CredShields

Smart Contract Audit

Jun 4th, 2024 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contracts audit performed by CredShields Technologies PTE. LTD. on behalf of Rex Protocol between May 5th, 2024, and May 15th, 2024. And a retest was performed on May 30th, 2024.

Author

Shashank (Co-founder, CredShields)

shashank@CredShields.com

Reviewers

- Aditya Dixit (Research Team Lead) - aditya@CredShields.com
- Shreyas Koli (Auditor) - shreyaskoli@credshields.com
- Naman Jain (Auditor) - naman@credshields.com

Prepared for

Rex Protocol

Table of Contents

1. Executive Summary

Rex Protocol engaged CredShields to perform a smart contract audit from May 5th, 2024, to May 15th, 2024. During this timeframe, 25 vulnerabilities were identified. **A retest was performed on May 30th, 2024, and all the bugs have been addressed.**

During the audit, Six (6) vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Rex Protocol" 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 Contracts	1	5	4	5	3	7	**25**
	1	**5**	**4**	**5**	**3**	**7**	**25**

Table: Vulnerabilities Per Asset in Scope

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

2. Methodology

Rex Protocol engaged CredShields to perform a Rex Protocol 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 May 5th, 2024, to May 15th, 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
Repository: https://github.com/REXProtocol/REX-SmartContract-Testing/ **Commits:** • ea4b012192bf7e7ec555b041fa8c0d3ed6e751f8 • d9f15c8a8c99e5965bd003f1bd59d7c5ca1cefdc • e7c2aa5efb84008ba0dbb6cf358fd635881e43ce • 96046275d2ad23bda51e86d979e6887b83833e0a

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

Rex Protocol 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 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, 25 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC | Vulnerability Type
Missing Call to alterUsersEarningRateIndex in deposit_token_for Function	High	Business Logic
Potential Underflow Vulnerability in Fee Calculation and Asset Freezing	High	Denial of Service
Missing Handling of Fees on Transfer in ERC20 Token Transfers	High	Improper Calculation
Incorrect Conditions in revertTrade Function	High	Business Logic
Flawed Logic in revertTrade Function	High	Business Logic
Lack of Access Control in revertTrade Function	High	Missing Access Control

Lack of Access Control in chargeMassinterest() Function	Medium	Missing Access Control
Potential Reentrancy Vulnerability in SubmitOrder Function	Medium	Reentrancy
Missing Functionality and Error Handling in fulfill() Function	Medium	Missing Functionality
Function Call Mismatch in SubmitOrder() Function	Medium	Function Call Mismatch
Funds Stuck in Smart Contract due to Lack of Withdrawal Mechanism	Low	Lock-up Of Assets
Incomplete Admin Role Revocation in alterAdminRoles() Function	Low	Missing Access Control
Floating and Outdated Pragma	Low	Floating Pragma
Use Ownable2Step	Low	Missing Best Practices
Missing Events in Important Functions	Low	Missing Best Practices
Missing State Variable Visibility	Informational	Missing Best Practices
Hardcoded Static Address	Informational	Missing Best Practices
Require with Empty Message	Informational	Code optimization
Dead Code	Gas	Gas Optimization
Variables should be Immutable	Gas	Gas Optimization
Boolean Equality	Gas	Gas Optimization

State Variable Can Be Marked As Constants	Gas	Gas Optimization
Cheaper Conditional Operators	Gas	Gas Optimization
Functions should be declared External	Gas	Gas Optimization
Gas Optimization in 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	Outdated pragma version used
SWC-103	Floating Pragma	Not Vulnerable	The 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	Vulnerable	Bug ID #8
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	Bug ID #9
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	Bug ID #19
SWC-136	Unencrypted Private Data On-Chain	Not Vulnerable	No such scenario was found

4. Remediation Status

Rex Protocol is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on May 30th, 2024, and all the issues have been addressed.**

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

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Missing Call to alterUsersEarningRateIndex in deposit_token_for Function	High	Fixed
Potential Underflow Vulnerability in Fee Calculation and Asset Freezing	High	Fixed
Missing Handling of Fees on Transfer in ERC20 Token Transfers	High	Fixed
Incorrect Conditions in revertTrade Function	High	Fixed
Flawed Logic in revertTrade Function	High	Fixed
Lack of Access Control in revertTrade Function	High	Fixed
Lack of Access Control in chargeMassinterest() Function	Medium	Fixed

Potential Reentrancy Vulnerability in SubmitOrder Function	Medium	Fixed
Missing Functionality and Error Handling in fulfill() Function	Medium	Fixed
Function Call Mismatch in SubmitOrder() Function	Medium	Fixed
Funds Stuck in Smart Contract due to Lack of Withdrawal Mechanism	Low	Fixed
Incomplete Admin Role Revocation in alterAdminRoles() Function	Low	Fixed
Floating and Outdated Pragma	Low	Fixed
Use Ownable2Step	Low	Fixed
Missing Events in Important Functions	Low	Not Fixed
Missing State Variable Visibility	Informational	Partially Fixed
Hardcoded Static Address	Informational	Fixed
Require with Empty Message	Informational	Not Fixed
Dead Code	Gas	Fixed
Variables should be Immutable	Gas	Not Fixed
Boolean Equality	Gas	Fixed
State Variable Can Be Marked As Constants	Gas	Not Fixed

Cheaper Conditional Operators	Gas	Fixed
Functions should be declared External	Gas	Fixed
Gas Optimization in Require Statements	Gas	Partially Fixed

Table: Summary of findings and status of remediation

5. Bug Reports

Bug ID#1 [Fixed]

Missing Call to alterUsersEarningRateIndex in deposit_token_for Function

Vulnerability Type
Business Logic Flaw

Severity
Critical

Description
In the deposit_token_for() function of the given contract, there is a missing call to alterUsersEarningRateIndex() when a user deposits funds for a token they don't currently hold. The function checks the length of the tokens array obtained from the ReadUserData() function to determine if the user already has tokens. If the length is zero, indicating that the user has no tokens, the function increments the totalHistoricalUsers counter and then calls alterUsersEarningRateIndex(). However, it fails to check if the user holds the specific token being deposited. As a result, if the user already holds other tokens, the function skips calling alterUsersEarningRateIndex(), potentially leading to incorrect interest calculations for the deposited token. This becomes critical if the current interest rate variable of that user is zero. In such a case, while accruing the interest, it will accrue from zero to the current interest rate of the market, resulting in a fund drain of the protocol.

Affected Code

- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/96046275d2ad23bda51e86d979e6887b83833e0a/contracts/depositvault.sol#L395-L398

Impacts

The missing call to alterUsersEarningRateIndex() when a user deposits funds for a token they don't currently hold can lead to incorrect interest rate indexing. This vulnerability allows an attacker to deposit funds for a token they don't hold, bypassing the interest rate indexing mechanism and potentially receiving more rewards than expected. Additionally, if the current interest rate variable of the user is zero, it can result in a fund drain of the protocol as interest accrues from zero to the current market interest rate.

Remediation

Update the deposit_token_for() function to call alterUsersEarningRateIndex() only if the user doesn't hold the specific token being deposited. This can be achieved by checking the asset variable obtained from the ReadUserData function

Retest

This vulnerability has been fixed by alterUsersEarningRateIndex()

Bug ID #2 [Fixed]

Potential Underflow Vulnerability in Fee Calculation and Asset Freezing

Vulnerability Type
Dos

Severity
High

Description
In the given contract, the executeTrade() function attempts to calculate the fees to be charged to the user using the tradeFee() function. However, the fee calculation in tradeFee() subtracts the fee value from 1e18, which can potentially result in an underflow if the feeType(0) value is greater than the feeType(1) value. This underflow may lead to unexpected behavior or revert the transaction. Additionally, if the transaction reverts due to the underflow, users' assets may become stuck in the contract indefinitely because there is no option to unfreeze the frozen assets. revertTrade() offerce unfreezing of users assets but only for requestTime[requestId] + 1 hours > block.timestamp . after that user can't unfreeze assets.

Affected Code

- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/e7c2aa5efb84008ba0dbb6cf358fd635881e43ce/contracts/executor.sol#L257-L263

Impacts
If the feeType(0) value is greater than the feeType(1) value, the calculation in executeTrade() will result in an underflow, causing the transaction to revert. users assets may become stuck in the contract indefinitely because there is no mechanism to unfreeze the frozen assets.

Remediation

Before performing the fee calculation in executeTrade(), implement bounds checking to ensure that the feeType(0) value is less than or equal to the feeType(1) value. If the condition is not met, handle the scenario appropriately to prevent underflows or inaccuracies in the fee calculation.

Retest

Vulnerability hasb been fixed by applying validation between feeType(0) and feeType(1)

Bug ID #3 [Fixed]

Missing Handling of Fees on Transfer in ERC20 Token Transfers

Vulnerability Type
Calculation Inaccuracy

Severity
High

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/REXProtocol/REX-SmartContract-Testing/blob/96046275d2ad23bda51e86d979e6887b83833e0a/contracts/depositvault.sol#L172
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/96046275d2ad23bda51e86d979e6887b83833e0a/contracts/depositvault.sol#L386

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 fees on every transfer while accounting.

Retest

The vulnerability has been addressed by implementing a check that calculates the balance of the contract before and after the transfer is completed.

Bug ID #4 [Fixed]

Incorrect Conditions in revertTrade Function

Vulnerability Type
Business Logic Flaw

Severity
High

Description
The revertTrade function in the contract contains an incorrect condition that always evaluates to false due to an unset variable. The condition checks if the timestamp of the request creation time plus one hour is greater than the current block timestamp. However, the requestTime variable is not set anywhere in the contract, leading to its default value of zero. As a result, requestTime[requestId] + 1 hour will always be less than the current block timestamp, causing the condition to evaluate to false.

Affected Code
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/Updated_hardhat_env/contracts/Oracle.sol#L49
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/Updated_hardhat_env/contracts/Oracle.sol#L95

Impacts
The revertTrade function falsely reports success without actually reverting any transactions. This misleading behavior can deceive users into believing that their transactions have been reverted when, in reality, they continue to execute.

Remediation
Ensure that the requestTime variable is properly initialized and updated whenever a request is made. Store the timestamp of the request creation time to enable accurate time-based checks in functions such as revertTrade.

Retest

This vulnerability has been fixed by updating requestTime variable when required.

Bug ID #5 [Fixed]

Flawed Logic in revertTrade Function

Vulnerability Type
Business Logic Flaw

Severity
High

Description
The revertTrade function in the contract contains flawed logic that can lead to users' assets becoming stuck in the contract indefinitely. The issue arises from the condition used to determine the amount of assets to add back to users' balances. If a user's assets are zero, the condition assigns zero to the variable MakerbalanceToAdd, resulting in no assets being added back to the user's balance. As a consequence, the user's assets remain stuck in the contract, and the pending balances are not properly adjusted.

Affected Code

- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/d9f15c8a8c99e5965bd003f1bd59d7c5ca1cefdc/contracts/Oracle.sol#L291-L293

Impacts
Users' assets can become stuck in the contract indefinitely due to the flawed logic in the revertTrade function. This can result in users being unable to access their assets or use them for further transactions.

Remediation
Conduct a thorough review of the revertTrade function to identify and correct the flawed logic. Ensure that the conditions used to determine the amount of assets to add back to users' balances are accurate and account for all possible scenarios.

Retest
This vulnerability has been fixed by updating logic of revertTrade().

Bug ID #6 [Fixed]

Lack of Access Control in revertTrade Function

Vulnerability Type
Access Control

Severity
High

Description
The revertTrade function in the contract does not include access control mechanisms to restrict its usage to authorized users. Any user can call this function with a request ID belonging to another user, leading to the unintended reverting of trades initiated by other users.

Affected Code

- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/d9f15c8a8c99e5965bd003f1bd59d7c5ca1cefdc/contracts/Oracle.sol#L91-L121

Impacts
Any user can maliciously or mistakenly revert trades initiated by other users by calling the revertTrade function with their request IDs

Remediation
Ensure that the revertTrade function validates the caller's authorization before proceeding with the trade reversion. Verify that the caller has the necessary permissions to revert trades associated with the specified request ID.

Retest
This vulnerability has been addressed by adding a validation check for msg.sender.

Bug ID#7 [Fixed]

Lack of Access Control in chargeMassinterest() Function

Vulnerability Type
Access Control

Severity
Medium

Description
In the deposit_token() function of the given contract, the contract calls the chargeMassinterest() function of the interestContract contract. However, the chargeMassinterest() function is restricted to the owner only. As a result, any attempt to call chargeMassinterest from the deposit_token() function will revert, as the caller is not the owner of the contract.

Affected Code

- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/ea4b012192bf7e7ec555b041fa8c0d3ed6e751f8/contracts/depositvault.sol#L181
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/ea4b012192bf7e7ec555b041fa8c0d3ed6e751f8/contracts/depositvault.sol#L198

Impacts
The lack of proper access control in the chargeMassinterest function allows any external account to trigger the function, which will result in a transaction revert due to the function's owner-only restriction. This limitation prevents the deposit_token function from successfully executing, potentially disrupting the intended functionality of the contract.

Remediation
Modify the chargeMassinterest function in the interestContract contract to make it publicly accessible or implement a permissioned access control mechanism to allow authorized accounts to call the function.

Retest

This vulnerability has been fixed by making chargeMassinterest() as public function.

Bug ID#8 [Fixed]

Potential Reentrancy Vulnerability in SubmitOrder Function

Vulnerability Type
Reentrancy

Severity
Medium

Description
The SubmitOrder() function in the given contract allows users to submit orders by calling the Airnode contract specified in the airnode_details parameter. However, the function executes a low-level call to the user-provided address, which can result in a reentrancy vulnerability. Additionally, the function transfers the value of msg.value to the specified Airnode contract, which can lead to a loss of funds if not handled properly.

Affected Code

- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/ea4b012192bf7e7ec555b041fa8c0d3ed6e751f8/contracts/executor.sol#L111

Impacts
An attacker could exploit the reentrancy vulnerability by deploying a malicious contract that reverts the state changes after receiving the value from the SubmitOrder function. This could allow the attacker to repeatedly call back into the SubmitOrder function before state changes are finalized, potentially manipulating the state of the contract and causing unexpected behavior.

Remediation
Add a validation check to ensure that the address specified in the airnode_details parameter points to a legitimate Airnode contract. This can help prevent calling arbitrary or malicious contracts. If low level call to external address is not nesassary then remove that low level call.

Retest

This vulnerability hab been fixed by adding validation on airnode address

Bug ID#9 [Fixed]

Missing Functionality and Error Handling in fulfill() Function

Vulnerability Type
Missing Functionality

Severity
Medium

Description
In the fulfill() function of the contract, there is a call to a function named revertTrade() on the Executor contract. However, the revertTrade() function does not exist on the Executor contract, causing the call to always revert. Additionally, there is no fallback() function implemented in the contract to handle this condition gracefully, leading to an abrupt revert of the transaction whenever the condition is met.

Affected Code

- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/ea4b012192bf7e7ec555b041fa8c0d3ed6e751f8/contracts/Oracle.sol#L248-L254

Impacts
The absence of error handling mechanisms, such as a fallback() function, exacerbates the issue by abruptly reverting transactions without providing any meaningful feedback or graceful handling of unexpected conditions.

Remediation
Review the requirements and specifications for the fulfill() function and ensure that all referenced functions, such as revertTrade(), exist and are correctly implemented on the Executor contract. If the revertTrade() function is not intended to be called, remove the corresponding code block from the fulfill() function.

Retest
This vulnerability has been fixed by adding require statement in fulfill()

Bug ID #10 [Fixed]

Function Call Mismatch in SubmitOrder() Function

Vulnerability Type
Function Call Mismatch

Severity
Medium

Description
In the provided contract, the SubmitOrder() function calls Oracle.ProcessTrade() with only 6 parameters, while the actual Oracle.ProcessTrade() function requires 9 parameters. This mismatch in the function call parameters will cause any invocation of SubmitOrder() to fail or revert, as it does not provide the required arguments for Oracle.ProcessTrade().

Vulnerable Code

- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/96046275d2ad23bda51e86d979e6887b83833e0a/contracts/executor.sol#L159-L169

Impacts
The incorrect function call in SubmitOrder() prevents the intended functionality of the contract from being executed properly. This can lead to unexpected behavior and disruptions in the trading process.

Remediation
Modify the SubmitOrder() function to provide the correct parameters when calling Oracle.ProcessTrade(). Ensure that all required parameters are included in the function call to Oracle.ProcessTrade() to avoid transaction reverts and ensure the proper execution of the trading process.

Retest
This vulnerability has been fixed by supplying right inputs to ProcessTrade() internal call.

Bug ID #11 [Fixed]

Funds Stuck in Smart Contract due to Lack of Withdrawal Mechanism

Vulnerability Type
Lock-up Of Assets

Severity
Low

Description
The smart contract contains a receive() function that allows it to accept incoming Ether transfers. However, there is no corresponding function to withdraw or utilize these deposited funds. As a result, any Ether sent to this contract will become permanently trapped, unable to be retrieved or utilized by the contract owner or users. This creates a significant flaw in the functionality of the contract and can lead to financial loss

Affected Code

- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/d9f15c8a8c99e5965bd003f1bd59d7c5ca1cefdc/contracts/Oracle.sol#L304
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/d9f15c8a8c99e5965bd003f1bd59d7c5ca1cefdc/contracts/datahub.sol#L714
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/d9f15c8a8c99e5965bd003f1bd59d7c5ca1cefdc/contracts/depositvault.sol#L444
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/d9f15c8a8c99e5965bd003f1bd59d7c5ca1cefdc/contracts/executor.sol#L513
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/d9f15c8a8c99e5965bd003f1bd59d7c5ca1cefdc/contracts/interestData.sol#L593

Impacts
Without a withdrawal mechanism in place, any Ether sent to the contract becomes permanently trapped, leading to financial loss and frustration for users.

Remediation

It is recommended that either implement a proper withdrawal function that only authorized users can call or remove the receive() function entirely from the smart contract.

Retest

This vulnerability has been fixed by adding withdrawAll() function in contracts

Bug ID #12 [Fixed]

Incomplete Admin Role Revocation in alterAdminRoles() Function

Vulnerability Type
Access Control

Severity
Low

Description
The alterAdminRoles() function assigns admin privileges to new addresses without revoking the privileges from old addresses. This incomplete revocation mechanism allows previous admins to retain their access rights even after new admins are added.

Affected Code

- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/datahub.sol#L42-L56
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/Oracle.sol#L45-L54
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/depositvault.sol#L34-L45
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/executor.sol#L26-L41
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/interestData.sol#L16-L29
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/utils.sol#L14-L29

Impacts
The vulnerability enables unauthorized individuals to maintain administrative control, potentially leading to unauthorized actions, data breaches, or manipulation of contract functionality.

Remediation

To address this vulnerability, modify the alterAdminRoles() function to include a step for revoking admin privileges from existing addresses before assigning them to new addresses.

Retest

This vulnerability has been addressed by adding the revokeAdminRole() function.

Bug ID #13 [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., 0.8.20. 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
- Every Contract

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 really low therefore this is only informational.

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

This issue has been fixed by replacing the pragma version to ‘=0.8.23’ instead of ‘=0.8.20’

Bug ID #14 [Fixed]

Use Ownable2Step

Vulnerability Type
Missing Best Practices

Severity
Low

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

Affected Code

- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/ProxyAdmin.sol#L13
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/Oracle.sol#L11
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/datahub.sol#L13
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/depositvault.sol#L12
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/executor.sol#L13
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/interestData.sol#L10
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/liquidator.sol#L12

- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/utils.sol#L12

Impacts

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

Remediation

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

Retest:

Ownable is now replaced with Ownable2Step.

Bug ID #15 [Not 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 -

- Nearly Every function with the checkRoleAuthority and onlyOwner Modifier.

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
Events has not been added in above mentioned instances

Bug ID #16 [Partially Fixed]

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://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/depositvault.sol#L61
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/depositvault.sol#66
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/liquidator.sol#L35
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/liquidator.sol#L59
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/liquidator.sol#L61

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
Few of them are fixed are few are not fixed.

Bug ID#17 [Fixed]

Hardcoded Static Address

Vulnerability Type
Missing Best Practices

Severity
Informational

Description
The contract was found to be using hardcoded addresses.
This could have been optimized using dynamic address update techniques along with proper access control to aid in address upgrade at a later stage.

Affected Code

- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/Oracle.sol#L21
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/depositvault.sol#L78

Impacts
Hardcoding address variables in the contract make it difficult for it to be modified at a later stage in the contract as everything will need to be deployed again at a different address if there's a code upgrade.

Remediation
It is recommended to create dynamic functions to address upgrades so that it becomes easier for developers to make changes at a later stage if necessary.
The said function should have proper access controls to make sure only administrators can call that function using access control modifiers.
There should also be a zero address validation in the function to make sure the tokens are not lost.

If the address is supposed to be hardcoded, it is advisable to make it a constant if its value is not getting updated.

Retest
The hard-coded addresses has been removed.

Bug ID #18 [Not Fixed]

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://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/datahub.sol#L498
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/depositvault.sol#L172
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/depositvault.sol#L173
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/depositvault.sol#L259
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/depositvault.sol#L323
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/executor.sol#L131
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/executor.sol#L137
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/executor.sol#L190

- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/liquidator.sol#L101-L105
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/liquidator.sol#L182

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 **require** statement to give more detail to the user about why the condition failed.

Retest

Messages has not been added in require statement

Bug ID #19 [Fixed]

Dead Code

Vulnerability Type
Code With No Effects - SWC-135

Severity
Gas

Description
The provided code contains dead code, specifically, the calculation involving the subtraction and multiplication of amountToAddToLiabilities with the result of Datahub.tradeFee(out_token, 1). This calculation appears to have no effect on the final result of the function and can be considered unnecessary.

Vulnerable Code

- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/96046275d2ad23bda51e86d979e6887b83833e0a/contracts/executor.sol#L244-L248

Impacts
Having dead and unused code in the contract leads to excessive gas usage when deploying on production chains. This could also mean that critical access control features have not been implemented properly.

Remediation
Remove the dead code snippet involving the calculation of amountToAddToLiabilities with Datahub.tradeFee(out_token, 1). Since this calculation does not affect the outcome of the function, its removal will streamline the code and improve gas efficiency

Retest
This issue has been fixed by removing the dead code line.

Bug ID #20 [Not Fixed]

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://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/utils.sol#L61
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/liquidator.sol#L17

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

Variables are not marked as Immutable.

Bug ID #21 [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/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/datahub.sol#L324
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/datahub.sol#L421
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/datahub.sol#L338
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/Oracle.sol#L204
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/depositvault.sol#L108
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/depositvault.sol#L168
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/depositvault.sol#L261
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/depositvault.sol#L381
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/executor.sol#L131

- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/executor.sol#L190
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/executor.sol#L237
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/executor.sol#L296
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/executor.sol#L405
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/liquidator.sol#L182
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/liquidator.sol#L363

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 fixed as recommended

Bug ID #22 [Not Fixed]

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/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/liquidator.sol#L15
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/liquidator.sol#L19
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/liquidator.sol#L21

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
Two variables have been removed, and one is still the same are before.

Bug ID #23 [Fixed]

Cheaper Conditional Operators

Vulnerability Type
Gas Optimization

Severity
Gas

Description
Upon reviewing the code, it has been observed that the contract uses conditional statements involving comparisons with unsigned integer variables. Specifically, the contract employs the conditional operators x != 0 and x > 0 interchangeably. However, it's important to note that during compilation, x != 0 is generally more cost-effective than x > 0 for unsigned integers within conditional statements.

Affected Code

- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/libraries/EVO_LIBRARY.sol#L291
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/depositvault.sol#L197
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/depositvault.sol#L341
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/depositvault.sol#L400

Impacts
Employing x != 0 in conditional statements can result in reduced gas consumption compared to using x > 0. This optimization contributes to cost-effectiveness in contract interactions.

Remediation

Whenever possible, use the x != 0 conditional operator instead of x > 0 for unsigned integer variables in conditional statements.

Retest

This issue has been fixed according to the remediation.

Bug ID #24 [Fixed]

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://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/ProxyAdmin.sol#L38C5-L44C6
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/depositvault.sol#L70C4-L72C6
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/executor.sol#L49C5-L61C6
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/executor.sol#L106C5-L108C6
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/executor.sol#L110C4-L112C6
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/executor.sol#L122C5-L170C6
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/utils.sol#L36C4-L46C6
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/liquidator.sol#L63-L171

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

Public functions are marked as external

Bug ID #25 [Partially Fixed]

Gas Optimization in Require Statements

Vulnerability Type
Gas Optimization

Severity
Gas

Description
The **require()** statement takes an input string to show errors if the validation fails.
The strings inside these functions that are longer than **32 bytes** require at least one additional MSTORE, along with additional overhead for computing memory offset and other parameters. For this purpose, having strings lesser than 32 bytes saves a significant amount of gas.

Vulnerable Code

- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/depositvault.sol#L167-L170
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/depositvault.sol#L260-L263
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/depositvault.sol#L272-L275
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/depositvault.sol#L276-L279
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/depositvault.sol#L281-L285
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/depositvault.sol#L380-L383
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/executor.sol#L149-L152
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/executor.sol#L154-L157

- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/liquidator.sol#L78-L81
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/liquidator.sol#L84-L94
- https://github.com/REXProtocol/REX-SmartContract-Testing/blob/061cac3641f66f09eb364e0290dd28959616d124/contracts/liquidator.sol#L96-L99

Impacts

Having longer require strings than **32 bytes** costs a significant amount of gas.

Remediation

It is recommended to shorten the strings passed inside **require()** statements to fit under **32 bytes**. This will decrease the gas usage at the time of deployment and at runtime when the validation condition is met.

Retest

Few of them are fixed are few are not fixed.

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.