EverRise

Protocol
Token & Staking v3

SMART CONTRACT AUDIT

03.04.2022

Made in Germany by Chainsulting.de

Table of contents

1. Disclaimer

The audit makes no statements or warrantees 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 EVERRISE PTE. 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 (25.03.2022)	Layout
0.4 (25.03.2022)	Automated Security Testing Manual Security Testing
0.5 (25.03.2022)	Verify Claims and Test Deployment
0.6 (26.03.2022)	Testing SWC Checks
0.9 (26.03.2022)	Summary and Recommendation
1.0 (26.03.2022)	Final document
1.1 (31.03.2022)	Re-check (1a7bf3e883143ef953793ae1deee2527d295be5a)
1.2 (03.04.2022)	Deployed Contract

2. About the Project and Company

EVERRISE PTE. LTD.
1 SCOTTS ROAD #24-10
SHAW CENTRE
SINGAPORE 228208

Website: https://www.everrise.com/

Twitter: https://twitter.com/EverRise

Discord: https://discord.com/invite/everrise

LinkedIn: https://www.linkedin.com/company/everrise-pte-ltd/about/

Telegram: https://t.me/everriseofficial

YouTube: https://www.youtube.com/channel/UCCDMjFJUr9OvV03I3wNX7lw

Instagram: https://www.instagram.com/everrisetoken

Facebook: https://www.facebook.com/EverRiseToken

Reddit: https://www.reddit.com/r/EverRise

2.1 Project Overview

Everrise offers multi chain solutions with an ecosystem of dApps on Ethereum, Binance Smart Chain, Polygon, Avalanche and Fantom networks. Ecosystem includes bridge, wallets, token, staking, swapping.

EverRise token is the keystone in the EverRise Ecosystem of dApps and the overarching key that unlocks multi-blockchain unification via the EverBridge. EverRise token transactions have 6% buyback and business development fees are collected 4% for token Buyback from the market, with bought back tokens directly distributed as ve-staking rewards 2% for Business Development (Development, Sustainability and Marketing).

EverRise Staking NFTs are Vote Escrowed. EverRise weighted governance tokens which generate rewards with a market driven yield curve, based on the transaction volume of EverRise trades and veEverRise sales. On sales of veEverRise Staking NFTs a 10% royalty fee is collected 6% for token Buyback from the market,with bought back tokens directly distributed as ve-staking rewards 4% for Business Development (Development, Sustainability and Marketing).

EverOwn is a dApp that allows developers and project owners to hand over ownership of a contract to their community rather than renouncing ownership. Renouncing ownership of a contract limits the growth of the ecosystem as it decreases the flexibility of the project. With EverOwn developers and project owners are able to empower their community, and still have the flexibility to improve and fix their contract.

EverWallet is a secure and decentralized vault residing on the blockchain and acts as a security protocol to enhance the security of pre-existing wallets. EverSwap allows the users to purchase, sell and transfer tokens available on PancakeSwap directly to and from EverWallet.

EverSale is a launchpad powered by the EverRise ecosystem for any BSC or ETH token presales. Unlike other platforms, EverSale will not be collecting tokens from projects using the launch pad as a fee, ensuring that projects are protected from presale platform dumps. Adding liquidity to a new token is crucial to ensure the project's development as it is one of the main revenue streams for an ecosystem. With EverLock, developers will be able to lock their initial liquidity and build trust with their community as the community will have the ability to vote for unlocking initial liquidity.

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 Chainsulting 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 Chainsulting 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 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)
EverRise_flat.sol	cd311a86b2871333877f6a77d7cbf25a
nftEverRise_flat.sol	093f8006990670f151067ad52e8e184f

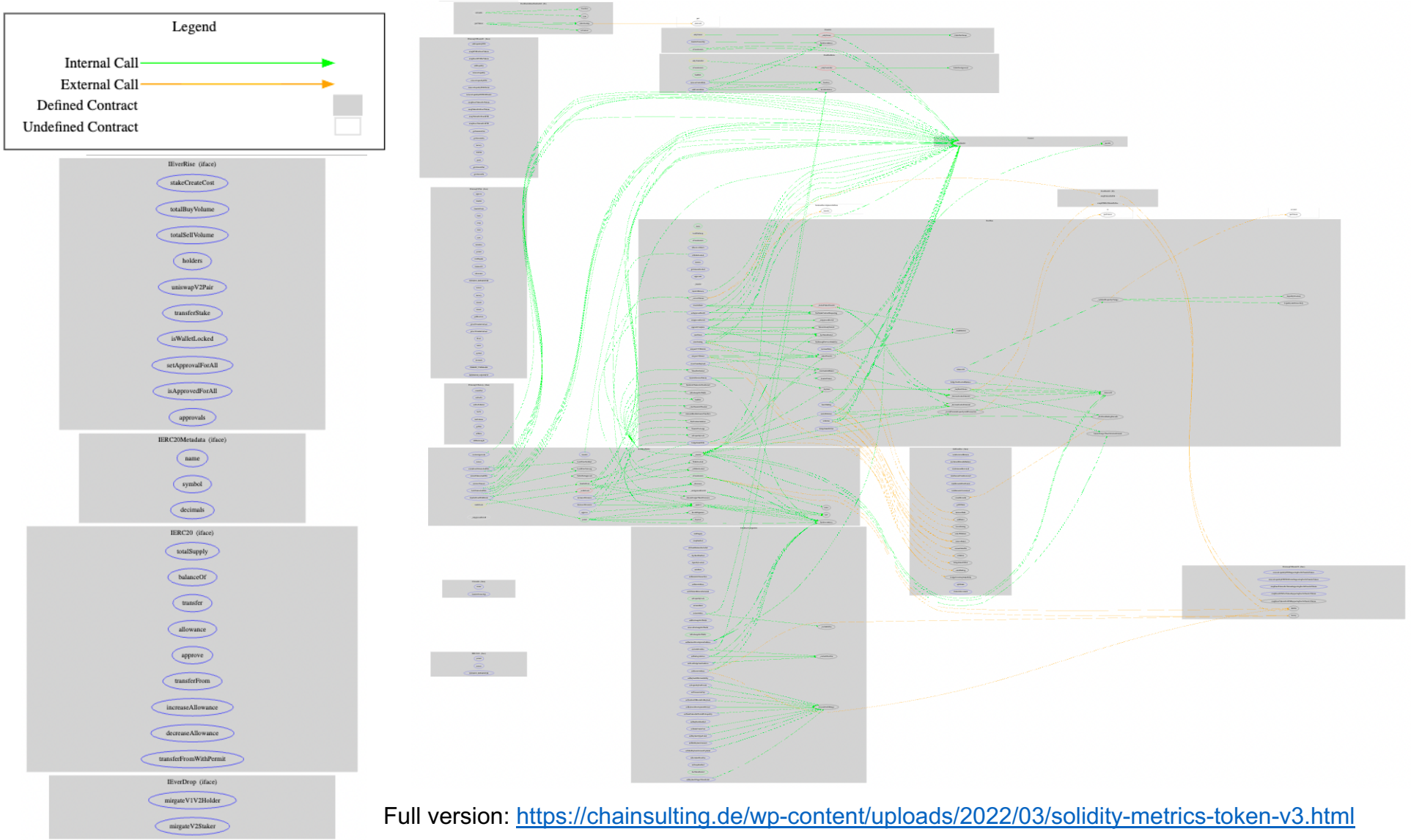
Language	Solidity
Token Standards	ERC20 / ERC721
Compiler Version	0.8.13
Buy Back Token	Yes
Staking Token	Yes
Burn Function	Yes
Mint	Yes
Cross Chain token	Yes

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

Dependency / Import Path	Source
@openzeppelin/contracts/token/ERC20/IERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.5/contracts/token/ERC20/IERC20.sol
@openzeppelin/contracts/token/ERC20/extensions/IERCMetadata.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.5/contracts/token/ERC20/extensions/IERC20Metadata.sol
@openzeppelin/contracts/access/Ownable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.5/contracts/access/Ownable.sol
@openzeppelin/contracts/interfaces/draft-IERC2612.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.5/contracts/interfaces/draft-IERC2612.sol
IUniswapv2Router02.sol	https://github.com/Uniswap/v2-periphery/blob/master/contracts/interfaces/IUniswapV2Router02.sol
@openzeppelin/contracts/interfaces/ERC721/IERC721.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.5/contracts/token/ERC721/IERC721.sol
@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.5/contracts/token/ERC721/extensions/IERC721Metadata.sol

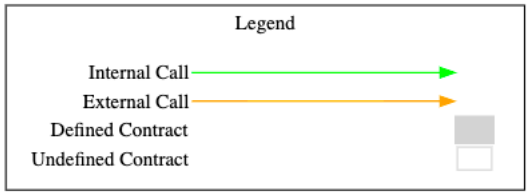
Dependency / Import Path	Source
@openzeppelin/contracts/contracts/utils/introspection/ERC165.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.5/contracts/utils/introspection/ERC165.sol
@openzeppelin/contracts/interfaces/IERC2981.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.5/contracts/interfaces/IERC2981.sol

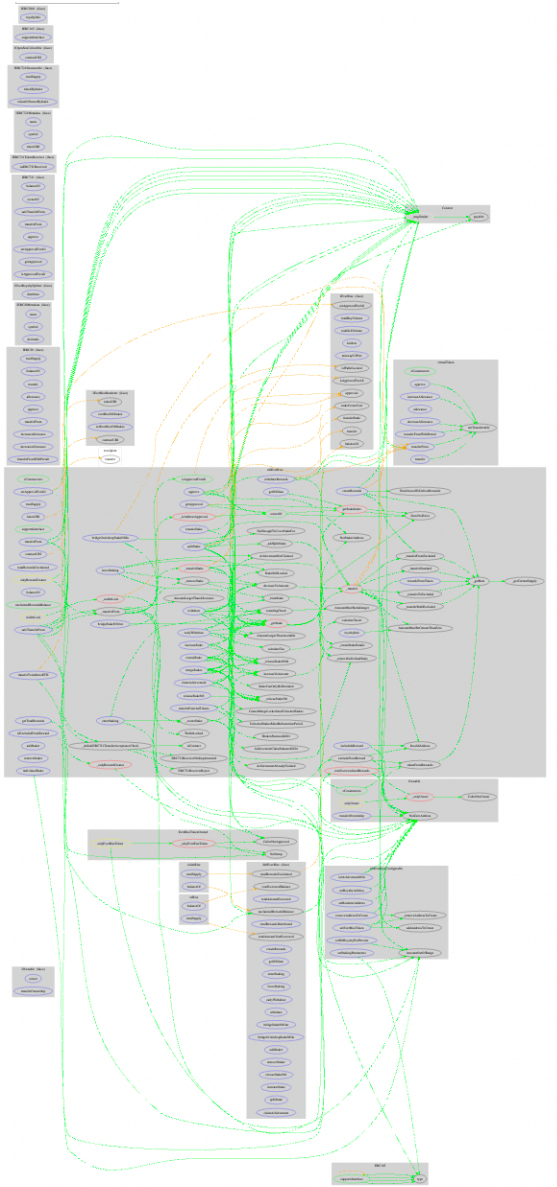
4.4 Metrics / CallGraph (Token v3)



Full version: https://chainsulting.de/wp-content/uploads/2022/03/solidity-metrics-token-v3.html

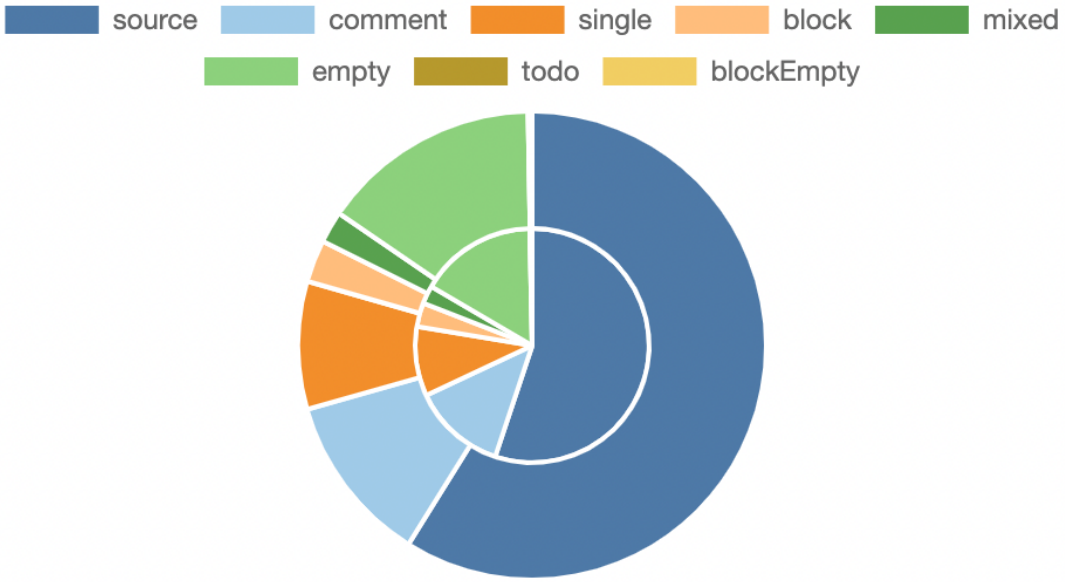
4.4.1 Metrics / CallGraph (Staking v3)

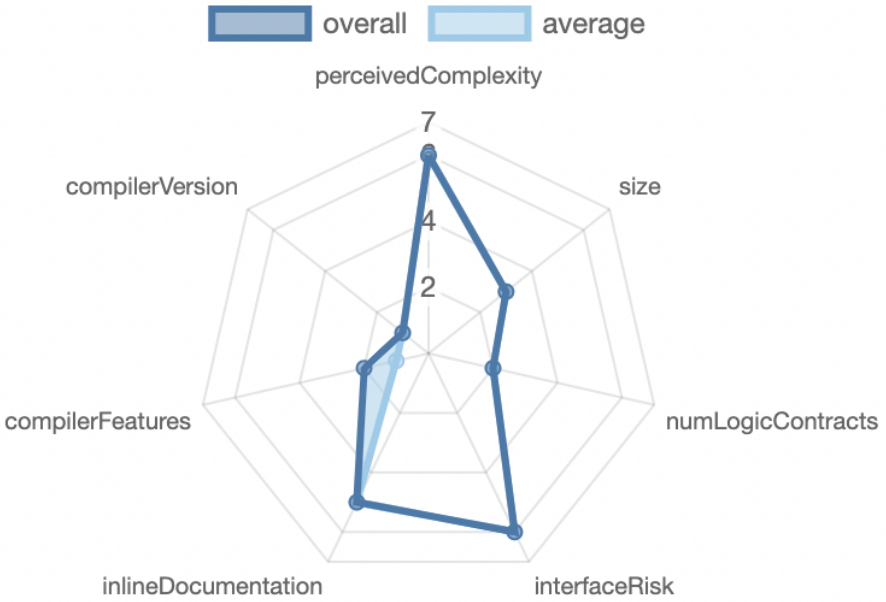


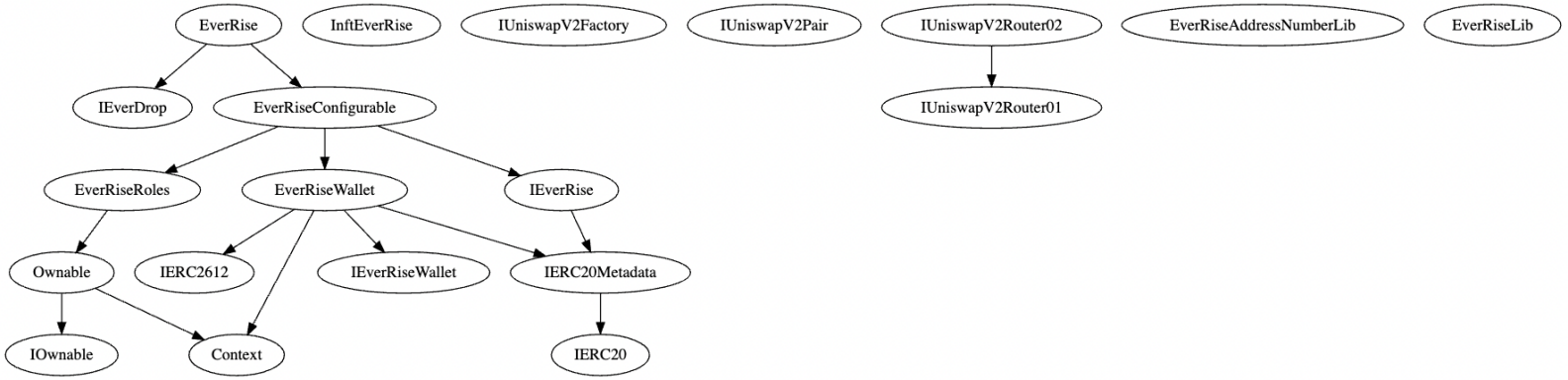


Full version: https://chainsulting.de/wp-content/uploads/2022/03/solidity-metrics-nft-staking-v3.html

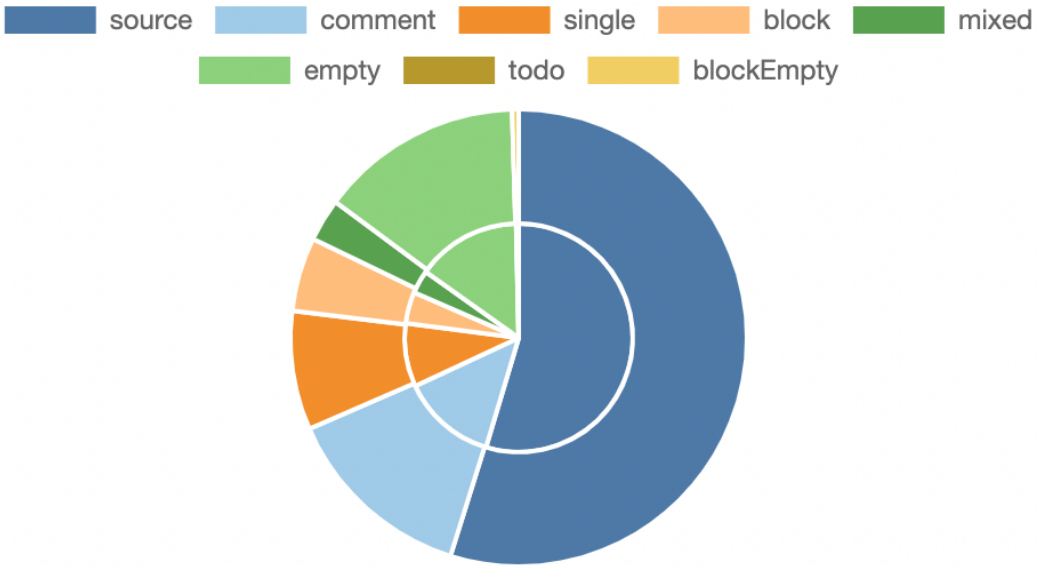
4.5 Metrics / Source Lines & Risk (Token v3)

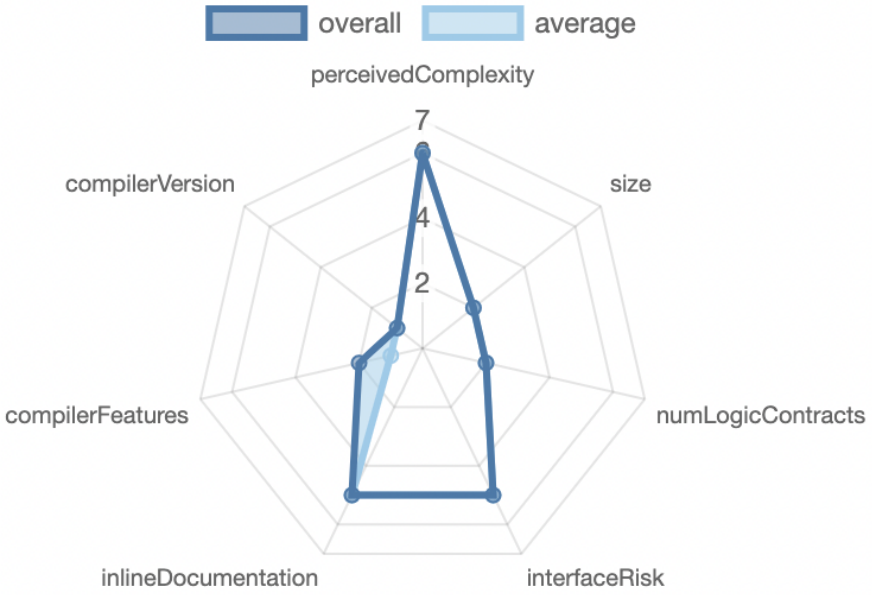


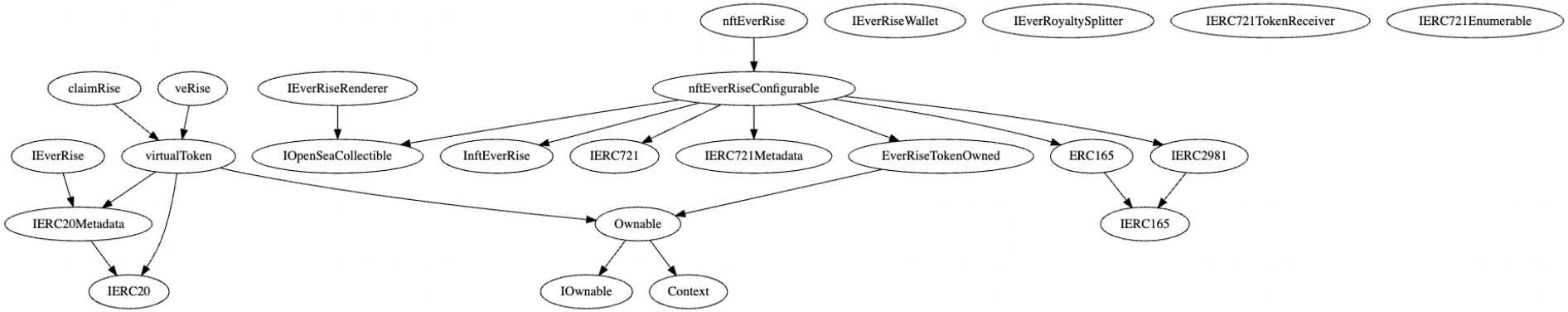




4.5.1 Metrics / Source Lines & Risk (Staking v3)







4.6 Metrics / Capabilities (Token v3)

Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
0.8.13		yes		

Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover	New/Create/Create2
yes			yes	yes	

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
186	7

External	Internal	Private	Pure	View
177	142	21	12	74

StateVariables

Total	Public
50	20

4.6.1 Metrics / Capabilities (Staking v3)

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥️ Uses Assembly	💣 Has Destroyable Contracts
`0.8.13`		`yes`		

📩 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧱 Uses Hash Functions	🪚 ECRecover	🌀 New/Create/Create2
`yes`					`yes` → `NewContract:veRise` → `NewContract:claimRise`

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
133	10

External	Internal	Private	Pure	View
123	77	32	8	75

StateVariables

Total	🌐Public
42	24

4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝📚🔍🎨	EverRise_flat.sol	8	12	1774	1528	1034	250	1063	💰💾🧱🪚🔆Σ
📝📚🔍🎨	**Totals**	**8**	**12**	**1774**	**1528**	**1034**	**250**	**1063**	💰💾🧱🪚🔆Σ

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝🔍🎨	nftEverRise_flat.sol	9	15	1577	1368	943	271	931	💰💾🌀🔆♻Σ
📝🔍🎨	**Totals**	**9**	**15**	**1577**	**1368**	**943**	**271**	**931**	💰💾🌀🔆♻Σ

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, external interfaces, ...)

5. Scope of Work

Following contracts have been tested:

- EverRise_flat.sol
- nftEverRise_flat.sol

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

- Fees are correctly distributed and can be set
- Owner/Deployer is not able to pause the token contract or freeze user funds
- The user can withdraw the stake at anytime and receiving the correct reward
- Owner/Deployer is not able to withdraw staked token from user
- Owner/Deployer cannot pause the staking
- Only the Owner is able to set rewards
- Royalties for the NFT sales are paid out correctly
- The contracts are not susceptible to reentrancy attacks
- The smart contract is coded according to the newest standards and in a secure way.

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, Chainsulting‘s experts found **no Critical issues** in the code of the smart contract.

HIGH ISSUES
During the audit, Chainsulting’s experts found **no High issues** in the code of the smart contract.

MEDIUM ISSUES
During the audit, Chainsulting’s experts found **2 Medium issues** in the code of the smart contract.

5.1.1 Overpowered onlyOwner rights
Severity: MEDIUM
Status: FIXED
Code: CWE-282, CWE-284
File(s) affected: All
Update: EverOwn is a smart contract locker available on the ETH/BSC/FTM/AVAX/MATIC blockchains. EverOwn has a DAO type of governance voting system that allows for 100% transparency and user involvement. Privileged roles are only granted when a successful vote is conducted on a proposed modification. The voting period in EverOwn has a minimum 24hr time span to allow for awareness that privileged operations are being requested. Additionally, EverOwn contains a Legacy Provision Protocol, where an alternate owner or backup wallet can be assigned as the owner the contract unlocks to in case something happens to the original owner, or the original owner's wallet becomes compromised. Transfer of ownership is only granted with a successful weighted community vote; as the community has governance over the EverOwn contract locker. This eliminates the risk of a single point of failure due to the private key of the owner account.

Attack / Description	Code Snippet	Result/Recommendation

EverRise_flat.sol contract contains onlyOwner functions that set
_setEverBridgeVaultAddress
_setStakingAddress
_setSwapEnabled
_transferOwnership
and Roles for Limits, Fees, BuyBack, NFTs, Exchanges, BuyBack and Staking
 	
nftEverRise_flat.sol contract contains onlyOwner functions that set
_transferOwnership
_setEverRiseToken
_setAchievementNfts
_addAddressToCreate
_removeAddressToCreate
_setNftRoyaltyFeePercent
_setRoyaltyAddress
_setRendererAddress
_excludeFromReward
_includeInReward
 	
This overpowered rights can be dangerous and if it is compromised or the owner acts maliciously it can lead to devastating consequences for
	Sample
	Line 281 – 346 (EverRise_flat.sol)
	`contract EverRiseRoles is Ownable {`
	`  mapping (Role => mapping (address => bool)) public roles;`
	
	`  enum Role`
	`  {`
	`  NotValidRole,`
	`  BuyBack,`
	`  Staking,`
	`  Limits,`
	`  Liquidity,`
	`  Fees,`
	`  Exchanges,`
	`  Nfts,`
	`  CrossChainBuyback,`
	`  Upgrader`
	`  }`
	
	`  event ControlAdded(address indexed controller, Role indexed role);`
	`  event ControlRemoved(address indexed controller, Role indexed role);`
	
	`  function _onlyController(Role role) private view {`
	`  if (!roles[role][_msgSender()]) revert CallerNotApproved();`
	`  }`
	
	`  modifier onlyController(Role role) {`
		It is recommended to use a Multisignature (Gnosis Safe) structure for the address, which has the ownership rights or roles. It is recommended to use a two-step process when transferring ownership, to ensure the new owner has access and control to the new Owner address. That avoids loss of ownership over the contract.
		
		The Ownership is supposed to be transferred to EverOwn, which is out of our audit scope and needs to be fully trusted for governance.

the token making it completely unusable.	``` _onlyController(role);
	        _;
	    }
	
	    constructor() {
	        address deployer = _msgSender();
	        roles[Role.BuyBack][deployer] = true;
	        roles[Role.Staking][deployer] = true;
	        roles[Role.Limits][deployer] = true;
	        roles[Role.Liquidity][deployer] = true;
	        roles[Role.Fees][deployer] = true;
	        roles[Role.Exchanges][deployer] = true;
	        roles[Role.Nfts][deployer] = true;
	    }
	
	    function hasRole(Role role, address
	controller) public view returns (bool) {
	        return roles[role][controller];
	    }
	
	    function addControlRole(address
	newController, Role role) external onlyOwner
	    {
	        if (role == Role.NotValidRole) revert
	NotZero();
	        if (newController == address(0)) revert
	NotZeroAddress();
	        if (roles[role][newController]) revert
	InvalidAddress();
	
	        roles[role][newController] = true;
	
	        emit ControlAdded(newController, role);```	

```
    }

    function removeControlRole(address
oldController, Role role) external onlyOwner
    {
        if (role == Role.NotValidRole) revert
NotZero();
        if (oldController == address(0)) revert
NotZeroAddress();
        if (!roles[role][oldController]) revert
InvalidAddress();

        roles[role][oldController] = false;

        emit ControlRemoved(oldController,
role);
    }
}
```

5.1.2 Exclude addresses

Severity: MEDIUM
Status: FIXED
Code: NA
File(s) affected: nftEverRise_flat.sol
Update: EverOwn is a smart contract locker available on the ETH/BSC/FTM/AVAX/MATIC blockchains. EverOwn has a DAO type of governance voting system that allows for 100% transparency and user involvement. Privileged roles are only granted when a successful vote is conducted on a proposed modification. The voting period in EverOwn has a minimum 24hr time span to allow for awareness that privileged operations are being requested. Additionally, EverOwn contains a Legacy Provision Protocol, where an alternate owner or backup wallet can be assigned as the owner the contract unlocks to in case something happens to the original owner, or the original owner's wallet becomes compromised. Transfer of ownership is only granted with a successful weighted community vote; as the community has governance over the EverOwn contract locker. This eliminates the risk of a single point of failure due to the private key of the owner account.

Attack / Description	Code Snippet	Result/Recommendation

The owner can exclude an account from the rewards and include it back later. Abusing this mapping can cause distribution of rewards only to specific addresses and exclude others.	Line 703 – 714 (nftEverRise_flat.sol)
	
	`function excludeFromReward(address account) public onlyOwner() {`
	`if (account == address(0)) revert NotZeroAddress();`
	`if (_isExcludedFromReward[account]) revert InvalidAddress();`
	
	`if(_rOwned[account] > 0) {`
	`_tOwned[account] = tokenFromRewards(_rOwned[account]);`
	`}`
	`_isExcludedFromReward[account] = true;`
	`_excludedList.push(account);`
	
	`emit ExcludedFromRewards(account);`
	`}`
		It is recommended to use a Multisignature (Gnosis Safe) structure for the address, which has the ownership rights or roles. It is recommended to use a two-step process when transferring ownership, to ensure the new owner has access and control to the new Owner address. That avoids loss of ownership over the contract.
		
		The Ownership is supposed to be transferred to EverOwn, which is out of our audit scope and needs to be fully trusted for governance.

LOW ISSUES

During the audit, Chainsulting's experts found **3 Low issues** in the code of the smart contract.

5.1.3 Variables initialized as their types default value
Severity: LOW
Status: FIXED
Code: NA
File(s) affected: EverRise_flat.sol, nftEverRise_flat.sol

Attack / Description	Code Snippet	Result/Recommendation

State variables do not need to be initialized to their default variables. For example uint a; defaults to a = 0;	Line 846-849 (EverRise_flat.sol)
	`uint256 public totalBuyVolume = 0;`
	`uint256 public totalSellVolume = 0;`
	`uint256 public transactionCap = 0;`
	
	Line 858-860 (EverRise_flat.sol)
	`uint256 internal _nextBuybackAmount = 0;`
	`uint256 internal _latestBuybackBlock = 0;`
	
	Line 1194 (EverRise_flat.sol)
	`uint256 private _holders = 0;`
	
	Line 600-602 (nftEverRise_flat.sol)
	`uint256 public totalAmountEscrowed = 0;`
	`uint256 public totalAmountVoteEscrowed = 0;`
	`uint256 public totalRewardsDistributed = 0;`
		It is recommended to just declare the variables e.g. uint256 public totalBuyVolume;
		if they start as the default values like 0 for uint, without the need to initialize them first.

5.1.4 Variables that can be made constant or immutable

Severity: LOW
Status: ACKNOWLEDGED
Code: NA
File(s) affected: EverRise_flat.sol

Attack / Description	Code Snippet	Result/Recommendation

State variables that do not change during lifetime or are set at construction and never change, can be made constant or immutable.	Line 829 (EverRise_flat.sol)
	`address public burnAddress = 0x000000000000000000000000000000000000dEaD;`
	
	Line 826 (EverRise_flat.sol)
	`address payable public businessDevelopmentAddress =payable(0x24D8DAbebD6c0d5CcC88EC40D95Bf8eB64F0CF9E);`
		It is recommended to make burnAddress and businessDevelopmentAddress constant variables. Variables that never change can be declared constant, which saves gas.

5.1.5 Hardcoded address

Severity: INFORMATIONAL
Status: ACKNOWLEDGED
Code: NA
File(s) affected: EverRise_flat.sol

Attack / Description	Code Snippet	Result/Recommendation
Hardcoded values like addresses can impact the life time of the contract, as they may change in the future.	`address payable public businessDevelopmentAddress = payable(0x24D8DAbebD6c0d5CcC88EC40D95Bf8eB64F0CF9E); // Business Development Address`	It is recommended to keep addresses replaceable during lifetime. Additionally it reduces chances of making errors with addresses, consider tests that check if the value of this address is the correct value.

INFORMATIONAL ISSUES

During the audit, Chainsulting's experts found **2 Informational issues** in the code of the smart contract.

5.1.6 Missing test cases

Severity: INFORMATIONAL
Status: ACKNOWLEDGED
Code: NA
File(s) affected: All

Attack / Description	Code Snippet	Result/Recommendation
The current implementation has no unit-tests.	NA	It is highly recommended to write test cases for every function and control the behaviour at any time of contract executions. It is necessary to check the desired functionality of executed code. Tests are also a good proof that the written code is working in the intended way.

5.1.7 Missing natspec documentation

Severity: INFORMATIONAL
Status: ACKNOWLEDGED
Code: CWE-1053
File(s) affected: All

Attack / Description	Code Snippet	Result/Recommendation
Solidity contracts can use a special form of comments to provide rich documentation for function, return variables, and more. This special form is named Ethereum Natural	N/A	It is recommended to include natspec documentation and follow the doxygen style including @author, @title, @notice, @dev, @param, @return and make it easier to review and understand your smart contract.

Language Specification Format(NatSpec).		

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	✅

ID	Title	Relationships	Test Result
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	✅
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	✅

ID	Title	Relationships	Test Result
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	✅
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	✅

ID	Title	Relationships	Test Result
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	✅
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

5.3.1 Fees are correctly distributed and can be set
Status: tested and verified ✅

5.3.2 Owner/Deployer is not able to pause the token contract or freeze user funds
Status: tested and verified ✅

5.3.3 The user can withdraw the stake at anytime and receiving the correct reward
Status: tested and verified ✅

5.3.4 Owner/Deployer is not able to withdraw staked token from user
Status: tested and verified ✅

5.3.5 Owner/Deployer cannot pause the staking
Status: tested and verified ✅

5.3.6 Only the Owner is able to set rewards
Status: tested and verified ✅

5.3.7 Royalties for the NFT sales are paid out correctly
Status: tested and verified ✅

5.3.8 The contracts are not susceptible to reentrancy attacks
Status: tested and verified ✅

5.3.9 The smart contract is coded according to the newest standards and in a secure way.
Status: tested and verified ✅

6. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase. The final debriefs took place on the March 26, 2022.

The main goal of the audit was to verify the claims regarding the security of the smart contract and the claims inside the scope of work. During the audit, 2 Medium, 3 Low and 2 Informational issues were found after the manual and automated security testing.

Update (01.04.2022): https://github.com/everrise-ecosystem/everrise-v3-contracts/commit/b92f0971db700ac9428e076a0f7af6051b4eeef4 has addressed issues and the auditor has acknowledged EverOwn as safe to use alternative to Gnosis safe.

7. Deployed Smart Contract

VERIFIED

EverRise: https://etherscan.io/address/0xC17c30e98541188614dF99239cABD40280810cA3#code

nftEverRise: https://etherscan.io/address/0x23cD2E6b283754Fd2340a75732f9DdBb5d11807e#code

veRiseToken: https://etherscan.io/address/0xDbA7b24257fC6e397cB7368B4BC922E944072f1b#code

claimRiseToken: https://etherscan.io/address/0xbBD7B847C6d0d0B5691518a363194D71426475F1#code

8. About the Auditor

Chainsulting is a professional software development firm based in Germany that provides comprehensive distributed ledger technology (DLT) solutions. Some of their services include blockchain development, smart contract audits and consulting.

Chainsulting conducts code audits on market-leading blockchains such as Hyperledger, Tezos, Ethereum, Binance Smart Chain, and Solana to mitigate risk and instil trust and transparency into the vibrant crypto community. They have also reviewed and secure the smart contracts of 1Inch, POA Network, Unicrypt, Amun, Furucombo among numerous other top DeFi projects.

Chainsulting currently secures $100 billion in user funds locked in multiple DeFi protocols. The team behind the leading audit firm relies on their robust technical know-how in the blockchain sector to deliver top-notch smart contract audit solutions tailored to the clients' evolving business needs.

The blockchain security provider brings the highest security standards to crypto and blockchain platforms, helping to foster growth and transparency within the fast-growing ecosystem.

Check our website for further information: https://chainsulting.de