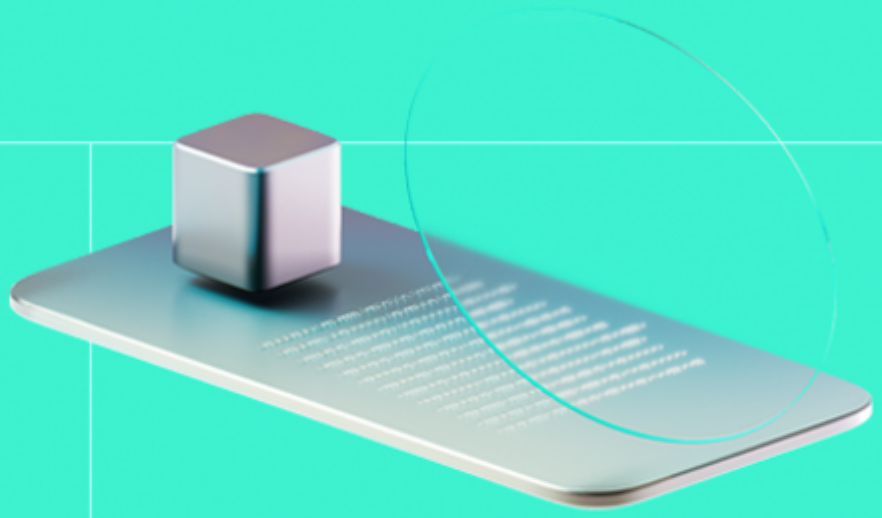
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: BlockSquare

Date: 20/02/2024

We express our gratitude to the BlockSquare team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

The Blocksquare Revenue Distribution System is an innovative smart contract ecosystem, specifically designed for managing and distributing revenue in a property management or real estate investment context.

Platform: EVM

Language: Solidity

Tags: Airdrop; Proxy; Upgradable

Timeline: 24/01/2024 - 20/02/2024

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/blocksquare/blocksquare-contracts
Commit	61b77c4

Audit Summary

10/10	10/10	96.15%	10/10
Security Score	Code quality score	Test coverage	Documentation quality score

Total 9.9/10

The system users should acknowledge all the risks summed up in the risks section of the report

4	4	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	0
Medium	2
Low	2

Vulnerability	Status
F-2024-0570 - Inadequate Constructor Initialization in Upgradeable Contract	Fixed
F-2024-0575 - Checks-Effects-Interactions Pattern Violation	Fixed
F-2024-0579 - Incompatibility with Non-Standard ERC20 Token Interfaces	Fixed
F-2024-0583 - Token Address Alteration in Revenue Distribution Contract	Fixed

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for BlockSquare
Audited By	Ivan Bondar
Approved By	Grzegorz Trawinski
Website	https://blocksquare.io/
Changelog	26/01/2024 - Preliminary Report
	20/02/2024 - Final Report

Table of Contents

System Overview

The Blocksquare Revenue Distribution System is an innovative smart contract ecosystem built on Ethereum, specifically designed for managing and distributing revenue in a property management or real estate investment context. It centralizes the revenue management process for property-related investments, offering a transparent and efficient way to handle revenue distributions and claims.

The files in the scope:

- RevenueDistribution.sol - this contract is central to the system, facilitating various functionalities:
  - Revenue Tracking and Distribution:
    - Manages revenue associated with different properties (PropTokens).
    - Tracks individual and total revenue amounts for properties and users.
    - Enables the addition of revenue for properties and its distribution among multiple users.
  - Revenue Claiming:
    - Provides functions for users to claim revenue for specific properties or multiple properties.
    - Supports claiming for individual and multiple wallets.
  - Data and Token Integration:
    - Interacts with external contracts/interfaces indicated by _data (for permissions) and _tokenAddress (for token transfers).
- RevenueDistributionProxy.sol - a Transparent Upgradeable Proxy that serves as a flexible and upgradable front for the RevenueDistribution contract

Privileged roles

- RevenueDistribution:
  - Contract Owner:
    - Has the authority to transfer ownership and renounce ownership.
    - Can update the data proxy (_data) address, affecting the system's core functionalities.
  - Authorized Users:
    - Users with special permissions (verified through the _data contract) can add revenue to the system.
    - This role is restricted to system administrators or trusted community partners.
- RevenueDistributionProxy:
  - Proxy Admin:
    - Responsible for managing and executing upgrades to the RevenueDistribution contract.
    - Holds significant power over the proxy contract, with the ability to change the business logic address (logic) and initialize new logic contracts.

Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements have some gaps:
  - Project overview is detailed
  - The roles within the system are described.
  - Use cases are described and detailed.
  - For each contract all futures are described
  - All interactions are described.
- Technical description is robust:
  - Run instructions are provided.
  - Technical specification is provided.
  - NatSpec is sufficient.

Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is configured.

Test coverage

Code coverage of the project is **96.15%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage is present.
- Interactions by several users are tested.

Security score

Upon auditing, the code was found to contain **0** critical, **0** high, **2** medium, and **2** low severity issues. All issues were fixed in the remediation part of this audit, leading to a security score of **10** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

Summary

The comprehensive audit of the customer's smart contract yields an overall score of **9.9**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- **Upgradability and Future Version Modifications**:
  - The contract's upgradable nature means that its implementation can be modified in future versions. This flexibility allows for necessary updates and improvements but also introduces uncertainty for users regarding future changes in the contract's behavior and rules. Users should stay informed about any contract upgrades and understand their implications.
- **Criticality of Accurate Revenue Allocation in addRevenue Function**:
  - The `addRevenue` function plays a crucial role in distributing revenues accurately among users. It requires precise input parameters to ensure fair and correct allocation. Incorrect or manipulated inputs can lead to improper revenue distribution, affecting the trust and fairness of the system. Users and community partners must exercise due diligence in verifying the accuracy of the input data.
- **Changeability of Access Control via RevenueDistributionHelpers _data**:
  - The contract allows for the modification of the RevenueDistributionHelpers `_data`, which controls access permissions, particularly for the `addRevenue` function. Changes to this helper can alter who has the authority to distribute revenues, potentially impacting the consistency and integrity of revenue allocations. Users should be aware of any changes to these access controls and understand their potential impact on revenue distribution practices.
- **Solidity Version Compatibility and Cross-Chain Deployment**:
  - The project utilizes Solidity version 0.8.20 or higher, which includes the introduction of the PUSH0 (0×5f) opcode. This opcode is currently supported on the Ethereum mainnet but may not be universally supported across other blockchain networks. Consequently, deploying the contract on chains other than the Ethereum mainnet, such as certain Layer 2 (L2) chains or alternative networks, might lead to compatibility issues or execution errors due to the lack of support for the PUSH0 opcode. In scenarios where deployment on various chains is anticipated, selecting an appropriate Ethereum Virtual Machine (EVM) version that is widely supported across these networks is crucial to avoid potential operational disruptions or deployment failures.

Findings

Vulnerability Details

F-2024-0579 - Incompatibility with Non-Standard ERC20 Token Interfaces - Medium

Description:

The RevenueDistribution contract is designed to manage and distribute revenue among property owners and users. It relies on the RevenueDistributionHelpers interface for critical operations, including permission checks (`canDistributeRent`) and token transfers (`transfer`, `transferFrom`).

The contract uses the RevenueDistributionHelpers interface for ERC20 token transfer operations. However, not all ERC20 tokens adhere to the same standards for return values, particularly for `transfer` and `transferFrom` methods. For example, USDT and some other tokens do not return a boolean value upon successful transfer.

Affected code:

```
function addRevenue(address property, address[] memory users, uint25
6[] memory amount, uint256 from, uint256 to) public {
//...
require(RevenueDistributionHelpers(_tokenAddress).transferFrom(msg.s
ender, address(this), amountToPay));
//...
}

function _claimRevenue(address property, address wallet) private {
//...
RevenueDistributionHelpers(_tokenAddress).transfer(wallet, revenue);
//...
}
```

This deviation could cause **addRevenue** function to fail or behave unexpectedly due to their reliance on the return value for successful execution and limits the contract's flexibility in supporting a broader range of ERC20 tokens without requiring upgrades or redeployment.

Assets:

- contracts/RevenueDistribution.sol [https://github.com/blocksquare/blocksquare-contracts]

Status: Fixed

Classification

Severity: Medium

Impact:

Likelihood [1-5]: 5

Impact [1-5]: 3

Exploitability [1,2]: 2

Complexity [0-2]: 0

Final Score: 2.5 [Medium]

Recommendations

Recommendation:

To ensure compatibility with both standard and non-standard ERC20 tokens:

- Modify the RevenueDistributionHelpers interface to remove `transfer` and `transferFrom` methods. Focus this interface solely on permission-related functionality (`canDistributeRent`).
- Implement OpenZeppelin's `SafeERC20` library for token transfer operations in the contract. `SafeERC20` provides wrapper functions that handle the variability in return values and throw exceptions in case of failed transfers.
- Replace direct calls to `transfer` and `transferFrom` with SafeERC20's safe transfer methods (`safeTransfer`, `safeTransferFrom`) in `addRevenue` and `_claimRevenue` functions.

Remediation (Revised commit: 61b77c4) : The previously identified issue with the RevenueDistribution contract's dependency on the RevenueDistributionHelpers interface for ERC20 token transfers was addressed. The interface no longer includes `transfer` and `transferFrom` methods, ensuring compatibility with a broader range of ERC20 tokens. Additionally, the contract now utilizes OpenZeppelin's SafeERC20 library, replacing direct token transfer calls with `safeTransfer` and `safeTransferFrom` in the `addRevenue` and `_claimRevenue` functions.

F-2024-0583 - Token Address Alteration in Revenue Distribution Contract - Medium

Description:

The RevenueDistribution contract contains functions `addRevenue` and `_claimRevenue` which interact with a token address stored in the contract. The token address is used for transferring funds and calculating revenues.

The inclusion of the `setToken` function in the contract enables the token address to be altered by the contract owner. This flexibility could lead to scenarios where the token address is inadvertently or maliciously changed, potentially to a token of lesser value or an entirely different nature than initially intended.

```
function setToken(address token) public onlyOwner {
_tokenAddress = token;
}
```

Altering the token address could have several unintended consequences:

- Blocked Revenue Claims: If changed to a non-existent or non-functional token, users might be unable to claim their rightfully earned revenues.
- Devaluation of Rewards: Switching to a less valuable token could reduce the actual value of users' earnings.
- System Integrity: An unexpected change in the token address can damage the platform's credibility and trust among users and Community Partners.

Assets:

- contracts/RevenueDistribution.sol [https://github.com/blocksquare/blocksquare-contracts]

Status: Fixed

Classification

Severity: Medium

Impact:

Likelihood [1-5]: 3

Impact [1-5]: 5

Exploitability [1,2]: 2

Complexity [0-2]: 0

Final Score: 2.5 [Medium]

Recommendations

Recommendation: It is recommended to entirely remove the `setToken` functionality from the RevenueDistribution contract. This approach significantly reduces the risk associated with altering the token address post-deployment, thereby maintaining the integrity and expected behavior of the contract.

If the system's design requires the RevenueDistribution contract to work with multiple tokens, a redesign of the contract should be considered. This redesign would enable the contract to handle multiple asset distributions securely and efficiently, rather than relying on a single token address that can be changed.

Remediation (Revised commit: 61b77c4) : The `setToken` function, which allowed the token address to be changed in the RevenueDistribution contract, was removed. This change ensures a fixed token address, eliminating risks associated with altering the token post-deployment and maintaining the integrity and reliability of the revenue distribution system.

F-2024-0570 - Inadequate Constructor Initialization in Upgradeable Contract - Low

Description:

In the RevenueDistribution contract, which is designed to be upgradeable, the constructor is implemented with the `initializer` modifier. However, for upgradeable contracts, the best practice is to use `_disableInitializers` in the constructor to prevent the base contract's constructor from being called more than once in the proxy pattern.

Affected Code:

```
/// @dev Initialize contract params with `initialize` function behin
d a proxy
constructor() initializer {}
```

Neglecting to use `_disableInitializers` in the constructor could potentially lead to initialization issues in upgradeable contracts. While this does not pose an immediate security threat, it deviates from standard best practices and might lead to unexpected behavior, especially when the contract undergoes future upgrades or changes.

Assets:

- contracts/RevenueDistribution.sol [https://github.com/blocksquare/blocksquare-contracts]

Status: Fixed

Classification

Severity: Low

Impact:

Likelihood [1-5]: 3

Impact [1-5]: 2

Exploitability [1,2]: 2

Complexity [0-2]: 0

Final Score: 1.8 [Low]

Recommendations

Recommendation:

To align with best practices for upgradeable contracts, it is recommended to replace the `initializer` modifier in the constructor with a call to `_disableInitializers`. This change ensures the correct initialization

pattern for upgradeable contracts and maintains consistency with widely accepted standards.

Code Snippet:

```
constructor() {
_disableInitializers();
}
```

Remediation (Revised commit: 61b77c4) : The `initializer` modifier was removed, and `_disableInitializers()` is now invoked within the constructor body.

External References:

- Openzeppelin

F-2024-0575 - Checks-Effects-Interactions Pattern Violation - Low

Description:

The RevenueDistribution contract manages revenue allocation and distribution for a property token system. The `_claimRevenue` function, integral to this process, handles the internal logic for transferring claimed revenue to a user's wallet.

In the `_claimRevenue` function, the contract interacts with an external token contract to transfer funds before updating its internal state. This order of operations violates the Checks-Effects-Interactions pattern and exposes the function to reentrancy attacks.

```
function _claimRevenue(address property, address wallet) private {
uint256 revenue = _revenueAmount[property][wallet];
RevenueDistributionHelpers(_tokenAddress).transfer(wallet, revenue);
_revenueAmount[property][wallet] = 0;
_totalRevenue[wallet] = _totalRevenue[wallet] - revenue;
emit RevenueClaimed(property, wallet, revenue, block.timestamp);
}
```

The RevenueDistribution contract is designed for use with stablecoins like DAI, USDC, and USDT, which are typically safe from triggering reentrant calls. However, caution is advised for potential future use with different tokens. Extending the contract's usage to other token types might lead to reentrancy risks, potentially resulting in unintended behavior or vulnerabilities within the contract.

Assets:

- contracts/RevenueDistribution.sol [https://github.com/blocksquare/blocksquare-contracts]

Status: Fixed

Classification

Severity: Low

Impact:

Likelihood [1-5]: 2

Impact [1-5]: 4

Exploitability [1,2]: 2

Complexity [0-2]: 1

Final Score: 2.0 [Low]

Recommendations

Recommendation: Reorder the function to follow the Checks-Effects-Interactions pattern:

1. Validate conditions.
2. Update state variables.
3. Interact with external contracts.

Revised Function:

```
function _claimRevenue(address property, address wallet) private {
uint256 revenue = _revenueAmount[property][wallet];
_revenueAmount[property][wallet] = 0;
_totalRevenue[wallet] -= revenue;
RevenueDistributionHelpers(_tokenAddress).transfer(wallet, revenue);
emit RevenueClaimed(property, wallet, revenue, block.timestamp);
}
```

Remediation (Revised commit: 61b77c4) : The `_claimRevenue` function in the RevenueDistribution contract was restructured. The transfer operation is now executed at the end of the function, after state variables update.

Observation Details

F-2024-0569 - Potential Loss of Ownership Control in Contract Using OwnableUpgradeable - Info

Description:

The RevenueDistribution contract employs OwnableUpgradeable from OpenZeppelin for ownership management.

If ownership is mistakenly transferred, it may result in the irrevocable loss of control over the contract. Any function guarded by the `onlyOwner` modifier would become inaccessible to the original owner, effectively freezing critical administrative functionalities.

Assets:

- contracts/RevenueDistribution.sol [https://github.com/blocksquare/blocksquare-contracts]

Status: Fixed

Recommendations

Recommendation:

Integrate Ownable2StepUpgradeable, which implements a two-step ownership transfer process. This requires the new owner to actively accept ownership, adding an additional layer of security against accidental transfers.

Remediation (Revised commit: 61b77c4) : The RevenueDistribution contract was updated to use Ownable2StepUpgradeable instead of OwnableUpgradeable. This change introduces a two-step ownership transfer process, enhancing security against unintentional ownership changes.

F-2024-0571 - Missing Events in Key Functions - Info

Description:

The RevenueDistribution contract lacks event emissions in two critical owner-only functions: `setDataProxy` and `setToken`. These functions are used to update the addresses of crucial contract dependencies – the data proxy and token contracts, respectively.

The absence of events in these functions means that there is no on-chain traceability or transparency when these addresses are updated.

Assets:

- contracts/RevenueDistribution.sol [https://github.com/blocksquare/blocksquare-contracts]

Status: Fixed

Recommendations

Recommendation:

To enhance transparency and traceability, it is recommended to emit events whenever the data proxy and token contract addresses are updated. This will allow users and external services to monitor and react to changes.

Remediation (Revised commit: 61b77c4) : Introduced the `DataProxyChanged` event within the `setDataProxy` function. Additionally, the `setToken` function was completely removed from the contract.

F-2024-0572 - `event` Declared But Not Emitted - Info

Description: It was identified that `RevenueSent` event is declared but not utilized in any of the solution's functionality. Having unused event declarations consumes additional Gas during the deployment.

```
event RevenueSent(address indexed property, address indexed user, uint256 amount, uint256 time);
```

Assets:

- contracts/RevenueDistribution.sol [https://github.com/blocksquare/blocksquare-contracts]

Status: Fixed

Recommendations

Recommendation: Consider removing the unused event declaration to optimize the contract and enhance clarity. If there is an intent for this event to be part of certain operations, ensure it is emitted appropriately. Otherwise, for the sake of clean and efficient code, it is advisable to remove any unused declarations.

Remediation (Revised commit: 61b77c4) : The `RevenueSent` event, which was previously declared but unused in the contract's operations, was removed.

F-2024-0573 - Function Parameter "owner" Shadows Function owner() from OwnableUpgradeable Contract - Info

Description:

In the RevenueDistribution contract's `initialize` function, the parameter `owner` shadows the `owner()` function inherited from the OwnableUpgradeable contract. Shadowing occurs when a local identifier (variable, parameter, etc.) in a scope has the same name as an identifier in an outer scope, potentially leading to confusion and errors in understanding the code.

This issue does not pose a direct security risk, but it can lead to readability and maintainability issues.

Assets:

- contracts/RevenueDistribution.sol [https://github.com/blocksquare/blocksquare-contracts]

Status: Fixed

Recommendations

Recommendation:

Rename the `owner` parameter in the `initialize` function to a distinct name that does not conflict with any inherited functions or state variables.

Remediation (Revised commit: 61b77c4) : The `initialize` function's parameter previously named `owner` was renamed to `ownerAddress`, resolving the shadowing issue with the inherited `owner()` function from the OwnableUpgradeable contract.

F-2024-0574 - Public Functions That Should Be External - Info

Description: Functions that are meant to be exclusively invoked from external sources should be designated as `external` rather than `public`. This is essential to enhance both the gas efficiency and the overall security of the contract.

Assets:

- contracts/RevenueDistribution.sol [https://github.com/blocksquare/blocksquare-contracts]

Status: Fixed

Recommendations

Recommendation: To optimize gas usage and improve code clarity, declare functions that are not called internally within the contract and are intended for external access as `external` rather than `public`. This ensures that these functions are only callable externally, reducing unnecessary gas consumption and potential security risks.

Affected functions in the **RevenueDistribution** contract:

```
function addRevenue(address property, address[] calldata users, uint
256[] calldata amount, uint256 from, uint256 to) external { ... }
function claimRevenuesForWalletForProperty(address wallet, address p
roperty) external { ... }
function claimRevenuesForWalletForMultipleProperties(address wallet,
address[] calldata properties) external { ... }
function pushRevenueToUser(address property, address[] calldata wall
ets) external { ... }
function setDataProxy(address dataProxy) external onlyOwner { ... }
function setToken(address token) external onlyOwner { ... }
function pendingRevenue(address property, address user) external vie
w returns (uint256) { ... }
function totalPendingRevenue(address user) external view returns (ui
nt256) { ... }
function getAverageMonthlyPayout(address property) external view ret
urns (uint256) { ... }
function totalPayout(address property) external view returns (uint25
6) { ... }
```

Remediation (Revised commit: 61b77c4) : Mentioned functions visibility in the RevenueDistribution contract was updated to `external`.

F-2024-0576 - Missing checks for `address(0)` - Info

Description:

The RevenueDistribution contract lacks essential validations to check for the zero address (`address(0)`) in several functions. The zero address check is a fundamental security measure in smart contracts to prevent operations involving uninitialized or default addresses.

Affected functions:

- The `initialize` function does not validate the `data` and `token` addresses.
- The `addRevenue` function allows adding revenue without validating the user addresses in the `users` array and `property` address, potentially causing revenue allocation to the zero address.
- The `_claimRevenue` function, called by `claimRevenuesForWalletForProperty`, `claimRevenuesForWalletForMultipleProperties`, and `pushRevenueToUser`, can potentially be called for `address(0)`.
- The `setDataProxy` and `setToken` functions allow setting critical contract addresses without zero address checks, risking the misconfiguration of the contract.

Assets:

- contracts/RevenueDistribution.sol [https://github.com/blocksquare/blocksquare-contracts]

Status: Fixed

Recommendations

Recommendation:

Implement zero address validations for all address parameters in the affected functions. Ensure that addresses provided to these functions are always non-zero to maintain the integrity of contract operations.

Remediation (Revised commit: 61b77c4) : Zero address validations was implemented across mentioned functions in the RevenueDistribution contract.

F-2024-0580 - Redundant Zero Address Check in Contract Initialization - Info

Description:

The `initialize` function in the RevenueDistribution contract is designed to set up the contract with initial parameters, such as data proxy, token addresses, and owner information. This function is part of the initialization process for upgradeable contracts.

The `initialize` function includes a redundant check for the zero address on the `owner` parameter. This check is unnecessary as the `__Ownable_init` function from OpenZeppelin's OwnableUpgradeable already includes a similar check to ensure the owner address is not zero.

Implementation of OwnableUpgradeable.sol:

```
function __Ownable_init(address initialOwner) internal onlyInitializ
ing {
__Ownable_init_unchained(initialOwner);
}

function __Ownable_init_unchained(address initialOwner) internal onl
yInitializing {
if (initialOwner == address(0)) {
revert OwnableInvalidOwner(address(0));
}
_transferOwnership(initialOwner);
}
```

While not a security risk, redundant checks can increase the complexity and gas cost of contract deployment and initialization. Simplifying the code by removing unnecessary checks can enhance readability and efficiency.

Assets:

- contracts/RevenueDistribution.sol [https://github.com/blocksquare/blocksquare-contracts]

Status: Fixed

Recommendations

Recommendation:

Remove the redundant zero address check for the `owner` parameter in the `initialize` function. The `__Ownable_init` call sufficiently handles this check, ensuring the provided owner address is not zero.

Remediation (Revised commit: 61b77c4) : The redundant zero address check for the `ownerAddress` parameter in the `initialize` function of the RevenueDistribution contract was removed.

F-2024-0582 - Unnecessary Revenue Transfer for Zero Amount - Info

Description:

The `_claimRevenue` function in the RevenueDistribution contract is used to transfer revenue to a wallet for a specified property. This function is invoked by public functions like `claimRevenuesForWalletForProperty`, `claimRevenuesForWalletForMultipleProperties`, and `pushRevenueToUser`.

Currently, the `_claimRevenue` function proceeds with the revenue transfer process without checking if the revenue amount is greater than zero. This leads to unnecessary calls and computation when the revenue amount is zero, increasing the gas cost and reducing efficiency.

```
function _claimRevenue(address property, address wallet) private {
uint256 revenue = _revenueAmount[property][wallet];
RevenueDistributionHelpers(_tokenAddress).transfer(wallet, revenue);
_revenueAmount[property][wallet] = 0;
_totalRevenue[wallet] = _totalRevenue[wallet] - revenue;
emit RevenueClaimed(property, wallet, revenue, block.timestamp);
}
```

Performing transfers of zero amounts leads to wasteful gas consumption and can clutter event logs. While not a security issue, optimizing these calls can enhance the contract's performance and reduce unnecessary operations.

Status: Fixed

Recommendations

Recommendation:

Implement a check to ensure that revenue transfer is only attempted when the revenue amount is greater than zero.

Remediation (Revised commit: 61b77c4) : In the RevenueDistribution contract, the `_claimRevenue` function was updated to include a condition that ensures token transfer and state updates occur only if the revenue amount is above zero.

F-2024-0591 - SPDX License Identifier - Info

Description: Trust in smart contracts can be better established if their source code is available. Since making source code available always touches on legal problems with regards to copyright, the Solidity compiler encourages the use of machine-readable SPDX license identifiers. Every source file should start with a comment indicating its license.

The RevenueDistributionProxy contract lacks an SPDX License Identifier.

Assets:

- contracts/RevenueDistributionProxy.sol [https://github.com/blocksquare/blocksquare-contracts]

Status: Fixed

Recommendations

Recommendation: Implement SPDX license identifiers at the beginning of the RevenueDistributionProxy contract.

Remediation (Revised commit: 61b77c4) : An SPDX License Identifier was added to the beginning of the RevenueDistributionProxy contract

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details

Repository	https://github.com/blocksquare/blocksquare-contracts
Commit	dbb42e8598c4c03d0879f4c7f03c5f717b613ba0
Whitepaper	N/A
Requirements	NatSpec
Technical Requirements	https://docs.blocksquare.io/for-developers/revenue-distribution-contract; NatSpec

Contracts in Scope

RevenueDistribution.sol
RevenueDistributionProxy.sol