Audited by

CredShields

Smart Contract Audit

March 12th, 2025 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Hemi Labs between March 4th, 2025, and March 7th, 2025. A retest was performed on March 10th, 2025.

Author
Shashank (Co-founder, CredShields) shashank@CredShields.com

Reviewers
Aditya Dixit (Research Team Lead), Shreyas Koli(Auditor), Naman Jain (Auditor), Sanket Salavi (Auditor), Yash Shah (Auditor)

Prepared for
Hemi Labs

Table of Contents

1. Executive Summary --------------------

Hemi Labs engaged CredShields to perform a smart contract audit from March 4th, 2025, to March 7th, 2025. During this timeframe, 10 vulnerabilities were identified. **A retest was performed on March 10th, 2025, and all the bugs have been addressed.**

During the audit, 1 vulnerability was found, with a severity rating of either high or critical. These vulnerabilities represent the greatest immediate risk to "Hemi Labs" and should be prioritized for remediation.

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
VUSD Contracts	0	1	3	0	2	4	**10**
	0	**1**	**3**	**0**	**2**	**4**	**10**

Table: Vulnerabilities Per Asset in Scope

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

2. The Methodology --------------------

Hemi Labs engaged CredShields to perform a VUSD 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 March 4th, 2025, to March 7th, 2025, 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/hemilabs/vusd-stablecoin/tree/54f9f235f26df813152b3d3235e7f4373ce473b6

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

Hemi Labs 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	● None	● 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 Summary -------------------

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, 10 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC | Vulnerability Type
Missing zero _minOut validation while swapping	High	Missing Input Validation
Missing handling of fees on transfer in ERC20 token transfers	Medium	Calculation Inaccuracy
claimCompAndConvertTo() swap with 'deadline: block.timestamp' offers no protection	Medium	Incorrect Deadline
Chainlink oracle Min/Max price validation	Medium	Missing Input Validation
Missing events in important functions	Low	Missing Best Practices
Outdated pragma	Low	Outdated Compiler Version
Public constants can be private	Gas	Gas Optimization
Cheaper inequalities in require()	Gas	Gas Optimization

Gas optimization in increments	Gas	Gas Optimization
Splitting require Statements	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	Bug ID #6
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 ------------------

Hemi Labs is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on March 10th, 2025, and all the issues have been addressed.**

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

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Missing zero _minOut validation while swapping	High	**Acknowledged** [March 10th, 2025]
Missing handling of fees on transfer in ERC20 token transfers	Medium	**Fixed** [March 10th, 2025]
claimCompAndConvertTo() swap with 'deadline: block.timestamp' offers no protection	Medium	**Acknowledged** [March 10th, 2025]
Chainlink oracle Min/Max price validation	Medium	**Acknowledged** [March 10th, 2025]
Missing events in important functions	Low	**Partially Fixed** [March 10th, 2025]
Outdated pragma	Low	**Acknowledged** [March 10th, 2025]
Public constants can be private	Gas	**Acknowledged** [March 10th, 2025]
Cheaper inequalities in require()	Gas	**Acknowledged** [March 10th, 2025]
Gas optimization in increments	Gas	**Acknowledged** [March 10th, 2025]
Splitting require Statements	Gas	**Acknowledged** [March 10th, 2025]

Table: Summary of findings and status of remediation

5. Bug Reports ------------------------

Bug ID #1 [Acknowledged]

Missing zero _minOut validation while swapping

Vulnerability Type
Missing Input Validation

Severity
High

Description
The claimCompAndConvertTo() function allows swapping COMP tokens to a specified token using a DEX router. The function accepts a _minOut parameter that determines the minimum output amount when performing the swap. However, the function does not validate that this value is greater than zero. When a keeper or governor calls this function with _minOut set to 0, they effectively approve any amount of tokens in return, even if it's substantially below market value.

Affected Code

- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Treasury.sol#L168-L187

Impacts
A malicious keeper can exploit this by front-running the transaction and manipulating the pool's price, causing the function to receive far fewer tokens than what would be fair for the input amount. This can lead to direct fund loss for the protocol as the swap could execute at highly unfavorable rates. MEV bots can also target these transactions, further extracting value from the protocol.

Remediation
It is recommended to add a validation to the _minOut is not equal to zero.

Retest
Client's comment: Won't fix assuming the keeper role is a trusted role.

Bug ID #2 [Fixed]

Missing handling of fees on transfer in ERC20 token transfers

Vulnerability Type
Calculation Inaccuracy

Severity
Medium

Description
The contract contains a vulnerability related to transferring ERC20 tokens without considering the possibility of fees charged on transfer. Some ERC20 tokens implement a fee mechanism, where a certain percentage of tokens is deducted as a fee during each transfer. However, the contract does not account for this possibility when transferring tokens using the safeTransferFrom/TransferFrom function.

Affected Code

- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Minter.sol#L237-L249

Impacts
Failure to account for transfer fees can lead to accounting errors and financial losses. When tokens with transfer fees are transferred, the actual received amount may be less than expected due to the deduction of fees. As a result, the contract's internal accounting and balance tracking may become inaccurate, leading to discrepancies in token balances and potential financial losses for users.

Remediation
To address this vulnerability, it is recommended to add a mechanism to calculate the balance of the contract before and after the transfer is completed.

Retest
This issue has been fixed.

Bug ID #3 [Acknowledged]

claimCompAndConvertTo() swap with 'deadline: block.timestamp' offers no protection

Vulnerability Type
Incorrect Deadline

Severity
Medium

Description
Using block.timestamp as deadline parameter in claimCompAndConvertTo() during interaction with router, which suggests that whenever the miner decides to include the transaction in a block, it will be valid at that time, since block.timestamp will be the current timestamp.
Whichever block the txn will be included in will be block.timestamp, so setting the deadline to block.timestamp offers no protection as validators can hold the transaction indefinitely.

Affected Code

- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Treasury.sol#L183

Impacts
It may be more profitable for a miner to deny the transaction from being mined until the transaction incurs the maximum amount of slippage. A malicious miner can hold the transaction as the deadline is set to block.timestamp, which means that whenever the miner decides to include the transaction in a block, it will be valid at that time, since block.timestamp will be the current timestamp. The transaction might be left hanging in the mempool and be executed way later than the user wanted.

Remediation
It is recommended to add a deadline argument to the claimCompAndConvertTo() function that interacts with the router, and pass it along to the router calls.

Retest
Client's comment: will fix it in future when we have plan to update treasury contract.

Bug ID #4 [Acknowledged]

Chainlink oracle Min/Max price validation

Vulnerability Type
Missing Input Validation

Severity
Medium

Description
Chainlink has a library AggregatorV3Interface with a function called latestRoundData(). This function returns the price feed among other details for the latest round.
Chainlink aggregators have a built-in circuit breaker if the price of an asset goes outside of a predetermined price band. The result is that if an asset experiences a huge drop in value, the price of the oracle will continue to return the minPrice instead of the actual price of the asset.

Affected Code

- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Redeemer.sol#L137
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Minter.sol#L214
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Minter.sol#L259

Impacts
This would allow users to store their allocations with the asset but at the wrong price.

Remediation
The contract should check the returned answer/price against the minPrice/maxPrice and revert if the answer is outside of the bounds.

```
if (price >= maxPrice or price <= minPrice) revert(); // eg
```

Retest
Client's comment: We acknowledge this issue. The contract already has a price tolerance parameter. This is not an issue if price tolerance is in a narrow range and less than circuit breaker

prices. The governor is supposed to set price tolerance in a very narrow range (may be less than 10%).

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

- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Governed.sol#L43-L47
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Treasury.sol#L96-L105
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Treasury.sol#L112-L118
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Treasury.sol#L135-L138
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Treasury.sol#L144-L146
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Treasury.sol#L308-L318
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Treasury.sol#L96-L105
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Minter.sol#L80-L89
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Minter.sol#L95-L100

- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Minter.sol#L206-L229

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: Added missing events in Minter.sol. Presently not upgrading Treasury.sol so will fix other contract in future

Bug ID #6 [Acknowledged]

Outdated pragma

Vulnerability Type
Outdated Compiler Version (SWC-102)

Severity
Low

Description
The smart contract is using an outdated version of the Solidity compiler specified by the pragma directive i.e. 0.8.24. Solidity is actively developed, and new versions frequently include important security patches, bug fixes, and performance improvements. Using an outdated version exposes the contract to known vulnerabilities that have been addressed in later releases. Additionally, newer versions of Solidity often introduce new language features and optimizations that improve the overall security and efficiency of smart contracts.

Affected Code
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Governed.sol#L3
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Minter.sol#L3
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Redeemer.sol#L3
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Treasury.sol#L3
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/VUSD.sol#L3

Impacts
The use of an outdated Solidity compiler version can have significant negative impacts. Security vulnerabilities that have been identified and patched in newer versions remain exploitable in the deployed contract.
Furthermore, missing out on performance improvements and new language features can result in inefficient code execution and higher gas costs.

Remediation
It is suggested to use the 0.8.28 pragma version.

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

Retest

Client's comment: Will fix it in near future when we have plan to update all contracts including Treasury

Bug ID #7 [Acknowledged]

Public constants can be private

Vulnerability Type
Gas Optimization

Severity
Gas

Description
Public constant variables cost more gas because the EVM automatically creates getter functions for them and adds entries to the method ID table. The values can be read from the source code instead.

Affected Code

- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Redeemer.sol#L14
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Redeemer.sol#L15
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Redeemer.sol#L20
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Treasury.sol#L19
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Treasury.sol#L20
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Minter.sol#L19
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Minter.sol#L20
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Minter.sol#L27

Impacts
Public constants are more costly due to the default getter functions created for them, increasing the overall gas cost.

Remediation

If reading the values for the constants is not necessary, consider changing the public visibility to private.

Retest

Client's comment: Name and version contents are intentionally kept public for UI.

Bug ID #8 [Acknowledged]

Cheaper inequalities in require( )

Vulnerability Type
Gas Optimization

Severity
Gas

Description
The contract was found to be performing comparisons using inequalities inside the require statement. When inside the require statements, non-strict inequalities (>=, <=) are usually costlier than strict equalities (>, <).

Affected Code

- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Redeemer.sol#L40
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Redeemer.sol#L49
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Redeemer.sol#L139
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Minter.sol#L108
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Minter.sol#L114
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Minter.sol#L129
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Minter.sol#L224
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Minter.sol#L268
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Minter.sol#L274

Impacts
Using non-strict inequalities inside "require" statements costs more gas.

Remediation

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

Retest:
Client Fix : Some of these are not possible to use '>=' because we need to use subtract or addition operation . At the end gas cost will be more or less same.

Bug ID #9 [Acknowledged]

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/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/VUSD.sol#L49
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Treasury.sol#L197
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Treasury.sol#L248
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Treasury.sol#L237
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Treasury.sol#L322

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
Client Fix : will fix it when we have plan to upgrade treasury contract.

Bug ID #10 [Acknowledged]

Splitting require Statements

Vulnerability Type
Gas Optimization

Severity
Gas

Description
Require 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/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Redeemer.sol#L139
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Minter.sol#L224
- https://github.com/hemilabs/vusd-stablecoin/blob/54f9f235f26df813152b3d3235e7f4373ce473b6/contracts/Minter.sol#L268

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

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

Retest
Client's comment: We would leave this as is for readability purpose.

6. The Disclosure ---------------------

The Reports provided by CredShields are 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.

YOUR SECURE FUTURE STARTS HERE

At CredShields, we're more than just auditors. We're your strategic partner in ensuring a secure Web3 future. Our commitment to your success extends beyond the report, offering ongoing support and guidance to protect your digital assets

Audited by