CredShields

Smart Contract Audit

19th of July, 2024 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Kresus between 12/06/2024, and 21/06/2024. A retest was performed on 08/07/2024.

Author

Shashank (Co-founder, CredShields)

shashank@CredShields.com

Reviewers

Aditya Dixit (Research Team Lead)

Shreyas Koli (Auditor)

Naman Jain (Auditor)

Prepared for

Kresus

Table of Contents

1. Executive Summary

Kresus engaged CredShields to perform a smart contract audit from 12/06/2024 to 21/06/2024. During this timeframe, 12 vulnerabilities were identified. **A retest was performed on 08/07/2024, and all the bugs have been addressed.**

During the audit, 2 vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Kresus" 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
Kresus Smart Contracts	1	1	1	3	0	6	**12**
	1	**1**	**1**	**3**	**0**	**6**	**12**

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in the Kresus Smart Contract”s scope during the testing window while abiding by the policies set forth by the Kresus 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 Kresus'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 Kresus can implement both manual and automated procedures to eliminate entire classes of vulnerabilities in the future. By taking a proactive approach, Kresus can future-proof its security posture and protect its assets.

2. Methodology

Kresus engaged CredShields to perform a Kresus 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 12/06/2024 to 21/06/2024 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
Kresus Smart Contracts - https://github.com/kresuslabs/vault-contracts/tree/3d414145d54ea3b9a80ced802d0403a7d03114a4

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

Kresus 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, and 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 about 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, 12 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC | Vulnerability Type
Ownership Transfer to Zero Address	Critical	Lack of Address Validation
Cross-Chain Signature Replay Attack	High	Cross-Chain Signature Replay
Storage Layout Conflict in KresusVault	Medium	Storage Layout Conflict
Use of Multiple Pragma Versions	Low	Missing Best Practices
Missing Events in Important Functions	Low	Missing Best Practices
Floating and Outdated Pragma	Low	Floating Pragma (SWC-103)
Boolean Equality	Gas	Gas Optimization

Splitting Require/Revert Statements	Gas	Gas Optimization
State Variable Can Be Marked As Constants	Gas	Gas Optimization
Cheaper Inequalities in if()	Gas	Gas Optimization
Gas Optimization in Increments	Gas	Gas Optimization
Unused Imports	Gas	Gas Optimization

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	Vulnerable	Versions ^0.8.23, and ^0.8.20 are used
SWC-103	Floating Pragma	Vulnerable	The contract uses floating pragma
SWC-104	Unchecked Call Return Value	Not Vulnerable	This is not vulnerable
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	This is not vulnerable
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	Vulnerable	Bug ID #3
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	Vulnerable	Bug ID #12
SWC-136	Unencrypted Private Data On-Chain	Not Vulnerable	No such scenario was found

4. Remediation Status

Kresus is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on 08/07/2024, and all the issues have been addressed.**

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

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Ownership Transfer to Zero Address	Critical	**Fixed**
Cross-Chain Signature Replay Attack	High	**Fixed**
Storage Layout Conflict in KresusVault	Medium	**Fixed**
Use of Multiple Pragma Versions	Low	**Fixed**
Missing Events in Important Functions	Low	**Partially Fixed**
Floating and Outdated Pragma	Low	**Fixed**
Boolean Equality	Gas	**Fixed**
Splitting Require/Revert Statements	Gas	**Won't Fix**
State Variable Can Be Marked As Constants	Gas	**Won't Fix**
Cheaper Inequalities in if()	Gas	**Won't Fix**
Gas Optimization in Increments	Gas	**Fixed**
Unused Imports	Gas	**Fixed**

Table: Summary of findings and status of remediation

5. Bug Reports

Bug ID #1 [Fixed]

Ownership Transfer to Zero Address

Vulnerability Type
Lack of Address Validation

Severity
Critical

Description
The `transferOwnershipTrustee()` function is designed to transfer ownership to a trustee. However, after validating the signatures, it sets the `_TRUSTEE` to the zero address and transfers the ownership to this zero address. This design flaw leads to the ownership being permanently transferred to an inaccessible address, effectively making the contract ownerless. Additionally, there is no validation to ensure the trustee address is not a zero address before transferring ownership, compounding the risk of permanent loss.

Affected Code

- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/KresusVault.sol#L160-L161

Impacts
Once the ownership is transferred to the zero address, the contract becomes ownerless, and no further administrative actions can be taken, including managing funds or executing privileged functions. If the contract manages funds, these funds could become permanently locked, rendering them inaccessible and causing financial loss.

Remediation

To remediate this issue, it is recommended to transfer the ownership to the trustee before resetting the trustee address.

Retest

This issue has been addressed by transferring the ownership to the trustee before resetting its address.

Bug ID #2 [Fixed]

Cross-Chain Signature Replay Attack

Vulnerability Type

Cross-Chain Signature Replay

Severity

High

Description

The transferOwnershipTrustee() function in the contract appears to be vulnerable to a cross-chain signature replay attack. This type of attack occurs when a signature from one chain is used on another chain, effectively replaying the action in a different context. In this function, a signature is used to validate the request, but there is no differentiation between chains, allowing attackers to potentially use a valid signature from one chain on another.

Affected Code

- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/KresusVault.sol#L130-L163

Impacts

If this vulnerability is exploited, it could lead to unintended transfers of ownership. An attacker could replay a legitimate request signature from one chain on another chain, causing assets to be transferred to the recipient unintentionally. This could result in financial losses and unexpected behavior in the contract.

Remediation

Add logic to ensure that the request and signature are valid only within the intended chain. This can be achieved by including the chain's identifier or network ID in the data that is signed. When verifying the signature, check that the chain ID matches the expected value.

Retest

This issue has been fixed by adding a chain ID in the hash

Bug ID #3 [Fixed]

Storage Layout Conflict in KresusVault

Vulnerability Type
Storage Layout Conflict

Severity
Medium

Description
The KresusVault contract is an upgradeable contract using the UUPS pattern and inherits from the BaseLightAccount contract. The KresusVault contract defines storage variables but lacks a storage gap. The BaseLightAccount contract, which KresusVault inherits, defines an enum variable SignatureType. Due to the absence of a storage gap in the KresusVault contract, adding new storage variables can potentially overwrite the storage layout of the inherited BaseLightAccount contract, leading to critical misbehaviors.

Affected Code

- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/KresusVault.sol#L51-L65
- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/common/BaseLightAccount.sol#L19-L23

Impacts
Without a storage gap, new storage variables introduced in the KresusVault contract can overwrite the beginning of the storage layout, causing unexpected behavior and potentially severe vulnerabilities.

Remediation
Introduce a storage gap in the KresusVault contract to reserve space for future storage variables without affecting the inherited contract's storage layout. Or you can use namespace variables.

Retest

This issue has been addressed by adding a storage gap in KresusVault.sol

Bug ID#4 [Fixed]

Use of Multiple Pragma Versions

Vulnerability Type
Missing Best Practices

Severity
Low

Description
The contracts were found to be using multiple Solidity Compiler versions across different solidity files. This is not a good coding practice because different versions of the compiler have different caveats, breaking changes and introducing vulnerabilities.

Affected Code

- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/Interfaces/IAccessControl.sol#L2
- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/Interfaces/IConstants.sol#L2
- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/helpers/MultisigAuth.sol#L2
- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/access/AccessControl.sol#L3

Impacts
Having different pragma versions across multiple contracts increases the chances of introducing vulnerabilities since each solidity version have their own set of issues and coding practices. Some major version upgrades may also break the contract logic if not handled properly.

Remediation
Instead of using different versions of the Solidity compiler with different bugs and security checks, it is better to use one version across all contracts.

Retest

This issue has been addressed by having the same pragma versions.

Bug ID #5 [Partially Fixed]

Missing Events in Important 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
The following functions were affected -

- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/KresusVaultFactory.sol#L36-L51
- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/KresusVault.sol#L123-L128

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 important functions to keep track of them.

Retest

Client’s Comment: Partially implemented due to alchemy implementation

Bug ID #6 [Fixed]

Floating and Outdated Pragma

Vulnerability Type

Floating Pragma (SWC-103)

Severity

Low

Description

Locking the pragma helps ensure that the contracts do not accidentally get deployed using an older version of the Solidity compiler affected by vulnerabilities.

The contract allowed floating or unlocked pragma to be used, i.e., ^0.8.23, ^0.8.20. This allows the contracts to be compiled with all the solidity compiler versions above the limit specified. The following contracts were found to be affected -

Affected Code

- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/Interfaces/IAccessControl.sol#L2
- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/Interfaces/IConstants.sol#L2
- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/helpers/MultisigAuth.sol#L2
- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/access/AccessControl.sol#L3
- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/common/BaseLightAccount.sol#L2
- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/common/BaseLightAccountFactory.sol#L2
- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/common/CustomSlotInitializable.sol#L4
- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/common/ERC1271.sol#L2

- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/KresusVault.sol#L2
- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/KresusVaultFactory.sol#L2

Impacts

If the smart contract gets compiled and deployed with an older or too recent version of the solidity compiler, there's a chance that it may get compromised due to the bugs present in the older versions or unidentified exploits in the new versions.

Incompatibility issues may also arise if the contract code does not support features in other compiler versions, therefore, breaking the logic.

The likelihood of exploitation is low.

Remediation

Keep the compiler versions consistent in all the smart contract files. Do not allow floating pragmas anywhere. It is suggested to use the 0.8.23 pragma version

Reference: https://swcregistry.io/docs/SWC-103

Retest

Floating pragma versions are now fixed as recommended.

Bug ID #7 [Fixed]

Boolean Equality

Vulnerability Type

Gas Optimization

Severity

Gas

Description

The contract was found to be equating variables with a boolean constant inside a "require()" statement which is not recommended and is unnecessary. Boolean constants can be used directly in conditionals.

Affected Code

- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/KresusVault.sol#L149-L155
- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/access/AccessControl.sol#L162-L168
- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/access/AccessControl.sol#L193-L199
- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/access/AccessControl.sol#L224-L230

Impacts

Equating the values to boolean constants in conditions cost gas and can be used directly.

Remediation

It is recommended to use boolean constants directly. It is not required to equate them to true or false.

Retest

This issue has been resolved by making changes as recommended.

Bug ID #8 [Won't Fix]

Splitting Require/Revert Statements

Vulnerability Type

Gas Optimization

Severity

Gas

Description

Require/Revert statements when combined using operators in a single statement usually lead to a larger deployment gas cost but with each runtime calls, the whole thing ends up being cheaper by some gas units.

Affected Code

- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/common/BaseLightAccount.sol#L78-L80
- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/common/BaseLightAccount.sol#L124-L131
- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/KresusVault.sol#L117-L119
- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/KresusVault.sol#L124-L126
- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/access/AccessControl.sol#L46-L52
- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/access/AccessControl.sol#L139-L145

Impacts

The multiple conditions in one **require/revert** statement combine require/revert statements in a single line, increasing deployment costs and hindering code readability.

Remediation

It is recommended to separate the **require/revert** statements with one statement/validation per line.

Retest

Client's Comment: Not implemented due to the nature of functionality

Bug ID #9 [Won’t Fix]

State Variable Can Be Marked As Constants

Vulnerability Type

Gas Optimization

Severity

Gas

Description

The contract has defined state variables whose values are never modified throughout the contract.

The variables whose values never change should be marked as constant to save **gas**.

Affected Code

- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/common/BaseLightAccountFactory.sol#L12

Impacts

Not marking unchanging state variables as constant in the contract can waste gas.

Remediation

Make sure that the values stored in the variables flagged above do not change throughout the contract. If this is the case, then consider setting these variables as **constant**.

Retest

Client’s Comment: Cannot change the functionality of alchemy implementation

Bug ID #10 [Won't Fix]

Cheaper Inequalities in if()

Vulnerability Type

Gas & Missing Best Practices

Severity

Gas

Description

The contract was found to be doing comparisons using inequalities inside the "if" statement. When inside the "if" statements, non-strict inequalities (>=, <=) are usually cheaper than the strict equalities (>, <).

Affected Code

- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/KresusVault.sol#L217
- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/KresusVault.sol#L290

Impacts

Using strict inequalities inside "if" statements costs more gas.

Remediation

It is recommended to go through the code logic, and, **if possible**, modify the strict inequalities with the non-strict ones to save gas as long as the logic of the code is not affected.

Retest:

Client's Comment: Cannot change the functionality of alchemy implementation

Bug ID#11 [Fixed]

Gas Optimization in Increments

Vulnerability Type

Gas optimization

Severity

Gas

Description

The contract uses two for loops**,** which use post increments for the variable "**i**".
The contract can save some gas by changing this to **++i**.
++i costs less gas compared to **i++** or **i += 1** for unsigned integers. In **i++**, the compiler has to create a temporary variable to store the initial value. This is not the case with **++i** in which the value is directly incremented and returned, thus, making it a cheaper alternative.

Vulnerable Code

- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/helpers/MultisigAuth.sol#L31

Impacts

Using **i++** instead of **++i** costs the contract deployment around 600 more gas units.

Remediation

It is recommended to switch to **++i** and change the code accordingly so the function logic remains the same and meanwhile saves some gas.

Retest

This issue has been addressed by making the changes as recommended.

Bug ID #12 [Fixed]

Unused Imports

Vulnerability Type
Gas Optimization

Severity
Gas

Description
The contract PositionRouter.sol was importing contracts ITimelock.sol & IVault.sol which was not used anywhere in the code. This increases the gas cost and overall contract's complexity.

Affected Code

- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/helpers/MultisigAuth.sol#L7
- https://github.com/kresuslabs/vault-contracts/blob/3d414145d54ea3b9a80ced802d0403a7d03114a4/contracts/common/ERC1271.sol#L5

Impacts
Unused imports in smart contracts can lead to an increase in the size of the code, making it more difficult to verify and potentially slowing down its execution. Moreover, having unused code in a smart contract can also increase the attack surface by potentially introducing vulnerabilities that can be exploited by malicious actors. This can lead to security issues and compromise the integrity of the contract.

Additionally, including unused imports in smart contracts can also increase deployment and gas costs, making it more expensive to deploy and run the contract on the Ethereum network.

Remediation

It is recommended to remove the import statement if the external contracts or libraries are not used anywhere in the contract.

Retest

This issue has been addressed by removing unused imports.

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.