUDIT REPORT

for

DistrictOne

Prepared By: Xiaomi Huang

PeckShield
February 5, 2024

Document Properties

Client	DistrictOne
Title	Smart Contract Audit Report
Target	DistrictOne
Version	1.0
Author	Xuxian Jiang
Auditors	Jason Shen, Xuxian Jiang
Reviewed by	Xiaomi Huang
Approved by	Xuxian Jiang
Classification	Final

Version Info

Version	Date	Author(s)	Description
1.0	February 5, 2024	Xuxian Jiang	Final Release
1.0-rc1	February 5, 2024	Xuxian Jiang	Release Candidate #1

Contact

For more information about this document and its contents, please contact PeckShield Inc.

Name	Xiaomi Huang
Phone	+86 183 5897 7782
Email	contact@peckshield.com

Contents

1 Introduction

Given the opportunity to review the design document and related smart contract source code of the `DistrictOne` protocol, we outline in the report our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between smart contract code and design document, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts can be further improved due to the presence of several issues related to either security or performance. This document outlines our audit results.

1.1 About DistrictOne

`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. The basic information of the audited protocol is as follows:

Table 1.1: Basic Information of DistrictOne

Item	Description
Name	DistrictOne
Website	https://districtone.io/
Type	EVM Smart Contract
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	February 5, 2024

In the following, we show the Git repository of reviewed files and the commit hash value used in this audit.

- https://github.com/OpenLeverageDev/districtone-contracts.git (dd5d0fa)

And this is the commit ID after all fixes for the issues found in the audit have been checked in:

- https://github.com/OpenLeverageDev/districtone-contracts.git (9ff0e9e)

1.2 About PeckShield

PeckShield Inc. [9] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystems by offering top-notch, industry-leading services and products (including the service of smart contract auditing). We are reachable at Telegram (https://t.me/peckshield), Twitter (http://twitter.com/peckshield), or Email (contact@peckshield.com).

Table 1.2: Vulnerability Severity Classification

Impact \ **Likelihood**	*High*	*Medium*	*Low*
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

1.3 Methodology

To standardize the evaluation, we define the following terminology based on the OWASP Risk Rating Methodology [8]:

- Likelihood represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- Impact measures the technical loss and business damage of a successful attack;
- Severity demonstrates the overall criticality of the risk.

Table 1.3: The Full Audit Checklist

Category	Checklist Items
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead Of Transfer
	Costly Loop
	(Unsafe) Use Of Untrusted Libraries
	(Unsafe) Use Of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
Semantic Consistency Checks	Semantic Consistency Checks
Advanced DeFi Scrutiny	Business Logics Review
	Functionality Checks
	Authentication Management
	Access Control & Authorization
	Oracle Security
	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	ERC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

Likelihood and impact are categorized into three ratings: *H*, *M* and *L*, i.e., *high*, *medium* and *low* respectively. Severity is determined by likelihood and impact and can be classified into four categories accordingly, i.e., *Critical*, *High*, *Medium*, *Low* shown in Table 1.2.

To evaluate the risk, we go through a checklist of items and each would be labeled with a severity category. For one check item, if our tool or analysis does not identify any issue, the contract is considered safe regarding the check item. For any discovered issue, we might further deploy contracts on our private testnet and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.3.

In particular, we perform the audit according to the following procedure:

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.

- Semantic Consistency Checks: We then manually check the logic of implemented smart contracts and compare with the description in the white paper.

- Advanced DeFi Scrutiny: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [7], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development. Though some categories used in CWE-699 may not be relevant in smart contracts, we use the CWE categories in Table 1.4 to classify our findings. Moreover, in case there is an issue that may affect an active protocol that has been deployed, the public version of this report may omit such issue, but will be amended with full details right after the affected protocol is upgraded with respective fixes.

1.4 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered

Table 1.4: Common Weakness Enumeration (CWE) Classifications Used in This Audit

Category	Summary
Configuration	Weaknesses in this category are typically introduced during the configuration of the software.
Data Processing Issues	Weaknesses in this category are typically found in functionality that processes data.
Numeric Errors	Weaknesses in this category are related to improper calculation or conversion of numbers.
Security Features	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.)
Time and State	Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.
Error Conditions, Return Values, Status Codes	Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function.
Resource Management	Weaknesses in this category are related to improper management of system resources.
Behavioral Issues	Weaknesses in this category are related to unexpected behaviors from code that an application uses.
Business Logic	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application.
Initialization and Cleanup	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
Arguments and Parameters	Weaknesses in this category are related to improper use of arguments or parameters within function calls.
Expression Issues	Weaknesses in this category are related to incorrectly written expressions within code.
Coding Practices	Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an exploitable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained.

comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Last but not least, this security audit should not be used as investment advice.

2 | Findings

2.1 Summary

Here is a summary of our findings after analyzing the implementation of the `DistrictOne` protocol. During the first phase of our audit, we study the smart contract source code and run our in-house static code analyzer through the codebase. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by our tool. We further manually review business logic, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	# of Findings	
Critical	0	
High	0	
Medium	1	■
Low	2	■ ■
Informational	0	
Total	3	

We have so far identified a list of potential issues: some of them involve subtle corner cases that might not be previously thought of, while others refer to unusual interactions among multiple contracts. For each uncovered issue, we have therefore developed test cases for reasoning, reproduction, and/or verification. After further analysis and internal discussion, we determined a few issues of varying severities need to be brought up and paid more attention to, which are categorized in the above table. More information can be found in the next subsection, and the detailed discussions of each of them are in Section 3.

2.2 Key Findings

Overall, these smart contracts are well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 1 medium-severity vulnerability and 2 low-severity vulnerabilities.

Table 2.1: Key DistrictOne Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Medium	Possible Sandwich/MEV Attacks For Reduced Returns	Time and State	Resolved
PVE-002	Low	Improved _updateSharesReward() Logic in SpaceShare	Business Logic	Resolved
PVE-003	Low	Trust Issue of Admin Keys	Security Features	Mitigated

Besides the identified issues, we emphasize that for any user-facing applications and services, it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms should kick in at the very moment when the contracts are being deployed on mainnet. Please refer to Section 3 for details.

3 Detailed Results

3.1 Possible Sandwich/MEV Attacks For Reduced Returns

- ID: PVE-001
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: OPZap
- Category: Time and State [6]
- CWE subcategory: CWE-682 [2]

Description

To facilitate user interactions, the DistrictOne protocol has a convenience contract OPZap that supports token swaps. When examining the token-swapping logic, we notice current implementation may be improved for better slippage control.

```
    function swapETHForOLE() external payable {
        WETH.deposit{value: msg.value}();
        _swapETHForOLE(msg.value, msg.sender);
    }
```

Listing 3.1: OPZap::swapETHForOLE()

```
    function _swapETHForOLE(uint256 ethAmount, address to) internal {
        (uint256 reserve0, uint256 reserve1, ) = OLE_ETH.getReserves();
        IERC20(address(WETH)).transferOut(address(OLE_ETH), ethAmount);
        if (oleIsToken0()) {
            OLE_ETH.swap(getAmountOut(ethAmount, reserve1, reserve0), 0, to, "");
        } else {
            OLE_ETH.swap(0, getAmountOut(ethAmount, reserve0, reserve1), to, "");
        }
    }
```

Listing 3.2: OPZap::_swapETHForOLE()

To elaborate, we show above the related routines. We notice the conversion is routed to the intended UniswapV2 pair in order to swap one asset to another. And the swap operation does not specify an effective restriction on possible slippage and is therefore vulnerable to possible front-running attacks, resulting in a smaller gain for this round of conversion.

Note that this is a common issue plaguing current AMM-based DEX solutions. Specifically, a large trade may be sandwiched by a preceding sell to reduce the market price, and a tailgating buy-back of the same amount plus the trade amount. Such sandwiching behavior unfortunately causes a loss and brings a smaller return as expected to the trading user because the swap rate is lowered by the preceding sell. As a mitigation, we may consider specifying the restriction on possible slippage caused by the trade or referencing the `TWAP` or `time-weighted average price` of `UniswapV2`. Nevertheless, we need to acknowledge that this is largely inherent to current blockchain infrastructure and there is still a need to continue the search efforts for an effective defense.

Recommendation Develop an effective mitigation (e.g., slippage control) to the above front-running attack to better protect the interests of farming users.

Status The issue has been fixed by this commit: `7d18e4f`.

3.2 Improved _updateSharesReward() Logic in SpaceShare

- ID: PVE-002
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `SpaceShare`
- Category: Business Logic [5]
- CWE subcategory: CWE-841 [3]

Description

The audited protocol has a core `SpaceShare` contract that is designed to handle the creation, buying, and selling of shares based on a simple linear pricing model (`P = KS + B`). And a portion of sale proceeds can be allocated as rewards to current share holders. In the process of examining the share reward allocation, we notice the current logic can be improved.

Specifically, we show below the related `_updateSharesReward()` implementation: it computes the increase of `rewardPerShareStored` to account for the new reward addition. However, it makes an early exit when there is no new reward `newReward = 0`. The early exit condition is also applicable when the total share supply is equal to zero, i.e., `sharesSupply[spaceId] == 0`.

```
    function _updateSharesReward(uint256 spaceId, uint256 newReward, address holder) internal {
        if (newReward == 0) {
            return;
        }
        rewardPerShareStored[spaceId] += (newReward * (1 ether)) / sharesSupply[spaceId];
```

```
        _updateHolderReward(spaceId, holder);
    }
```

Listing 3.3: `SpaceShare::_updateSharesReward()`

Recommendation Revise the above function to properly early exit when `sharesSupply[spaceId] == 0`.

Status The issue has been fixed by this commit: `9ff0e9e`.

3.3 Trust Issue of Admin Keys

- ID: PVE-003
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `Multiple Contracts`
- Category: Security Features [4]
- CWE subcategory: CWE-287 [1]

Description

In the audited protocol, there is a privileged `owner` account that plays a critical role in governing and regulating the system-wide operations (e.g., parameter setting and fund withdrawal). It also has the privilege to control or govern the flow of assets managed by this protocol. Our analysis shows that the privileged account needs to be scrutinized. In the following, we examine the privileged account and their related privileged accesses in current contracts.

```
    function setProtocolFeeDestination(address _protocolFeeDestination) external
        override onlyOwner {
        if (_protocolFeeDestination == address(0)) revert ZeroAddress();
        protocolFeeDestination = _protocolFeeDestination;
    }

    function setFees(uint16 _protocolFeePercent, uint16 _holderFeePercent) external
        override onlyOwner {
        // the total fee percent must le 50%
        if (_protocolFeePercent + _holderFeePercent > (FEE_DENOMINATOR / 2)) revert
            InvalidParam();
        protocolFeePercent = _protocolFeePercent;
        holderFeePercent = _holderFeePercent;
    }

    function setSignConf(address _signIssuerAddress, uint256 _signValidDuration)
        external override onlyOwner {
        if (_signIssuerAddress == address(0)) revert ZeroAddress();
        if (_signValidDuration == 0) revert InvalidParam();
        signIssuerAddress = _signIssuerAddress;
```

```
        signValidDuration = _signValidDuration;
    }
```

Listing 3.4: Example Privileged Operations in SpaceShare Contract

In addition, we notice the owner account that is able to add new markets and configure various liquidity. Apparently, if the privileged owner account is a plain EOA account, this may be worrisome and pose counter-party risk to the exchange users. Note that a multi-sig account could greatly alleviate this concern, though it is still far from perfect. Specifically, a better approach is to eliminate the administration key concern by transferring the role to a community-governed DAO. In the meantime, a timelock-based mechanism can also be considered as mitigation.

Recommendation Promptly transfer the privileged account to the intended DAO-like governance contract. All changed to privileged operations may need to be mediated with necessary timelocks. Eventually, activate the normal on-chain community-based governance life-cycle and ensure the intended trustless nature and high-quality distributed governance.

Status This issue has been mitigated with the use of multisig to manage the admin key.

4 Conclusion

In this audit, we have analyzed the design document and related smart contract source code of the DistrictOne protocol, which 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. The current code base is well structured and neatly organized. Those identified issues are promptly confirmed and addressed.

Moreover, we need to emphasize that Solidity-based smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

References

[1] MITRE. CWE-287: Improper Authentication. https://cwe.mitre.org/data/definitions/287.html.

[2] MITRE. CWE-682: Incorrect Calculation. https://cwe.mitre.org/data/definitions/682.html.

[3] MITRE. CWE-841: Improper Enforcement of Behavioral Workflow. https://cwe.mitre.org/data/definitions/841.html.

[4] MITRE. CWE CATEGORY: 7PK - Security Features. https://cwe.mitre.org/data/definitions/254.html.

[5] MITRE. CWE CATEGORY: Business Logic Errors. https://cwe.mitre.org/data/definitions/840.html.

[6] MITRE. CWE CATEGORY: Error Conditions, Return Values, Status Codes. https://cwe.mitre.org/data/definitions/389.html.

[7] MITRE. CWE VIEW: Development Concepts. https://cwe.mitre.org/data/definitions/699.html.

[8] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.

[9] PeckShield. PeckShield Inc. https://www.peckshield.com.