

AUDITOR LABS

Risedle

Leverage Protocol

SMART CONTRACT AUDIT

Made by auditorlabs.com

Table of contents

1. Disclaimer.....	3
2. About the Project and Company	4
2.1 Project Overview.....	5
3. Vulnerability & Risk Level	6
4. Auditing Strategy and Techniques Applied.....	7
4.1 Methodology	7
4.2 Used Code from other Frameworks/Smart Contracts	8
4.3 Tested Contract Files	9
4.4 Metrics / CallGraph.....	10
4.5 Metrics / Source Lines & Risk.....	13
4.6 Metrics / Capabilities	14
4.7 Metrics / Source Unites in Scope	15
5. Scope of Work	17
5.1 Manual and Automated Vulnerability Test.....	18
5.1.1 Overpowered Owner rights.....	18
5.1.2 Use arrays for gas efficiency	19
5.1.3 Variables that can be made constant.....	20
5.1.4 Public functions should be declared as external	21
5.1.5 State visibility is not set	22
5.1.6 Floating compiler version	22
5.1.7 ABIEncoder v2	23
5.1.8 Unused imports.....	23

5.2 SWC Attacks	24
5.3 Verify claims	28
6. Executive Summary.....	30
7. Deployed Smart Contract	30

1. Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Risedle Labs Ltd. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (11.01.2022)	Layout
0.2 (13.01.2022)	Test Deployment
0.5 (15.01.2022)	Automated Security Testing Manual Security Testing
0.6 (19.01.2022)	Testing SWC Checks
0.7 (18.01.2022)	Verify Claims
0.9 (20.01.2022)	Summary and Recommendation
1.0 (24.01.2022)	Final document
1.1 (02.02.2022)	Added deployed contract addresses

2. About the Project and Company

Company address:

Risedle Labs Ltd
BVI Company Number 2084175
Intershore Chambers, PO Box 4342
Road Town, Tortola, VG1110
British Virgin Islands

Website: <https://risedle.com>

Twitter: <https://twitter.com/risedle>

Discord: <https://discord.gg/YCSCd97SXj>

Github: <https://github.com/risedle>

Documentation: <https://docs.risedle.com>



2.1 Project Overview

The Risedle Leveraged Token protocol is a peer-to-peer system designed for leveraging any cryptocurrency assets (ERC-20 Token) on the Ethereum Virtual Machine (EVM) compatible or equivalence blockchain.

There are two components of Risedle Leveraged Tokens protocol:

1. **Risedle Vault:** The isolated lending & borrowing protocol specially designed for Leveraged Tokens.
2. **Risedle Leveraged Tokens:** ERC-20 token that represent leveraged position of underlying asset.

The protocol is implemented as a set of persistent, non-upgradable smart contracts; designed to prioritize censorship resistance, security, self-custody, and to function without any trusted intermediaries who may selectively restrict access.

There are currently only one version of the Risedle Leveraged Token protocol. Risedle Protocol V1 are open source and licensed under GPL. Once deployed, will function in perpetuity, with 100% uptime, provided the continued existence of the EVM compatible or equivalence blockchain.

Risedle Leveraged Token protocol is the first protocol that create lending and borrowing specially designed for Leveraged Tokens. These are the benefits of designing lending & borrowing protocol for leveraged tokens:

1. **Risk Isolation:** Anyone can create leveraged tokens market pair (vault as the supply & leveraged tokens as the demand) without worrying to add more risk to existing leveraged tokens market pair.
2. **No Liquidation:** Risedle Leveraged Protocol does not have liquidation bot, it have rebalancing bot instead. The user who borrow asset from the vault by minting the leveraged tokens will never get liquidated.
3. **Hands-free Borrowing Experience:** There is no such thing as health ratio, loan to value, or other complex stuff. The user just need hold the leveraged tokens to get leverage exposure.

3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
 - i. Review of the specifications, sources, and instructions provided to Auditor Labs to make sure we understand the size, scope, and functionality of the smart contract.
 - ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
 - iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Auditor Labs describe.
2. Testing and automated analysis that includes the following:
 - i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
 - ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

4.2 Used Code from other Frameworks/Smart Contracts (direct imports)

Dependency / Import Path	Source
lib/openzeppelin-contracts/contracts/access/Ownable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.4.1/contracts/access/Ownable.sol
lib/openzeppelin-contracts/contracts/security/ReentrancyGuard.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.4.1/contracts/security/ReentrancyGuard.sol
lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.4.1/contracts/token/ERC20
lib/openzeppelin-contracts/contracts/token/ERC20/IERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.4.1/contracts/token/ERC20/IERC20.sol
lib/openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Metadata.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.4.1/contracts/token/ERC20/extensions/IERC20Metadata.sol
lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.4.1/contracts/token/ERC20/utils/SafeERC20.sol

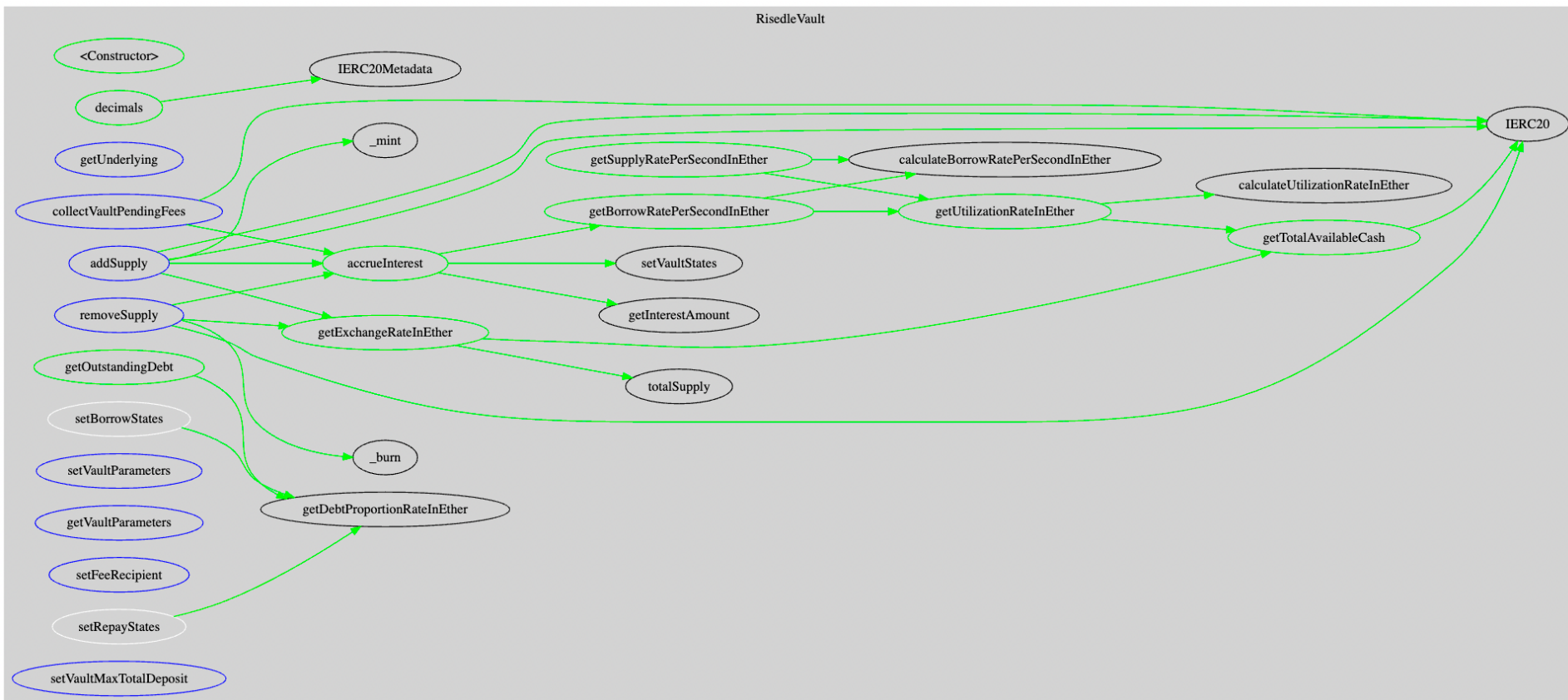
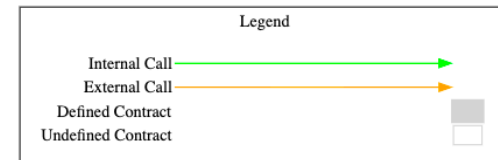
4.3 Tested Contract Files

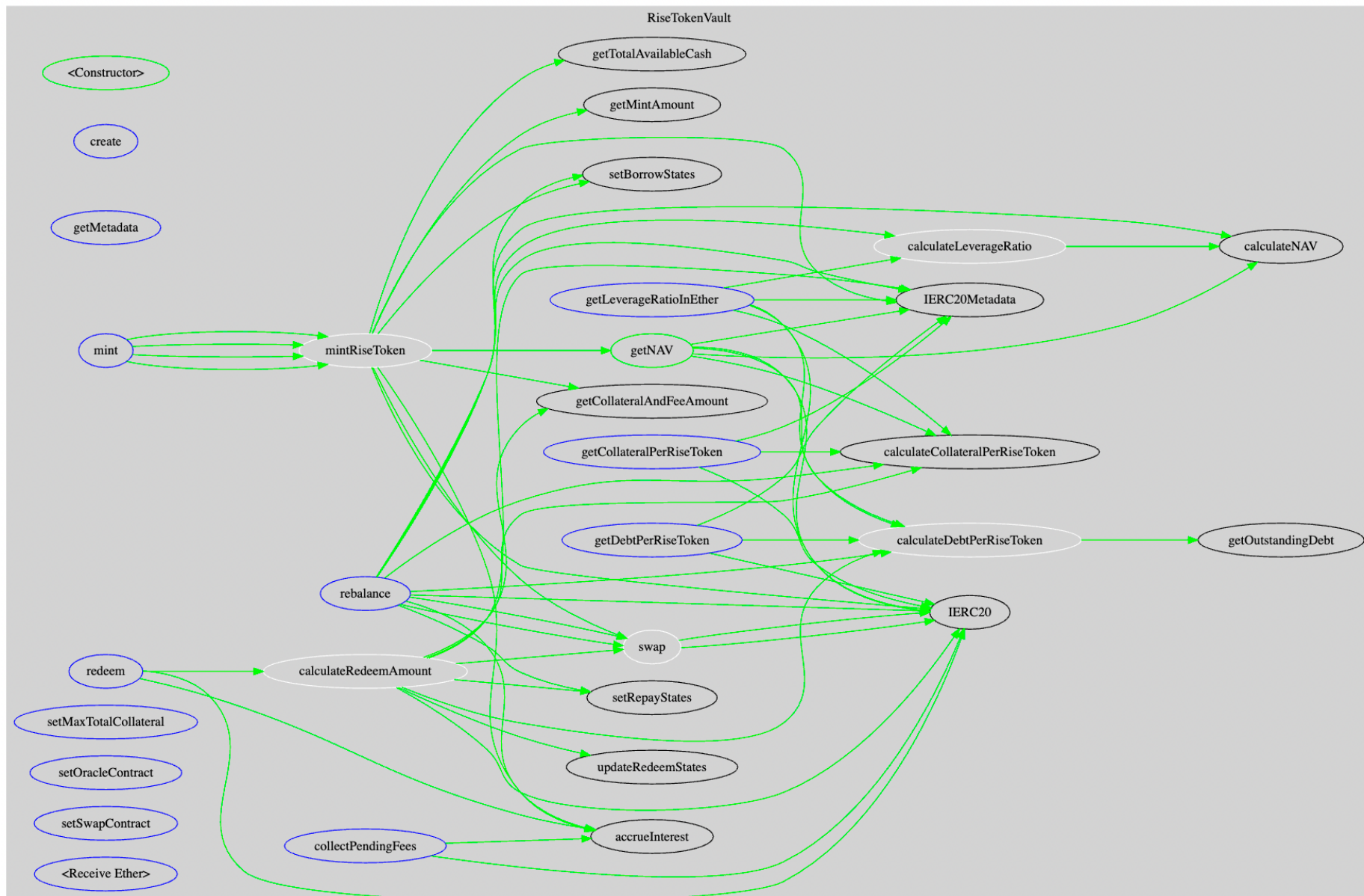
The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

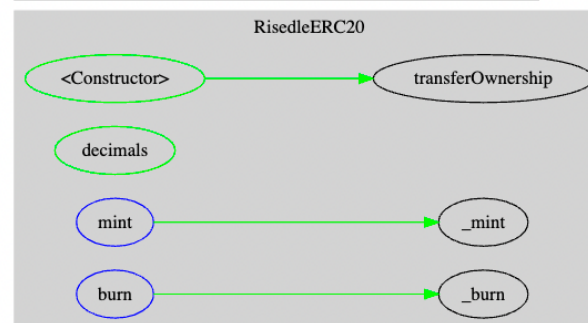
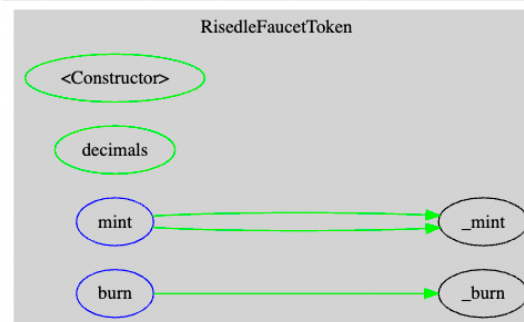
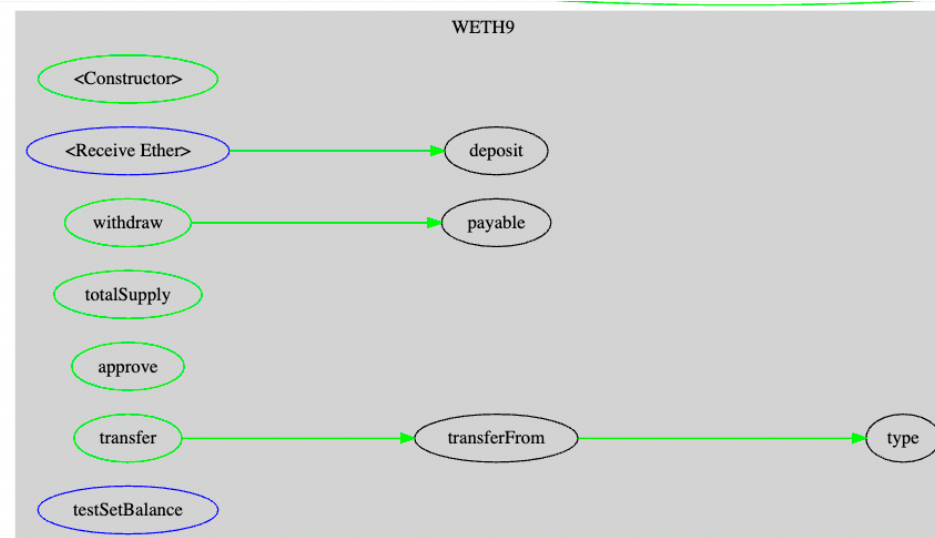
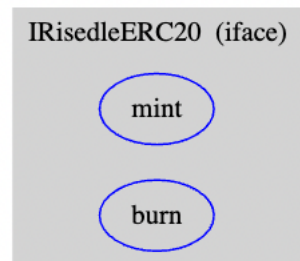
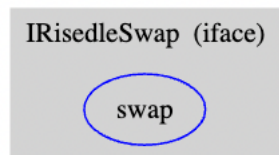
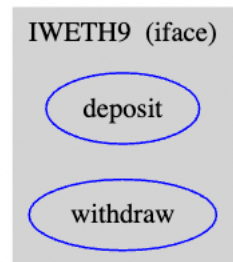
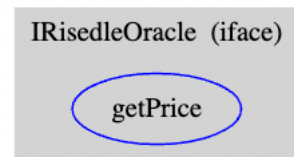
File	Fingerprint (MD5)
./src/interfaces/IRisedleERC20.sol	aa95f882969146593b990a6638a9ffd9
./src/interfaces/IRisedleSwap.sol	cf72be39d60d01fd915baf4ce73a6f5
./src/interfaces/IWETH9.sol	fb7e002524c547858fae88a6e3095d19
./src/interfaces/IRisedleOracle.sol	cfeec92acea714c96e31aa522b1db476
./src/tokens/RisedleFaucetToken.sol	15eee1dfd04bccc41219a2e878934c8
./src/tokens/WETH9.sol	360b244bea8ef24c8abfdb0ef8c1215a
./src/RiseTokenVault.sol	39b8459d290ac4e10deb1bf8967bb08b
./src/RisedleVault.sol	0691955edd16bd4a2817f6a41e022be0

Source: <https://github.com/risedle/etfs/tree/v1.0.0-audit-2021-jan-rev-1> (commit cab10fbeb9afa4cd81314a9e51ae5dff440cd88c)

4.4 Metrics / CallGraph

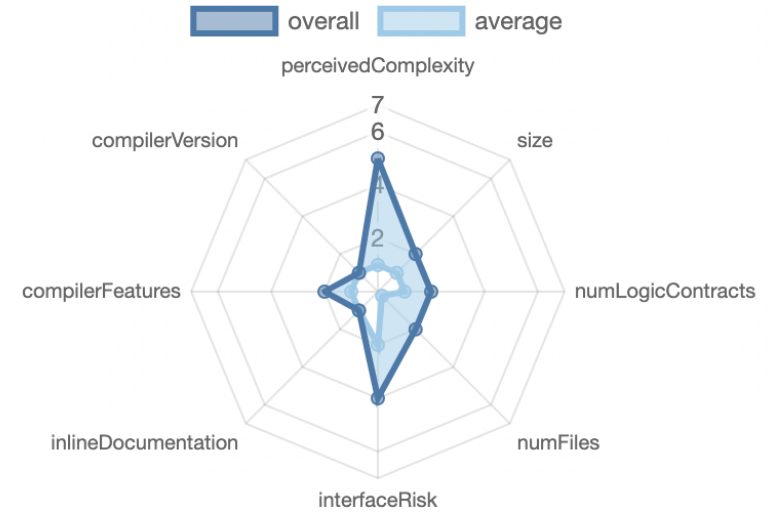
















4.5 Metrics / Source Lines & Risk

source comment single block mixed
empty todo blockEmpty





4.6 Metrics / Capabilities


Solidity Versions observed	 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts	
<div>>=0.8.9</div> <div>>=0.6.0</div>	<div>ABIEncoderV2</div>	<div>yes</div>	<div></div>	<div></div>	
 Transfers ETH	 Low-Level Calls	 DelegateCall	 Uses Hash Functions	 ECRecover	 New/Create/Create2
<div>yes</div>	<div></div>	<div></div>	<div></div>	<div></div>	<div></div>

Exposed Functions

















This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.





 Public	 Payable			
54	6			
External	Internal	Private	Pure	View
37	59	0	7	21

StateVariables

Total	 Public
25	9

4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	etfs-1.0.0-audit-2021-jan-rev-1/src/interfaces/IRisedleERC20.sol	_____	1	10	7	4	1	5	
	etfs-1.0.0-audit-2021-jan-rev-1/src/interfaces/IRisedleSwap.sol	_____	1	21	15	4	9	3	
	etfs-1.0.0-audit-2021-jan-rev-1/src/interfaces/IWETH9.sol	_____	1	15	11	5	5	10	
	etfs-1.0.0-audit-2021-jan-rev-1/src/interfaces/IRisedleOracle.sol	_____	1	10	9	4	3	3	
	etfs-1.0.0-audit-2021-jan-rev-1/src/tokens/RisedleERC20.sol	1	_____	58	58	25	24	18	
	etfs-1.0.0-audit-2021-jan-rev-1/src/tokens/RisedleFaucetToken.sol	1	_____	49	49	32	7	23	
	etfs-1.0.0-audit-2021-jan-rev-1/src/tokens/WETH9.sol	1	_____	86	82	58	7	44	
	etfs-1.0.0-audit-2021-jan-rev-1/src/RiseTokenVault.sol	1	_____	422	363	281	159	200	

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	etfs-1.0.0-audit-2021-jan-rev-1/src/RisedleVault.sol	1	_____	338	318	216	110	118	
	Totals	5	4	1009	912	629	325	424	

Legend: [—]

- **Lines**: total lines of the source unit
- **nLines**: normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC**: normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines**: lines containing single or block comments
- **Complexity Score**: a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls,

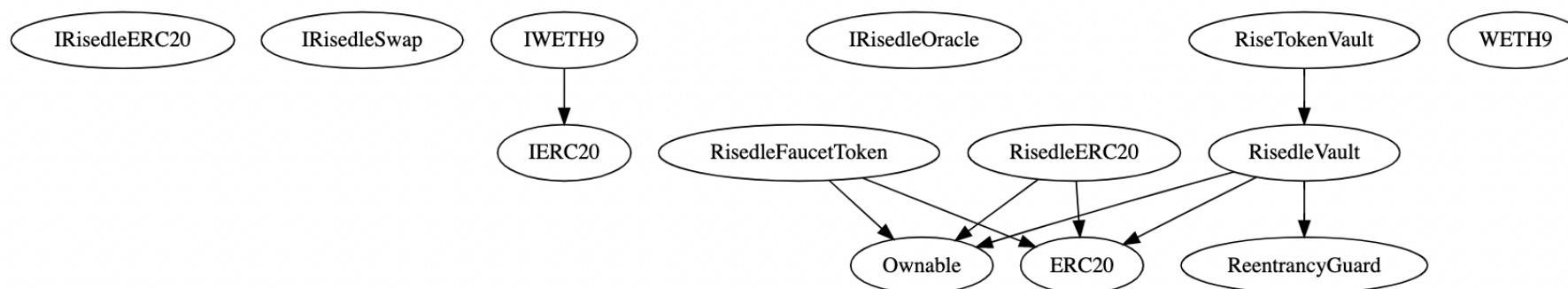
5. Scope of Work

The Risedle Team provided us with the files that needs to be tested. The scope of the audit are the Leveraged Token Protocol contracts.

The team put forward the following assumptions regarding the security, usage of the contracts:

- The smart contract is coded according to the newest standards and in a secure way
- Risedle Leveraged Token Protocol cannot be effected to hacks which happened to other lending platforms such as Celsius, Cream Finance or bzX

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.



5.1 Manual and Automated Vulnerability Test

CRITICAL ISSUES

During the audit, experts found **no Critical issues** in the code of the smart contract.

HIGH ISSUES

During the audit, experts found **no High issues** in the code of the smart contract.

MEDIUM ISSUES

5.1.1 Overpowered Owner rights

Severity: MEDIUM

Status: ACKNOWLEDGED

File(s) affected: RisedleVault.sol, RiseTokenVault.sol

Attack / Description	Code Snippet	Result/Recommendation
Owners can perform privileged activities like minting and burning tokens. The auditor has not recognized any multi-sig structure.	NA	It is recommended to use multisig for owner privileges. Additionally prevent renounceOwnership from being called without the owner transferring ownership to new owner.

LOW ISSUES

5.1.2 Use arrays for gas efficiency

Severity: LOW

Status: ACKNOWLEDGED

File(s) affected: RiseTokenVault.sol

Attack / Description	Code Snippet	Result/Recommendation
A function with many arguments can be less efficient.	<pre>Line 61 function create(bool isETH, // True if the collateral is ETH address tokenRiseAddress, // ERC20 token address that only RiseTokenVault can mint and burn address collateral, // The underlying token of TOKENRISE (e.g. WBTC), it's WETH if the isETH is true address oracleContract, // Contract address that implement IRisedleOracle interface address swapContract, // Uniswap V3 like token swapper uint256 maxSwapSlippageInEther, // Maximum slippage when mint, redeem and rebalancing (1% is 0.01 ether or 0.01*1e18) uint256 initialPrice, // Initial price of the TOKENRISE based on the Vault's underlying asset (e.g. 100 USDC => 100 * 1e6) uint256 feeInEther, // Creation and redemption fee in ether units (e.g. 0.001 ether = 0.1%)</pre>	We recommend to put uints and addresses in arrays, which is more gas efficient.

	<pre> uint256 minLeverageRatioInEther, // Minimum leverage ratio in ether units (e.g. 2x is 2 ether = 2*1e18) uint256 maxLeverageRatioInEther, // Maximum leverage ratio in ether units (e.g. 3x is 3 ether = 3*1e18) uint256 maxRebalancingValue, // The maximum value of buy/sell when rebalancing (e.g. 500K USDC is 500000 * 1e6) uint256 rebalancingStepInEther // The rebalancing step in ether units (e.g. 0.2 is 0.2 ether or 0.2 * 1e18) </pre>	
--	--	--

5.1.3 Variables that can be made constant

Severity: LOW

Status: ACKNOWLEDGED

File(s) affected: RisedleVault.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation several variables already declared and initialized can be made constant with UPPER_CASE_UNDERSCORE naming convention.	<pre> uint256 internal optimalUtilizationRateInEther = 0.9 ether; (RisedleVault.sol line 26) uint256 internal interestSlope1InEther = 0.2 ether; ether (RisedleVault.sol line 28) uint256 internal interestSlope2InEther = 0.6 ether; ether (RisedleVault.sol line 30) uint256 internal immutable totalSecondsInAYear = 31536000; ether (RisedleVault.sol line 32) uint256 internal maxBorrowRatePerSecondInEther = 50735667174; ether (RisedleVault.sol line 32) </pre>	It is recommended to make initialized variables constant to save on gas.

	uint256 internal performanceFeeInEther = 0.1 ether; ether (RisedleVault.sol line 30)	
--	--	--

5.1.4 Public functions should be declared as external

Severity: LOW

Status: ACKNOWLEDGED

File(s) affected: RisedleVault.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation several functions are declared as public where they could be external. For public functions Solidity immediately copies array arguments to memory, while external functions can read directly from calldata. Because memory allocation is expensive, the gas consumption of public functions is higher.	RisedleVault.decimals()	We recommend declaring functions as external if they are not used internally. This leads to lower gas consumption and better code readability.

5.1.5 State visibility is not set

Severity: LOW

Status: ACKNOWLEDGED

File(s) affected: RisedleTokenVault.sol

Attack / Description	Code Snippet	Result/Recommendation
State variables without visibility set.	mapping(address => RiseTokenMetadata) riseTokens;(RiseTokenVault.sol line 42)	State variable must be declared internal, private or public.

INFORMATIONAL

5.1.6 Floating compiler version

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

File(s) affected: All

Attack / Description	Code Snippet	Result/Recommendation
The current pragma solidity directive is floating. It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.	pragma solidity >=0.8.9;	It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee. i.e. Pragma solidity 0.8.9 See SWC-103: https://swcregistry.io/docs/SWC-103

5.1.7 ABIEncoder v2

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

File(s) affected: RiseTokenVault.sol

Attack / Description	Code Snippet	Result/Recommendation
Solidity 0.8.0 has ABI Encoder V2 activated by default. You can activate the old encoder using pragma abicoder v1, or explicitly select v2 using pragma abicoder v2 which has the same effect as pragam experimental ABIEncoderV2. ABI coder v2 is more complex than v1 but also performs additional checks on the input and supports a larger set of types than v1	Line 4 pragma experimental ABIEncoderV2;	ABIEnocoderV@ is activated by default since 0.8.0 and can be removed.

5.1.8 Unused imports

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

File(s) affected: RiseTokenVault.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation this contract has an unused imported file.	Line 10 import { RisedleERC20 } from "./tokens/RisedleERC20.sol";	Remove unused imports to reduce contract size.

5.2 SWC Attacks

ID	Title	Relationships	Test Result
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	✓
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	✓
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	✓
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	✓
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	✓
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	✓
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	✓
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	✓

ID	Title	Relationships	Test Result
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	✓
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	✓
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	✓
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	✓
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	✓
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	✓
SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✓
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	✓
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	✓


ID	Title	Relationships	Test Result
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	✓
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✓
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	✓
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	✓
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	✓
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	✗
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	✓
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	✓
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	✓
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	✓

ID	Title	Relationships	Test Result
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	✗
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	✓
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	✓
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	✓

5.3 Verify claims

Risedle Leveraged Token Protocol cannot be effected to hacks which happened to other lending platforms such as Celsius, Cream Finance or bzX

5.3.1 The smart contract is coded according to the newest standards and in a secure way

Status: tested and verified 

5.3.2 Risedle Leveraged Token Protocol cannot be effected to hacks which happened to other lending platforms such as Celsius, Cream Finance or bzX

Status:

1. Celsius :

Celsius network was effected by the BadgerDAO hack in december 2021. This hack was a front-end attack, and the smart contracts were untouched. A hacker was able steal an active API key of cloudflare, a content delivery network. With the help of the API key, the attacker injected malicious front-end code where the user was asked to approve a malicious contract address to spend tokens. The approved contract drained all approved tokens out of the user's wallet.

This hack can potentially happen to other dApps like Risedle when traders are asked for approval on trades. Users commonly do not double check the approved address and just allow the spending of tokens. To avoid this scenario, it is important to keep the front-end as safe as the contracts. It is recommended to update API keys and access keys like SSH keys regularly and keep them safe.

2. Cream Finance:

In the cream hack in October 2021 \$130m assets of the lending platform were stolen. The smart contracts have been outplayed with a flash loan attack. A borrower flash borrowed funds from Maker and borrowed loan tokens from cream vault multiple times. He manipulated the prices by burning the tokens.

This attack can happen to the Risedle protocol. Risedle can mitigate that risk with disabling multiple borrow transaction for one account (ex. within a block) .

3. bzX:

On the latest bzX hack, the private key of the deploying account was compromised by an attacker. An employer stored the mnemonic of the deployer wallet locally on his machine and received an email with a word document. The file ran a malicious macro and stole the mnemonic. With the privilege and funds of the deployer wallet, the attacker could drain out funds from the protocol by using the protocols tokens.

This attack could potentially happen to the Risedle platform as well. To prevent this scenario, it is recommended to have a security guideline that ensures the safe storage of private keys. Keys should never be stored on a machine that is used for network interactions such as customer requests. It is recommended to store the privileged private key securely offline. Other options can be multisig Wallets, such as Gnosis Safe.

6. Executive Summary

Two (2) independent Auditor Labs experts performed an unbiased and isolated audit of the smart contract codebase. The final debriefs took place on the January 24, 2021.

The main goal of the audit was to verify the claims regarding the security of the smart contract. During the audit, critical and high issues were found, after the manual and automated security testing and not all claims have been successfully verified. Please check all issues and get back to your auditor if issues have been fixed.

7. Deployed Smart Contract

VERIFIED

<https://arbiscan.io/address/0xf7edb240dbf7bbed7d321776afe87d1fbcfd0a94#code>

<https://arbiscan.io/address/0x46d06cf8052ea6fdbf71736af33ed23686ea1452#code>