Audited by

CredShields

Smart Contract Audit

March 24th, 2025 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of RealProton Inc. between March 6th, 2025, and March 12th, 2025. A retest was performed on March 21st, 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

RealProton Inc.

Table of Contents

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

RealProton Inc. engaged CredShields to perform a smart contract audit from March 6th, 2025, to March 12th, 2025. During this timeframe, 6 vulnerabilities were identified. **A retest was performed on March 21st, 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 "RealProton Inc. " 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
RealProton Smart Contracts	0	1	0	3	0	2	**6**
	0	**1**	**0**	**3**	**0**	**2**	**6**

Table: Vulnerabilities Per Asset in Scope

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

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

RealProton Inc. engaged CredShields to perform a RealProton 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 6th, 2025, to March 12th, 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/RealProtonOfficial/realproton_smart_contracts/tree/24219439c7b4b1963653543cf596b7b033e69355

2.1.2 Documentation

Documentation was provided by the RealProton team, and they promptly answered all the questions related to the audit and smart contracts.

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 emphasizes understanding core concepts, preparing test cases, and evaluating business logic for potential vulnerabilities.

2.2 Retesting Phase

RealProton Inc. is actively partnering with CredShields to validate the remediations implemented toward 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, 6 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC | Vulnerability Type
Call to the addWallet() function will always revert	High	Denial-of-service (DoS)
Floating and Outdated Pragma	Low	Floating Pragma
Missing Zero Address Validations	Low	Missing Input Validation
Missing Events in Important Functions	Low	Missing Best Practices
Splitting require Statements	Gas	Gas Optimization
Cheaper Inequalities in require()	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 #4
SWC-103	Floating Pragma	Vulnerable	Bug ID #4
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 -----------------

RealProton Inc. is actively partnering with CredShields from this engagement to validate the remediation of the discovered vulnerabilities. **A retest was performed on March 21st, 2025, and all the issues have been addressed.**

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

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Call to the addWallet() function will always revert	High	**Fixed** [March 21, 2025]
Floating and Outdated Pragma	Low	**Fixed** [March 21, 2025]
Missing Zero Address Validations	Low	**Fixed** [March 21, 2025]
Missing Events in Important Functions	Low	**Fixed** [March 21, 2025]
Splitting require Statements	Gas	**Fixed** [March 21, 2025]
Cheaper Inequalities in require()	Gas	**Fixed** [March 21, 2025]

Table: Summary of findings and status of remediation

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

Bug ID #1 [Fixed]

Call to the addWallet() function will always revert

Vulnerability Type
Denial-of-service (DoS)

Severity
High

Description
The addWallet() function is designed to allow a user to link a new wallet address to an existing identity. However, a logical flaw in the implementation prevents this function from ever succeeding. Specifically, the function first checks whether walletToIdentity[msg.sender] == address(0), and if this condition fails, it immediately reverts with "Wallet is already registered or linked." This ensures that only unregistered wallets proceed further. However, the function then assigns walletToIdentity[msg.sender] to _identity and subsequently checks if _identity != address(0). Since the function has already ensured that walletToIdentity[msg.sender] == address(0), the _identity variable is always set to address(0), causing the next require() statement to always revert with "Caller is not linked to any identity."

Affected Code

- https://github.com/RealProtonOfficial/realproton_smart_contracts/blob/24219439c7b4b1963653543cf596b7b033e69355/Realproton3643/src/IdentityRegistry.sol#L1582-L1607

Impacts
Users who wish to link multiple wallets to their identity will be unable to do so, leading to a loss of functionality.

Remediation
It is recommended to implement a check to verify if msg.sender is linked to an identity in a way that does not contradict previous conditions

Retest
This issue has been resolved.

Bug ID #2 [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., >= 0.8.0. 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/RealProtonOfficial/realproton_smart_contracts/blob/24219439c7b4b1963653543cf596b7b033e69355/Realproton3643/src/Realproton.sol#L2
- https://github.com/RealProtonOfficial/realproton_smart_contracts/blob/24219439c7b4b1963653543cf596b7b033e69355/Realproton3643/src/Realproton.sol#L540

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 0.8.28 pragma version
Reference: https://swcregistry.io/docs/SWC-103

Retest
This issue has been fixed.

Bug ID #3 [Fixed]

Missing Zero Address Validations

Vulnerability Type
Missing Input Validation

Severity
Low

Description:
The contracts were found to be setting new addresses without proper validations for zero addresses.
Address type parameters should include a zero-address check otherwise contract functionality may become inaccessible or tokens burned forever.
Depending on the logic of the contract, this could prove fatal and the users or the contracts could lose their funds, or the ownership of the contract could be lost forever.

Affected Code

- https://github.com/RealProtonOfficial/realproton_smart_contracts/blob/24219439c7b4b1963653543cf596b7b033e69355/Realproton3643/src/Realproton.sol#L2219
- https://github.com/RealProtonOfficial/realproton_smart_contracts/blob/24219439c7b4b1963653543cf596b7b033e69355/Realproton3643/src/Realproton.sol#L2221

Impacts
If address type parameters do not include a zero-address check, contract functionality may become unavailable or tokens may be burned permanently.

Remediation
Add a zero address validation to all the functions where addresses are being set.

Retest
This issue has been fixed.

Bug ID #4 [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/RealProtonOfficial/realproton_smart_contracts/blob/24219439c7b4b1963653543cf596b7b033e69355/Realproton3643/src/Realproton.sol#L2401
- https://github.com/RealProtonOfficial/realproton_smart_contracts/blob/24219439c7b4b1963653543cf596b7b033e69355/Realproton3643/src/Realproton.sol#L2413
- https://github.com/RealProtonOfficial/realproton_smart_contracts/blob/24219439c7b4b1963653543cf596b7b033e69355/Realproton3643/src/Realproton.sol#L3038

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
The issue has been resolved by adding events in important functions.

Bug ID #5 [Fixed]

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/RealProtonOfficial/realproton_smart_contracts/blob/24219439c7b4b1963653543cf596b7b033e69355/Realproton3643/src/Realproton.sol#L2603
- https://github.com/RealProtonOfficial/realproton_smart_contracts/blob/24219439c7b4b1963653543cf596b7b033e69355/Realproton3643/src/Realproton.sol#L2998
- https://github.com/RealProtonOfficial/realproton_smart_contracts/blob/24219439c7b4b1963653543cf596b7b033e69355/Realproton3643/src/Realproton.sol#L3097

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
This issue has been resolved.

Bug ID #6 [Fixed]

Cheaper Inequalities in require( )

Vulnerability Type
Gas & Missing Best Practices

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/RealProtonOfficial/realproton_smart_contracts/blob/24219439c7b4b1963653543cf596b7b033e69355/Realproton3643/src/Realproton.sol#L2473
- https://github.com/RealProtonOfficial/realproton_smart_contracts/blob/24219439c7b4b1963653543cf596b7b033e69355/Realproton3643/src/Realproton.sol#L2478
- https://github.com/RealProtonOfficial/realproton_smart_contracts/blob/24219439c7b4b1963653543cf596b7b033e69355/Realproton3643/src/Realproton.sol#L3000
- https://github.com/RealProtonOfficial/realproton_smart_contracts/blob/24219439c7b4b1963653543cf596b7b033e69355/Realproton3643/src/Realproton.sol#L3008
- https://github.com/RealProtonOfficial/realproton_smart_contracts/blob/24219439c7b4b1963653543cf596b7b033e69355/Realproton3643/src/Realproton.sol#L3099
- https://github.com/RealProtonOfficial/realproton_smart_contracts/blob/24219439c7b4b1963653543cf596b7b033e69355/Realproton3643/src/Realproton.sol#L3106
- https://github.com/RealProtonOfficial/realproton_smart_contracts/blob/24219439c7b4b1963653543cf596b7b033e69355/Realproton3643/src/Realproton.sol#L3268
- https://github.com/RealProtonOfficial/realproton_smart_contracts/blob/24219439c7b4b1963653543cf596b7b033e69355/Realproton3643/src/Realproton.sol#L3296

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:

This issue has been fixed wherever possible due to their business logic

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