CredShields
Smart Contract Audit

April 4th, 2024

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of ResearchHub between Feb 15th, 2024, and Feb 20th, 2024. And a retest was performed on April 1st, 2024.

Author

Shashank (Co-founder, CredShields)

shashank@CredShields.com

Reviewers

Aditya Dixit (Research Team Lead)

aditya@CredShields.com

Prepared for

ResearchHub

Table of Contents

1. Executive Summary

ResearchHub engaged CredShields to perform a smart contract audit from Feb 15th, 2024, to Feb 20th, 2024. During this timeframe, Thirteen (13) vulnerabilities were identified. **A retest was performed on April 1st, 2024.**

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 "ResearchHub" 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	4	1	4	4	**13**
	0	**0**	**4**	**1**	**4**	**4**	**13**

Table: Vulnerabilities Per Asset in Scope

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

2. Methodology

ResearchHub engaged CredShields to perform a ResearchHub 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 Feb 15th, 2024, to Feb 20th, 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
https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code

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

ResearchHub 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, Thirteen (13) security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC | Vulnerability Type
Lack of Zero Address Validation in changeController Function	Medium	Zero Address Check
Lack of Allowance Check for Controller in transferFrom Function	Medium	Centralization Risk
Inadequate Allowance Handling in approve and approveAndCall Functions	Medium	DOS
Unrestricted Access in tokenFactory.createCloneToken Function	Medium	Lack of Access Control
Floating and Outdated Pragma	Low	Floating Pragma (SWC-103)
Require with Empty Message	Informational	Code optimization

Use Call instead of Transfer	Informational	Best Practices
Using EXTCODESIZE To Check For Externally Owned Accounts	Informational	Misconfiguration
Missing State Variable Visibility	Informational	Missing Best Practices
Variables should be Immutable	Gas	Gas Optimization
Boolean Equality	Gas	Gas Optimization
Multiplication/Division by 2 should use Bit-Shifting	Gas	Gas Optimization
Functions should be declared External	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	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

ResearchHub is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on April 1st, 2024, and all the issues have been addressed.**

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

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Lack of Zero Address Validation in changeController Function	Medium	**Fixed [01/04/2024]**
Lack of Allowance Check for Controller in transferFrom Function	Medium	**Fixed [01/04/2024]**
Inadequate Allowance Handling in approve and approveAndCall Functions	Medium	**Pending Fix**
Unrestricted Access in tokenFactory.createCloneToken Function	Medium	**Pending Fix**
Floating and Outdated Pragma	Low	**Won't Fix [01/04/2024]**
Require with Empty Message	Informational	**Won't Fix [01/04/2024]**
Use Call instead of Transfer	Informational	**Won't Fix [01/04/2024]**
Using EXTCODESIZE To Check For Externally Owned Accounts	Informational	**Won't Fix [01/04/2024]**

Missing State Variable Visibility	Informational	**Won’t Fix [01/04/2024]**
Variables should be Immutable	Gas	**Won’t Fix [01/04/2024]**
Boolean Equality	Gas	**Won’t Fix [01/04/2024]**
Multiplication/Division by 2 should use Bit-Shifting	Gas	**Won’t Fix [01/04/2024]**
Functions should be declared External	Gas	**Won’t Fix [01/04/2024]**

Table: Summary of findings and status of remediation

5. Bug Reports

Bug ID #1 [Fixed]

Lack of Zero Address Validation in changeController Function

Vulnerability Type

Zero Address Check

Severity

Medium

Description:

The `changeController()` and `changeOwner()` function in the given smart contract is used to change the controller and owner address, which effectively transfers ownership. However, the function does not include validation to check whether the new owner/controller address provided is a zero address (0x0). This omission can potentially lead to a situation where the controller role is transferred to the zero address accidentally, resulting in a loss of control over the contract.

Affected Code

- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L68
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L693
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L642
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L74

Impacts

If the new owner/controller address provided is a zero address, control over the contract will be transferred to an invalid address, effectively resulting in loss of ownership, contract compromise, and potential fund loss.

Remediation

To mitigate the risk associated with transferring ownership to a zero address, consider implementing zero address validation within the `changeController()` and `changeOwner()` functions. Additionally, implementing a two-step ownership transfer mechanism can provide an added layer of security and prevent accidental ownership transfers.

Retest

This is fixed as the contract has been renounced.

Bug ID #2 [Fixed]

Lack of Allowance Check for Controller in transferFrom Function

Vulnerability Type

Centralization Risk

Severity

Medium

Description:

The `transferFrom` function in the given smart contract allows the controller to transfer tokens from one account to another without checking the allowance. Instead of verifying the allowance of the controller, the function directly performs the transfer if the sender is the controller address. This omission poses a centralization risk as it enables the controller to impersonate user balances and transfer tokens without proper authorization.

Affected Code

- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L190
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L196

Impacts

The controller can potentially exploit this vulnerability to transfer tokens from one account to another without allowance. This can result in unauthorized token movements and undermine the integrity of the token ecosystem.

Remediation

Allowing the controller to bypass the allowance check undermines the decentralized nature of the token transfer mechanism. It grants the controller excessive power to manipulate token balances and transfer tokens without the consent of token holders.

Retest

This is fixed as the contract has been renounced.

Bug ID #3 [Pending Fix]

Inadequate Allowance Handling in `approve` and `approveAndCall` Functions

Vulnerability Type

DOS

Severity

Medium

Description:

In the given contract, the `approveAndCall` function is used to approve a spender and immediately trigger the `receiveApproval` function of the spender contract. However, there is a vulnerability in the `approveAndCall` function where it does not handle the case where the user already has an existing allowance for the spender. `Approve` function reverts if the user already has some allowance. `approveAndCall` should apply validation to check whether the spender already has some allowance. if it has then it should change it to zero and then change the allowance otherwise `approveAndCall` will always revert.

Affected Code

- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L292
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L298

Impacts

If a user calls `approveAndCall` with a nonzero allowance for the spender, the function will revert. This behavior restricts users from approving spenders if they already have an existing allowance, leading to a suboptimal user experience

Remediation

Implement a safe mechanism in the `approveAndCall` function to handle cases where the user already has a nonzero allowance for the spender. Before approving the spender, check if the user has a nonzero allowance and reset it to zero if necessary.

Retest

The team has decided not to fix this at the moment.

Bug ID #4 [Pending Fix]

Unrestricted Access in `tokenFactory.createCloneToken` Function

Vulnerability Type

Lack of Access Control

Severity

Medium

Description:

The `createCloneToken` function in the given smart contract creates a clone token by calling the `createCloneToken` function from `tokenFactory`. However, the `createCloneToken` function in `tokenFactory` does not enforce any access control, allowing any user to call it and potentially create a cloned token with a user-controlled address as the parent token.

Affected Code

- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L378
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L583
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L384
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L586

Impacts

By setting the parent token address to a user-controlled address, an attacker can manipulate the balance of the cloned token, affecting the functionality and integrity of the smart contract's operations that rely on the parent token's balance.

Remediation

Modify the `createCloneToken` function in `tokenFactory` to include access control mechanisms, such as only allowing only MiniMeToken to create clone tokens in `tokenFactory.`

Retest

The team has decided not to fix the bug at this moment.

Bug ID #5 [Won't Fix]

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., >= 0.8.9. 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://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L5
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L9

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 0.8.20 pragma version

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

Retest

Due to low severity and negligible exploitation possibilities this will not be fixed and CredShields team agrees to the decision.

Bug ID #6 [Won't Fix]

Require with Empty Message

Vulnerability Type
Code optimization

Severity
Informational

Description
During analysis; multiple require statements were detected with empty messages. The statement takes two parameters, and the message part is optional. This is shown to the user when and if the require statement evaluates to false. This message gives more information about the conditional and why it gave a false response.

Affected Code

- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L58
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L180
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L197
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L217
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L219
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L237
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L257
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L263

- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L268
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L293
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L414
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L416
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L430
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L432
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L487
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L523
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L525
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L625
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L64
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L186
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L203
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L223
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L225
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L235
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L243

- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L263
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L269
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L274
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L299
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L420
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L422
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L436
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L438
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L527
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L529

Impacts

Having a short descriptive message in the require statement gives users and developers more details as to why the conditional statement failed and helps in debugging the transactions.

Remediation

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

Retest:

This is a suggested best practice and not a security vulnerability and the team has decided not to fix it because it will require redeployment of code and CredShields team agrees with the decision.

Bug ID #7 [Won't Fix]

Use Call instead of Transfer

Vulnerability Type
Best Practices

Severity
Informational

Description:
Using Solidity's transfer function has some notable shortcomings when the withdrawer is a smart contract, which can render ETH deposits impossible to withdraw. Specifically, the withdrawal will inevitably fail when:

- The withdrawer smart contract does not implement a payable fallback function.
- The withdrawer smart contract implements a payable fallback function which uses more than 2300 gas units.
- The withdrawer smart contract implements a payable fallback function which needs less than 2300 gas units but is called through a proxy that raises the call's gas usage above 2300.

Affected Code

- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L538
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L542

Impacts
The transfer function has some restrictions when it comes to sending ETH to contracts in terms of gas which could lead to transfer failure in some cases.

Remediation

It is recommended to transfer ETH using the call() function, handle the return value using require statement, and use the nonreentrant modifier wherever necessary to prevent reentrancy.
Ref: https://solidity-by-example.org/sending-ether/

Retest

This is a suggested best practice and not a security vulnerability and the team has decided not to fix it because it will require redeployment of code and CredShields team agrees with the decision.

Bug ID #8 [Won't Fix]

Using EXTCODESIZE To Check For Externally Owned Accounts

Vulnerability Type
Misconfiguration

Severity
Informational

Description
Upon reviewing the code, it has come to attention that the extcodesize opcode is used to determine whether an account is an externally owned account or another contract. While extcodesize typically returns 0 for externally owned accounts, there is an important consideration regarding its behavior during contract deployment or constructor execution. Specifically, when a contract is under construction or its constructor is running, extcodesize for the contract's address returns zero. This behavior can lead to inaccurate results when attempting to identify externally owned accounts during these specific circumstances.

Affected Code

- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L512
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L508

Impacts
During contract deployment or constructor execution, the extcodesize check may incorrectly identify the account as externally owned due to the opcode's behavior returning zero.

Remediation
To accurately identify externally owned accounts, consider using alternative methods or checks that are not affected by the behavior of extcodesize during contract deployment or constructor execution.

Retest

This is a suggested best practice and not a security vulnerability and the team has decided not to fix it because it will require redeployment of code and CredShields team agrees with the decision.

Bug ID #9 [Won't Fix]

Missing State Variable Visibility

Vulnerability Type
Missing Best Practices

Severity
Informational

Description
In Solidity, the visibility of state variables is important as it determines how those variables can be accessed and modified by other contracts or functions.
The contract defined state variables that were missing a visibility modifier.

Affected Code

- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L119
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L122
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L125
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L125
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L128
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L131

Impacts
If the visibility of a state variable is accidentally left out, it can cause unexpected behavior and security vulnerabilities. For example, if a state variable is supposed to be private and is accidentally declared without any visibility keyword, it will be treated as "internal" by

default, which may lead to it being accessible by other contracts or functions outside the intended scope. This can lead to a potential attack vector for malicious actors.

Remediation

Explicitly define visibility for all state variables. These variables can be specified as public, internal, or private.

Retest

This is a suggested best practice and not a security vulnerability and the team has decided not to fix it because it will require redeployment of code and CredShields team agrees with the decision.

Bug ID #10 [Won't Fix]

Variables should be Immutable

Vulnerability Type
Gas Optimization

Severity
Gas

Description:
Declaring state variables that are not updated following deployment as immutable can save gas costs in smart contract deployments and function executions. Immutable state variables are those that cannot be changed once they are initialized, and their values are set permanently.

By declaring state variables as immutable, the compiler can optimize their storage in a way that reduces gas costs. Specifically, the compiler can store the value directly in the bytecode of the contract, rather than in storage, which is a more expensive operation.

Affected Code:
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L88
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L107
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L111
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L114
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L131
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L622

- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L94
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L113
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L117
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L120
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L137

Impacts:

Gas usage is increased if the variables that are not updated outside of the constructor are not set as immutable.

Remediation:

An "`immutable`" attribute should be added in the parameters that are never updated outside of the constructor to save the gas.

Retest

This is a suggested best practice and not a security vulnerability and the team has decided not to fix it because it will require redeployment of code and CredShields team agrees with the decision.

Bug ID #11 [Won't Fix]

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://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L229
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L268
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L525
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L235
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L274
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L529

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 is a suggested best practice and not a security vulnerability and the team has decided not to fix it because it will require redeployment of code and CredShields team agrees with the decision.

Bug ID #12 [Won't Fix]

Multiplication/Division by 2 should use Bit-Shifting

Vulnerability Type

Gas Optimization

Severity

Gas

Description

In Solidity, the EVM (Ethereum Virtual Machine) executes operations in terms of gas consumption, where gas represents the computational cost of executing smart contract functions. Multiplication and division by two can be achieved using either traditional multiplication and division operations or bitwise left shift (<<) and right shift (>>) operations, respectively. However, using bit-shifting operations is more gas-efficient than using traditional multiplication and division operations.

- x * 2 can be replaced with x << 1.
- x / 2 can be replaced with x >> 1.

Affected Code

- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L472
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L478

Impacts

Gas consumption directly affects the cost of executing smart contracts on the Ethereum blockchain. Using bit-shifting operations for multiplication and division by two reduces the gas cost from 5 to 3, leading to more cost-effective and efficient smart contract execution. This optimization is particularly relevant in scenarios where gas efficiency is crucial, such as high-frequency operations or resource-intensive contracts.

Remediation

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

Retest

This is a suggested best practice and not a security vulnerability and the team has decided not to fix it because it will require redeployment of code and CredShields team agrees with the decision.

Bug ID #13 [Won't Fix]

Functions should be declared External

Vulnerability Type
Gas Optimization

Severity
Gas

Description
Public functions that are never called by a contract should be declared external in order to conserve gas.
The following functions were declared as public but were not called anywhere in the contract, making public visibility useless.

Affected Code
The following functions were affected -

- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L196
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L287
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L418

- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L434
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L452
- https://etherscan.io/token/0xd101dcc414f310268c37eeb4cd376ccfa507f571#code#L540
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L642
- https://etherscan.io/address/0x29070e64d6a9ea9eec83ee83c88d0c51d7efd257#code#L653

Impacts

Smart Contracts are required to have effective Gas usage as they cost real money and each function should be monitored for the amount of gas it costs to make it gas efficient.
"**public**" functions cost more Gas than "**external**" functions.

Remediation

Use the "**external**" state visibility for functions that are never called from inside the contract.

Retest

This is a suggested best practice and not a security vulnerability and the team has decided not to fix it because it will require redeployment of code and CredShields team agrees with the decision.

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.