CredShields

Smart Contract Audit

Sept 12th, 2023 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Balance Capital between July 25th, 2023, and Aug 24h, 2023.

Author

Shashank (Co-founder, CredShields)

shashank@CredShields.com

Reviewers

Aditya Dixit (Research Team Lead)

aditya@CredShields.com

Prepared for

Balance Capital

Table of Contents

1. Executive Summary

Balance Capital engaged CredShields to perform a smart contract audit from July 25th, 2023, to Aug 24h, 2023. During this timeframe, Fourteen (14) vulnerabilities were identified.

During the audit, Zero (0) vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Balance Capital" and should be prioritized for remediation, and fortunately, none were found.

The table below shows the in-scope assets and a breakdown of findings by severity per asset. Section 2.3 contains more information on how severity is calculated.

Assets in Scope	Critical	High	Medium	Low	info	Gas	Σ
Smart Contract	0	0	1	5	6	2	**14**
	0	**0**	**1**	**5**	**6**	**2**	**14**

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in Smart Contract's scope during the testing window while abiding by the policies set forth by Smart Contract's team.

State of Security

To maintain a robust security posture, it is essential to continuously review and improve upon current security processes. Utilizing CredShields' continuous audit feature allows both Balance Capital's internal security and development teams to not only identify specific vulnerabilities, but also gain a deeper understanding of the current security threat landscape.

To ensure that vulnerabilities are not introduced when new features are added, or code is refactored, we recommend conducting regular security assessments. Additionally, by analyzing the root cause of resolved vulnerabilities, the internal teams at Balance Capital can implement both manual and automated procedures to eliminate entire classes of vulnerabilities in the future. By taking a proactive approach, Balance Capital can future-proof its security posture and protect its assets.

2. Methodology

Balance Capital engaged CredShields to perform a Balance Capital Smart Contract audit. The following sections cover how the engagement was put together and executed.

2.1 Preparation phase

The CredShields team meticulously reviewed all provided documents and comments in the smart-contract code to gain a thorough understanding of the contract's features and functionalities. They meticulously examined all functions and created a mind map to systematically identify potential security vulnerabilities, prioritizing those that were more critical and business-sensitive for the refactored code. To confirm their findings, the team deployed a self-hosted version of the smart contract and performed verifications and validations during the audit phase.

A testing window from July 25th, 2023, to Aug 24h, 2023, was agreed upon during the preparation phase.

2.1.1 Scope

During the preparation phase, the following scope for the engagement was agreed-upon:

IN SCOPE ASSETS
https://github.com/ryzefi/ryze-contract/tree/main/contracts/binary

Table: List of Files in Scope

2.1.2 Documentation

Documentation was not required as the code was self-sufficient for understanding the project.

2.1.3 Audit Goals

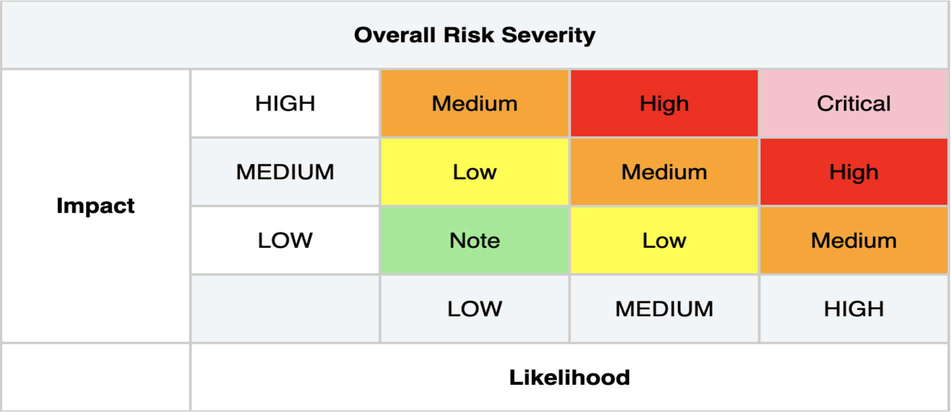
CredShields uses both in-house tools and manual methods for comprehensive smart contract security auditing. The majority of the audit is done by manually reviewing the contract source code, following SWC registry standards, and an extended industry standard self-developed checklist. The team places emphasis on understanding core concepts, preparing test cases, and evaluating business logic for potential vulnerabilities.

2.2 Retesting phase

Balance Capital is actively partnering with CredShields to validate the remediations implemented towards the discovered vulnerabilities.

2.3 Vulnerability classification and severity

CredShields follows OWASP's Risk Rating Methodology to determine the risk associated with discovered vulnerabilities. This approach considers two factors - Likelihood and Impact - which are evaluated with three possible values - **Low**, **Medium**, and **High**, based on factors such as Threat agents, Vulnerability factors, Technical and Business Impacts. The overall severity of the risk is calculated by combining the likelihood and impact estimates.



Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

Overall, the categories can be defined as described below -

1. Informational

We prioritize technical excellence and pay attention to detail in our coding practices. Our guidelines, standards, and best practices help ensure software stability and reliability. Informational vulnerabilities are opportunities for improvement and do not pose a direct risk to the contract. Code maintainers should use their own judgment on whether to address them.

2. **Low**

   Low-risk vulnerabilities are those that either have a small impact or can't be exploited repeatedly or those the client considers insignificant based on their specific business circumstances.

3. **Medium**

   Medium-severity vulnerabilities are those caused by weak or flawed logic in the code and can lead to exfiltration or modification of private user information. These vulnerabilities can harm the client's reputation under certain conditions and should be fixed within a specified timeframe.

4. **High**

   High-severity vulnerabilities pose a significant risk to the Smart Contract and the organization. They can result in the loss of funds for some users, may or may not require specific conditions, and are more complex to exploit. These vulnerabilities can harm the client's reputation and should be fixed immediately.

5. **Critical**

   Critical issues are directly exploitable bugs or security vulnerabilities that do not require specific conditions. They often result in the loss of funds and Ether from Smart Contracts or users and put sensitive user information at risk of compromise or modification. The client's reputation and financial stability will be severely impacted if these issues are not addressed immediately.

6. **Gas**

   To address the risk and volatility of smart contracts and the use of gas as a method of payment, CredShields has introduced a "Gas" severity category. This category deals with optimizing code and refactoring to conserve gas.

2.4 CredShields staff

The following individual at CredShields managed this engagement and produced this report:

- **Shashank, Co-founder CredShields**
    - shashank@CredShields.com

Please feel free to contact this individual with any questions or concerns you have around the engagement or this document.

3. Findings

This chapter contains the results of the security assessment. Findings are sorted by their severity and grouped by the asset and SWC classification. Each asset section will include a summary. The table in the executive summary contains the total number of identified security vulnerabilities per asset per risk indication.

3.1 Findings Overview

3.1.1 Vulnerability Summary

During the security assessment, Fourteen (14) security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC | Vulnerability Type
Improper Array Deletion	Low	Improper Handling of Arrays
Use Ownable2Step	Low	Missing Best Practices
Missing Events in Functions	Low	Missing Best Practices
Name Mapping Parameter in Solidity	Informational	Missing Best Practices
Require Statement Without Messages	Informational	Missing Best Practices
Missing Indexed Keyword in Events	Informational	Missing Best Practices

Multiplication/Division By Two Can Use Bit Shifting	Gas	Gas Optimization
Custom error to save gas	Gas	Gas Optimization
Missing Function To Remove Market	Low	Missing Best Practices
Removal of Unused Import	Informational	Missing Best Practices
Wrong Condition To Add and Remove Admin	Medium	Missing Best Practices
Lack Of Initializer Modifier On Initialize Function	Low	Missing Best Practices
Type Casting From uint16 to uint256	Informational	Missing Best Practices
Unnecessary Function	Informational	Dead Code

Table: Findings in Smart Contracts

3.1.2 Findings Summary

SWC ID	SWC Checklist	Test Result	Notes
SWC-100	Function Default Visibility	Not Vulnerable	Not applicable after v0.5.X (Currently using solidity v >= 0.8.6)
SWC-101	Integer Overflow and Underflow	Not Vulnerable	The issue persists in versions before v0.8.X.
SWC-102	Outdated Compiler Version	Not Vulnerable	Version 0^.8.0 and above is used
SWC-103	Floating Pragma	Not Vulnerable	Contract uses floating pragma
SWC-104	Unchecked Call Return Value	Not Vulnerable	call() is not used
SWC-105	Unprotected Ether Withdrawal	Not Vulnerable	Appropriate function modifiers and require validations are used on sensitive functions that allow token or ether withdrawal.
SWC-106	Unprotected SELFDESTRUCT Instruction	Not Vulnerable	selfdestruct() is not used anywhere
SWC-107	Reentrancy	Not Vulnerable	No notable functions were vulnerable to it.
SWC-108	State Variable Default Visibility	Not Vulnerable	Not Vulnerable
SWC-109	Uninitialized Storage Pointer	Not Vulnerable	Not vulnerable after compiler version, v0.5.0

SWC-110	Assert Violation	Not Vulnerable	Asserts are not in use.
SWC-111	Use of Deprecated Solidity Functions	Not Vulnerable	None of the deprecated functions like block.blockhash(), msg.gas, throw, sha3(), callcode(), suicide() are in use
SWC-112	Delegatecall to Untrusted Callee	Not Vulnerable	Not Vulnerable.
SWC-113	DoS with Failed Call	Not Vulnerable	No such function was found.
SWC-114	Transaction Order Dependence	Not Vulnerable	Not Vulnerable.
SWC-115	Authorization through tx.origin	Not Vulnerable	tx.origin is not used anywhere in the code
SWC-116	Block values as a proxy for time	Not Vulnerable	Block.timestamp is not used
SWC-117	Signature Malleability	Not Vulnerable	Not used anywhere
SWC-118	Incorrect Constructor Name	Not Vulnerable	All the constructors are created using the constructor keyword rather than functions.
SWC-119	Shadowing State Variables	Not Vulnerable	Not applicable as this won’t work during compile time after version 0.6.0
SWC-120	Weak Sources of Randomness from Chain Attributes	Not Vulnerable	Random generators are not used.
SWC-121	Missing Protection against Signature Replay Attacks	Not Vulnerable	No such scenario was found

SWC-122	Lack of Proper Signature Verification	Not Vulnerable	Not used anywhere
SWC-123	Requirement Violation	Not Vulnerable	Not vulnerable
SWC-124	Write to Arbitrary Storage Location	Not Vulnerable	No such scenario was found
SWC-125	Incorrect Inheritance Order	Not Vulnerable	No such scenario was found
SWC-126	Insufficient Gas Griefing	Not Vulnerable	No such scenario was found
SWC-127	Arbitrary Jump with Function Type Variable	Not Vulnerable	Jump is not used.
SWC-128	DoS With Block Gas Limit	Not Vulnerable	Not Vulnerable.
SWC-129	Typographical Error	Not Vulnerable	No such scenario was found
SWC-130	Right-To-Left-Override control character (U+202E)	Not Vulnerable	No such scenario was found
SWC-131	Presence of unused variables	Not Vulnerable	No such scenario was found
SWC-132	Unexpected Ether balance	Not Vulnerable	No such scenario was found
SWC-133	Hash Collisions With Multiple Variable Length Arguments	Not Vulnerable	abi.encodePacked() or other functions are not used.
SWC-134	Message call with hardcoded gas amount	Not Vulnerable	Not used anywhere in the code
SWC-135	Code With No Effects	Not Vulnerable	No such scenario was found
SWC-136	Unencrypted Private Data On-Chain	Not Vulnerable	No such scenario was found

4. Remediation Status

Balance Capital is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on <retest_date>, and all the issues have been addressed.**

Also, the table shows the remediation status of each finding.

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Improper Array Deletion	Low	**Pending Fix**
Use Ownable2Step	Low	**Pending Fix**
Missing Events in Functions	Low	**Pending Fix**
Name Mapping Parameter in Solidity	Informational	**Pending Fix**
Require Statement Without Messages	Informational	**Pending Fix**
Missing Indexed Keyword in Events	Informational	**Pending Fix**
Multiplication/Division By Two Can Use Bit Shifting	Gas	**Pending Fix**
Custom error to save gas	Gas	**Pending Fix**
Missing Function To Remove Market	Low	**Pending Fix**
Removal of Unused Import	Informational	**Pending Fix**
Wrong Condition To Add and Remove Admin	Medium	**Pending Fix**
Lack Of Initializer Modifier On Initialize Function	Low	**Pending Fix**
Type Casting From uint16 to uint256	Informational	**Pending Fix**

Unnecessary Function	Informational	**Pending Fix**

Table: Summary of findings and status of remediation

5. Bug Reports

Bug ID #1

Improper Array Deletion

Vulnerability Type
Improper Handling of Arrays

Severity
Low

Description
Elements inside Arrays in solidity can be deleted using delete or .length = 0 directives. However, this will NOT shift the elements in your array and will leave an element of string 0 in your array.

When a new element is added to the same array, the size of the array keeps on increasing if the length is not adjusted for the deleted element. This creates gigantic arrays and may lead to an Out-of-Gas exception.

Affected Code

- https://github.com/Balance-Capital/balance-contract/blob/af737eea8e695d8ec4db80d08411c79ba0b93671/contracts/binary/BinaryMarket.sol#L1026 - BinaryMarket.sol#setTimeframes();

Impacts
Due to the size of the array not being adjusted after deletion, adding new elements to the array becomes increasingly expensive in terms of gas consumption. This can lead to transactions running out of gas, resulting in failed or reverted transactions.

Remediation

Restrict adding too many elements into the storage array. Otherwise, allow partial deletion of the array's elements. Instead of using delete, and .length=0, use push and pop functions to interact with array elements.pop can be used to remove an element from the end of the array. This also implicitly calls delete on the removed element.

Retest:

Bug ID #2

Use Ownable2Step

Vulnerability Type

Missing Best Practices

Severity

Low

Description

The "Ownable2Step" pattern is an improvement over the traditional "Ownable" pattern, designed to enhance the security of ownership transfer functionality in a smart contract. Unlike the original "Ownable" pattern, where ownership can be transferred directly to a specified address, the "Ownable2Step" pattern introduces an additional step in the ownership transfer process. Ownership transfer only completes when the proposed new owner explicitly accepts the ownership, mitigating the risk of accidental or unintended ownership transfers to mistyped addresses.

Affected Code

- https://github.com/Balance-Capital/balance-contract/blob/af737eea8e695d8ec4db80d08411c79ba0b93671/contracts/binary/BinaryMarketManager.sol#L11 - BinaryMarketManager.sol
- https://github.com/Balance-Capital/balance-contract/blob/af737eea8e695d8ec4db80d08411c79ba0b93671/contracts/binary/BinaryVaultManager.sol#L10 - BinaryVaultManager.sol
- https://github.com/Balance-Capital/balance-contract/blob/af737eea8e695d8ec4db80d08411c79ba0b93671/contracts/binary/BinaryConfig.sol#L11 - BinaryConfig.sol
- https://github.com/Balance-Capital/balance-contract/blob/af737eea8e695d8ec4db80d08411c79ba0b93671/contracts/binary/BinaryVault.sol#L31 - BinaryVault.sol

Impacts

Without the "Ownable2Step" pattern, the contract owner might inadvertently transfer ownership to an unintended or mistyped address, potentially leading to a loss of control over the contract. By adopting the "Ownable2Step" pattern, the smart contract becomes more resilient against external attacks aimed at seizing ownership or manipulating the contract's behavior.

Remediation

It is recommended to use either Ownable2Step or Ownable2StepUpgradeable depending on the smart contract.

Retest:

Bug ID #3

Missing Events in Functions

Vulnerability Type

Missing Best Practices

Severity

Low

Description

Events are inheritable members of contracts. When you call them, they cause the arguments to be stored in the transaction's log—a special data structure in the blockchain. These logs are associated with the address of the contract which can then be used by developers and auditors to keep track of the transactions. The contract was found to be missing these events on certain critical functions which would make it difficult or impossible to track these transactions off-chain.

Affected Code

- https://github.com/Balance-Capital/balance-contract/blob/af737eea8e695d8ec4db80d08411c79ba0b93671/contracts/binary/BinaryConfig.sol#L54-L102
- https://github.com/Balance-Capital/balance-contract/blob/af737eea8e695d8ec4db80d08411c79ba0b93671/contracts/binary/BinaryConfig.sol#L118-L130

Impacts

Events are used to track the transactions off-chain, and missing these events on critical functions makes it difficult to audit these logs if they're needed at a later stage.

Remediation

Consider emitting events for the functions mentioned above. It is also recommended to have the addresses indexed.

Retest:

Bug ID #4

Name Mapping Parameter in Solidity

Vulnerability Type

Missing Best Practices

Severity

Informational

Description

In Solidity version 0.8.18 and later, a new feature was introduced to name mapping parameters. This enhancement allows developers to provide meaningful names for mappings, making the code more self-descriptive and easier to understand. Before this update, mappings were anonymous and required separate documentation or comments to explain their purpose. By naming mapping parameters, Solidity aims to improve code readability, maintainability, and overall developer experience.

Affected Code

- https://github.com/Balance-Capital/balance-contract/blob/af737eea8e695d8ec4db80d08411c79ba0b93671/contracts/binary/BinaryVaultManager.sol#L12
- https://github.com/Balance-Capital/balance-contract/blob/af737eea8e695d8ec4db80d08411c79ba0b93671/contracts/binary/BinaryConfig.sol#L30
- https://github.com/Balance-Capital/balance-contract/blob/af737eea8e695d8ec4db80d08411c79ba0b93671/contracts/binary/BinaryMarket.sol#L64-L77
- https://github.com/Balance-Capital/balance-contract/blob/af737eea8e695d8ec4db80d08411c79ba0b93671/contracts/binary/BinaryMarket.sol#L90

Impacts

Named mapping parameters provide clear and concise descriptions of their purposes, reducing the need for additional comments or documentation to understand the role of each mapping in the contract. The named mappings enable developers to write more self-explanatory code, making it easier for other team members and auditors to understand the intention behind each mapping and its usage in the contract.

Remediation

It is recommended to name the mapping parameters if Solidity 0.8.18 and above is used.

Retest:

Bug ID #5

Require Stemenent Without Messages

Vulnerability Type

Missing Best Practices

Severity

Informational

Description

During code analysis, a "require" statement was found in the smart contract without an accompanying error message. In Solidity, the "require" statement is used to enforce certain conditions, and it takes two parameters: a boolean expression and an optional error message. The error message provides essential information about the condition that must be satisfied and is shown to the user when the require statement evaluates to false. However, in this case, the "require" statement does not have an error message, potentially leading to ambiguity when the condition is not met.

Affected Code

- Multiple instances (advised to check all require statements and improve them wherever possible)

Impacts

Without a clear error message, users and developers will have difficulty understanding why the "require" statement failed and what condition was not met, making it challenging to troubleshoot and fix issues. When the "require" statement fails, it is crucial to provide informative error messages to users to help them understand why their transaction failed or was reverted.

Remediation

It is recommended to add a descriptive message, no longer than 32 bytes, inside the require statement to give more detail to the user about why the condition failed.

Retest:

Bug ID #6

Missing Indexed Keyword in Events

Vulnerability Type

Missing Best Practices

Severity

Informational

Description

During code analysis, it was observed that some events in the smart contract are missing the "indexed" keyword for their parameters. In Solidity, events can have parameters that are indexed, which allows for more efficient filtering and retrieval of specific data from the emitted events. When an event parameter is marked as indexed, it becomes part of the event's topic, enabling clients to filter and search for events based on those indexed parameters.

Affected Code

- https://github.com/Balance-Capital/balance-contract/blob/af737eea8e695d8ec4db80d08411c79ba0b93671/contracts/binary/BinaryMarket.sol#L138

Impacts

Without using indexed parameters, filtering events becomes less efficient, as clients will have to scan through all the event logs to find the relevant data, potentially leading to increased resource consumption and slower response times.

Remediation

Consider adding indexed keyword to crucial event parameters that could be used in off-chain tracking. Do remember that the indexed keyword costs more gas.

Retest:

Bug ID #7

Multiplication/Division By Two Can Use Bit Shifting

Vulnerability Type

Gas Optimization

Severity

Gas

Description

During code analysis, it was identified that the smart contract is using multiplication and division by two (x * 2 and x / 2) through regular arithmetic operations (MUL and DIV opcodes), instead of utilizing the more gas-efficient bit shifting operations (SHL and SHR). In Solidity, x * 2 is equivalent to x << 1 (bit shift left), and x / 2 is equivalent to x >> 1 (bit shift right).

Affected Code

- https://github.com/Balance-Capital/balance-contract/blob/af737eea8e695d8ec4db80d08411c79ba0b93671/contracts/binary/BinaryMarket.sol#L413
- https://github.com/Balance-Capital/balance-contract/blob/af737eea8e695d8ec4db80d08411c79ba0b93671/contracts/binary/BinaryMarket.sol#L553

Impacts

MUL and DIV opcodes cost 5 gas each, whereas SHL and SHR opcodes only cost 3 gas. The inefficient use of gas can lead to higher transaction costs for contract users and limit the scalability of the contract.

Remediation

It is recommended to use left and right shifts instead of multiplying and dividing by 2 to save some gas.

Retest:

Bug ID #8

Custom error to save gas

Vulnerability Type

Gas Optimization

Severity

Gas

Description

During code analysis, it was observed that the smart contract is using the revert() statements for error handling. However, since Solidity version 0.8.4, custom errors have been introduced, providing a better alternative to the traditional revert(). Custom errors allow developers to pass dynamic data along with the revert, making error handling more informative and efficient. Furthermore, using custom errors can result in lower gas costs compared to the revert() statements.

Affected Code

- https://github.com/Balance-Capital/balance-contract/blob/af737eea8e695d8ec4db80d08411c79ba0b93671/contracts/binary/BinaryVault.sol#L162
- https://github.com/Balance-Capital/balance-contract/blob/af737eea8e695d8ec4db80d08411c79ba0b93671/contracts/binary/BinaryMarketManager.sol#L69
- https://github.com/Balance-Capital/balance-contract/blob/af737eea8e695d8ec4db80d08411c79ba0b93671/contracts/binary/BinaryMarket.sol#L706

Impacts

Custom errors allow developers to provide more descriptive error messages with dynamic data. This provides better insights into the cause of the error, making it easier for users and developers to understand and address issues.

Remediation

It is recommended to replace all the instances of revert() statements with error() to save gas.

Retest:

Bug ID #9

Missing Function To Remove Market

Vulnerability Type

Missing Best Practices

Severity

Low

Description

During code analysis, it was observed that the smart contract contains a function named registerMarket() that is used to add new markets. However, there is no corresponding function or mechanism provided to remove markets once they have been added. This absence of a function to remove markets can have implications on the contract's functionality and may lead to potential issues in the long run.

Affected Code

- https://github.com/Balance-Capital/balance-contract/blob/af737eea8e695d8ec4db80d08411c79ba0b93671/contracts/binary/BinaryMarketManager.sol#L27-L33 - BinaryMarketManager.sol#registerMarket()

Impacts

Once a market is registered using the registerMarket() function, it becomes a permanent part of the contract. This can lead to an accumulation of inactive or obsolete markets, potentially affecting contract efficiency and storage costs. If any exploit happens in any market then there is no function to control the situation by removing the market.

Remediation

Add a new function to the contract that allows authorized users or administrators to remove markets that are no longer required or have become inactive. Ensure that proper access control mechanisms are in place to prevent unauthorized removal of markets

Retest

Bug ID #10

Removal of Unused Import

Vulnerability Type

Missing Best Practices

Severity

Informational

Description

During code analysis, it was observed that the smart contract includes an unused import statement for the library "SafeERC20." However, this library is not utilized within the contract, which indicates that the import is unnecessary.

Affected Code

- https://github.com/Balance-Capital/balance-contract/blob/af737eea8e695d8ec4db80d08411c79ba0b93671/contracts/binary/BinaryVaultManager.sol#L6 - BinaryVaultManager.sol# Import SafeERC20.sol

Impacts

While an unused import itself may not add significant overhead, when combined with other unused code or imports, it can contribute to an unnecessarily larger contract size. This may marginally increase the gas cost for deployment and execution.

Remediation

Remove the unused import statement for the "SafeERC20" library from the contract. This will eliminate unnecessary code and improve contract readability

Retest:

Bug ID #11

Wrong Condition To Add and Remove Admin

Vulnerability Type

Missing Best Practices

Severity

Medium

Description

During code analysis, it was identified that the setAdmin() function in the smart contract is used to add and remove admin roles. However, there were incorrect conditions within the require statements that check whether the given address already has or does not have the admin role. After careful review and correction, the conditions have been updated to reflect the correct logic. The corrected setAdmin() function ensures that the admin roles are managed accurately, and potential issues related to role assignments have been addressed. While setting admin inside require statement there is a wrong condition also while removing admin there is a wrong condition inside require statement.

Affected Code

- https://github.com/Balance-Capital/balance-contract/blob/af737eea8e695d8ec4db80d08411c79ba0b93671/contracts/binary/BinaryMarket.sol#L199-L210 - BinaryMarket.sol#setAdmin()

Impacts

After deploying the contract once the admin is set the mechanism to add and remove admin will always fail due to wrong conditions inside a require statement.

Remediation

Update the conditions inside the setAdmin() function to the below condition

```
   function setAdmin(address admin_, bool enable) external onlyAdmin {
        require(admin_ != address(0), "ZERO_ADDRESS");
        emit AdminChanged(admin_, enable);

        if (enable) {
                require(!hasRole(DEFAULT_ADMIN_ROLE, admin_), "Already enabled."); //
@audit report - medium - logic is incorrect
            grantRole(DEFAULT_ADMIN_ROLE, admin_);
        } else {
            require(hasRole(DEFAULT_ADMIN_ROLE, admin_), "Already disabled.");
            revokeRole(DEFAULT_ADMIN_ROLE, admin_);
        }
    }
```

Retest:

Bug ID #12

Lack Of Initializer Modifier On Initialize Function

Vulnerability Type

Missing Best Practices

Severity

Low

Description

Upon reviewing the code, it has been observed that the smart contract contains a series of initialize() functions that allow the admin to initialize the contract. However, there is currently no mechanism in place to prevent the initialize() function from being executed multiple times, which could lead to undesirable consequences. Ideally, the initialize() function should only be permitted to run once after the contract's deployment.

Affected Code

- https://github.com/Balance-Capital/balance-contract/blob/af737eea8e695d8ec4db80d08411c79ba0b93671/contracts/binary/vault/BinaryVaultFacet.sol#L132-L144 - BinaryVaultFacet.sol#initialize()
- https://github.com/Balance-Capital/balance-contract/blob/af737eea8e695d8ec4db80d08411c79ba0b93671/contracts/binary/vault/BinaryVaultNFTFacet.sol#L61-L67 - BinaryVaultNFTFacet.sol#initialize

Impacts

Allowing the initialize() function to be invoked multiple times can lead to redundant initialization processes. This could result in inconsistencies within the contract's state and behavior, which may not align with the intended design

Remediation

Implement an initializer modifier that examines whether the contract has already undergone initialization. This modifier should only permit the initialize() function to run if the contract has not been initialized yet.

Bug ID #13

Type Casting From uint16 to uint256

Vulnerability Type

Missing Best Practices

Severity

Informational

Description

Upon reviewing the code, it has been identified that the constructor takes an input parameter tradingFee_ of type uint16 and assigns it to the variable tradingFee, which is declared as uint256. This type casting from a smaller data type (uint16) to a larger data type (uint256) may result in unexpected behavior and potential data loss.

Affected Code

- https://github.com/Balance-Capital/balance-contract/blob/af737eea8e695d8ec4db80d08411c79ba0b93671/contracts/binary/BinaryConfig.sol#L40-L52 - BinaryConfig.sol#constructor

Impacts

When casting from a smaller data type (uint16) to a larger one (uint256), any data that exceeds the range of the smaller type might be lost or truncated. This can lead to inaccurate results and unintended contract behaviour. Type casting between different data types often requires additional gas for conversion operations. Using a larger data type than necessary can lead to increased gas consumption during contract execution.

Remediation

Change type of tradingFee_ constructor’s input variable to uint256

Bug ID #14

Unnecessary Function

Vulnerability Type

Dead Code

Severity

Informational

Description

The contract Oracle defines a function isWritable() which always returns true. This function is then called inside BinaryMarket._writeOraclePrice() to check whether the oracle.isWritable() returns true or false.

Due to the current implementation, it will always return true rendering the code and validation useless.

Affected Code

- https://github.com/Balance-Capital/balance-contract/blob/af737eea8e695d8ec4db80d08411c79ba0b93671/contracts/binary/Oracle.sol#L177-L179 - Oracle.isWritable()

Impacts

The current code implementation of the isWritable() function renders it useless as it will always return true so the validations will always pass.

Remediation

It is recommended to implement the function isWritable() in such a way that the owners are able to handle its return value. If it is not needed then delete the function and remove the “if” validation.

6. Disclosure

The Reports provided by CredShields is not an endorsement or condemnation of any specific project or team and do not guarantee the security of any specific project. The contents of this report are not intended to be used to make decisions about buying or selling tokens, products, services, or any other assets and should not be interpreted as such.

Emerging technologies such as Smart Contracts and Solidity carry a high level of technical risk and uncertainty. CredShields does not provide any warranty or representation about the quality of code, the business model or the proprietors of any such business model, or the legal compliance of any business. The report is not intended to be used as investment advice and should not be relied upon as such.

CredShields Audit team is not responsible for any decisions or actions taken by any third party based on the report.