InvestaX

Primary Issuer

SMART CONTRACT AUDIT

02.06.2023

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 IC SG 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 (09.05.2022)	Layout
0.4 (13.05.2022)	Automated Security Testing Manual Security Testing
0.5 (14.05.2023)	Verify Claims and Test Deployment
0.6 (15.05.2023)	Testing SWC Checks
0.9 (16.05.2023)	Summary and Recommendation
1.0 (19.05.2023)	Final document
1.1 (02.06.2023)	Re-check (374c438209e841a34907443e2da07843b99e3c6a)

2. About the Project and Company

Company address:

IC SG PTE. LTD.
100 TRAS STREET, #16-01, 100 AM
Singapore 079027

Website: https://investax.io

Twitter: https://twitter.com/investax

Telegram: https://t.me/investaxtelegram

LinkedIn: https://www.linkedin.com/company/investax

Medium: https://medium.com/@investax

YouTube: https://www.youtube.com/channel/UCAJI2c_gP8TUbaKOrjMX0IA

2.1 Project Overview

InvestaX, a pioneer in digital securities within private capital markets, has been transforming the financial landscape since its inception in 2015. Recognized and licensed by the Monetary Authority of Singapore (MAS), the platform offers comprehensive, end-to-end solutions for the issuance, trading, and custody of tokenized assets.

Utilizing state-of-the-art blockchain technology, InvestaX redefines the way we approach investments. By tokenizing assets, the platform provides a streamlined, secure, and transparent method for handling private market assets. This not only mitigates traditional frictions associated with asset management, but also reduces costs and enhances overall transactional transparency.

In a bid to further its innovative streak, InvestaX has introduced a groundbreaking product: a digital asset security tied to a Non-Fungible Token (NFT). This unique financial instrument synergistically merges the advantages of both Decentralized Finance (DeFi) and Centralized Finance (CeFi), reflecting a new wave of hybrid financial solutions.

InvestaX's initiative marks a significant milestone in transforming the way value is created and transferred within investor communities. With its commitment to leveraging blockchain's potential, InvestaX is truly driving the evolution of the financial sector.

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.

5. Metrics

The metrics section should give the reader an overview on the size, quality, flows and capabilities of the codebase, without the knowledge to understand the actual code.

5.1 Tested Contract Files

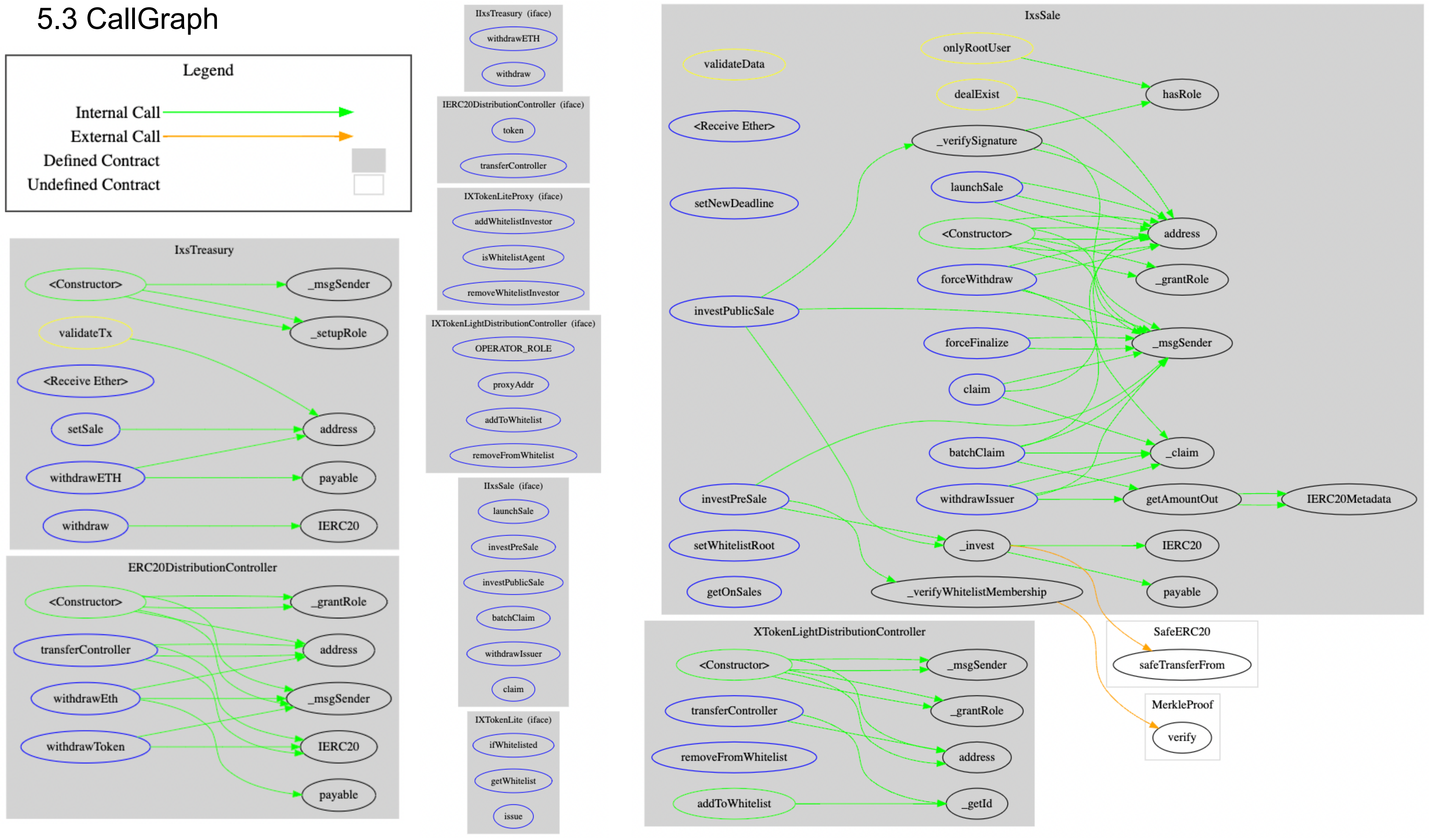
The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability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

File	Fingerprint (MD5)
./IxsTreasury.sol	a27b0c411d2f2cb6a7a701d8f603c652
./XTokenLightDistributionController.sol	1bfb4b3c093363aea3a7ae3c856705eb
./IxsSale.sol	4d1508a92d94196d28299dea76035c6e
./ERC20DistributionController.sol	695b8f5c2e89451a4da16fcdb7147b23
./interfaces/IXTokenLite.sol	0869742e12959c1fd42ccc47f8c2921e
./interfaces/IIxsTreasury.sol	5c466e3a5bae4b5370feb1b45719f893
./interfaces/IERC20DistributionController.sol	b595e2f8b540bc0fc6bf1cea1c16c6cc
./interfaces/IXTokenLiteProxy.sol	4cacf7676c535d1ec91d4e3db088ceab
./interfaces/IXTokenLightDistributionController.sol	0eb1277c375d6e828032b1c3196bd500
./interfaces/IIxsSale.sol	29cd279d4bfe1b62c9f6b2a148a53225

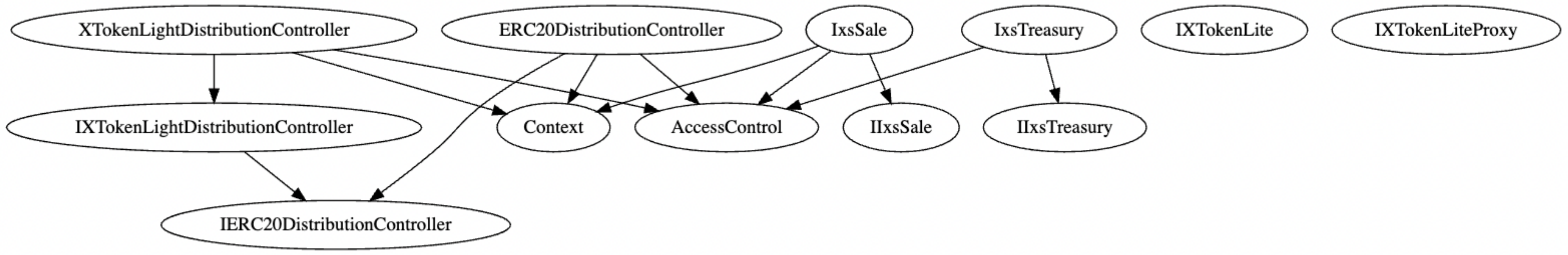
5.2 Used Code from other Frameworks/Smart Contracts (direct imports)

Dependency / Import Path	**Source**
@openzeppelin/contracts/access/AccessControl.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.8.0/contracts/access/AccessControl.sol
@openzeppelin/contracts/token/ERC20/IERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.8.0/contracts/token/ERC20/IERC20.sol
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.8.0/contracts/token/ERC20/extensions/IERC20Metadata.sol
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.8.0/contracts/token/ERC20/utils/SafeERC20.sol
@openzeppelin/contracts/utils/Context.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.8.0/contracts/utils/Context.sol
@openzeppelin/contracts/utils/cryptography/MerkleProof.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.8.0/contracts/utils/cryptography/MerkleProof.sol

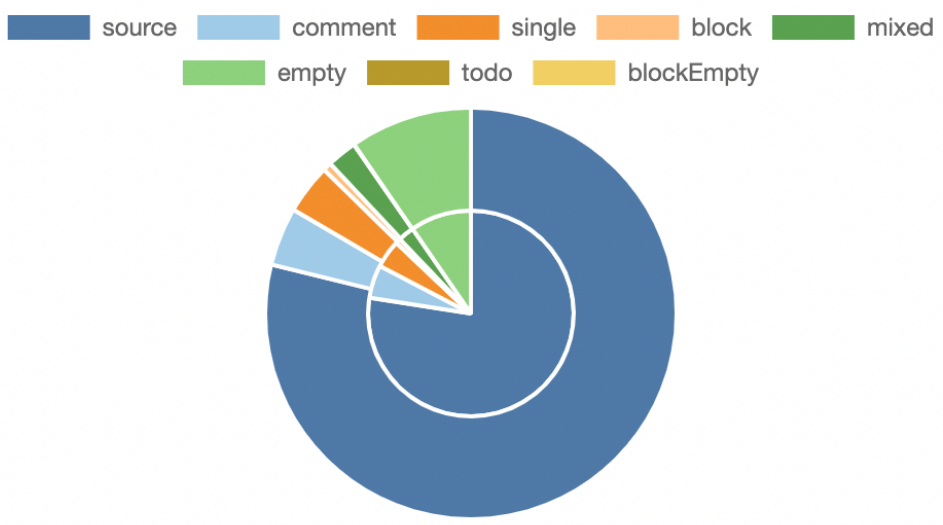
5.3 CallGraph

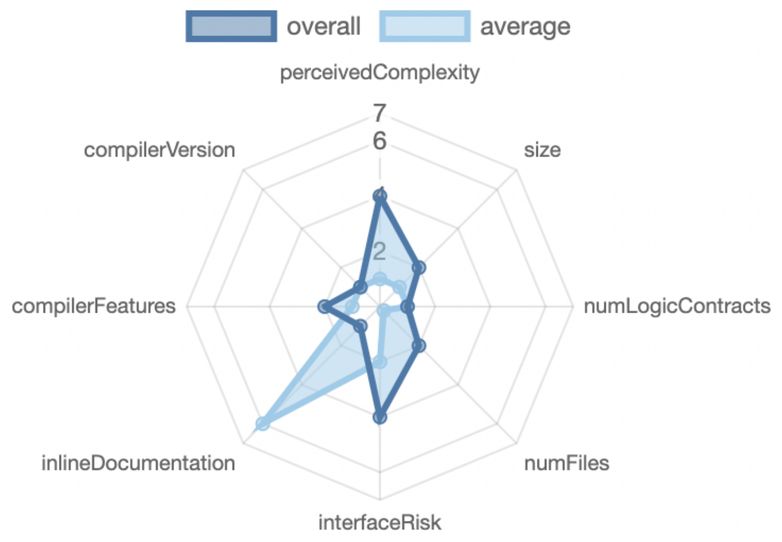


5.4 Inheritance Graph


XTokenLightDistributionController
ERC20DistributionController
IxsSale
IxsTreasury
IXTokenLite
IXTokenLiteProxy
IXTokenLightDistributionController
Context
AccessControl
IIxsSale
IIxsTreasury
IERC20DistributionController

5.5 Source Lines & Risk





5.6 Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
`^0.8.17`		`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
43	6

External	Internal	Private	Pure	View
41	43	0	2	6

StateVariables

Total	🌐Public
19	17

5.7 Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝	contracts/XTokenLightDistributionController.sol	1		71	59	51	1	49	🧮
📝	contracts/IxsSale.sol	1		472	410	378	23	207	💰📤🧮🔖
📝	contracts/ERC20DistributionController.sol	1		61	57	40	7	44	📤🧮
📝	contracts/IxsTreasury.sol	1		53	46	36	1	39	💰📤🧮
🔍	contracts/interfaces/IXTokenLite.sol		1	16	5	3	1	7	
🔍	contracts/interfaces/IIxsSale.sol		1	90	71	58	4	19	💰
🔍	contracts/interfaces/IXTokenLightDistributionController.sol		1	13	6	4	1	11	
🔍	contracts/interfaces/IXTokenLiteProxy.sol		1	10	5	3	1	7	
🔍	contracts/interfaces/IERC20DistributionController.sol		1	10	7	4	1	5	
🔍	contracts/interfaces/IIxsTreasury.sol		1	14	7	4	1	5	
📝🔍	**Totals**	**4**	**6**	**810**	**673**	**581**	**41**	**393**	💰📤🧮🔖

- **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, ...)

6. Scope of Work

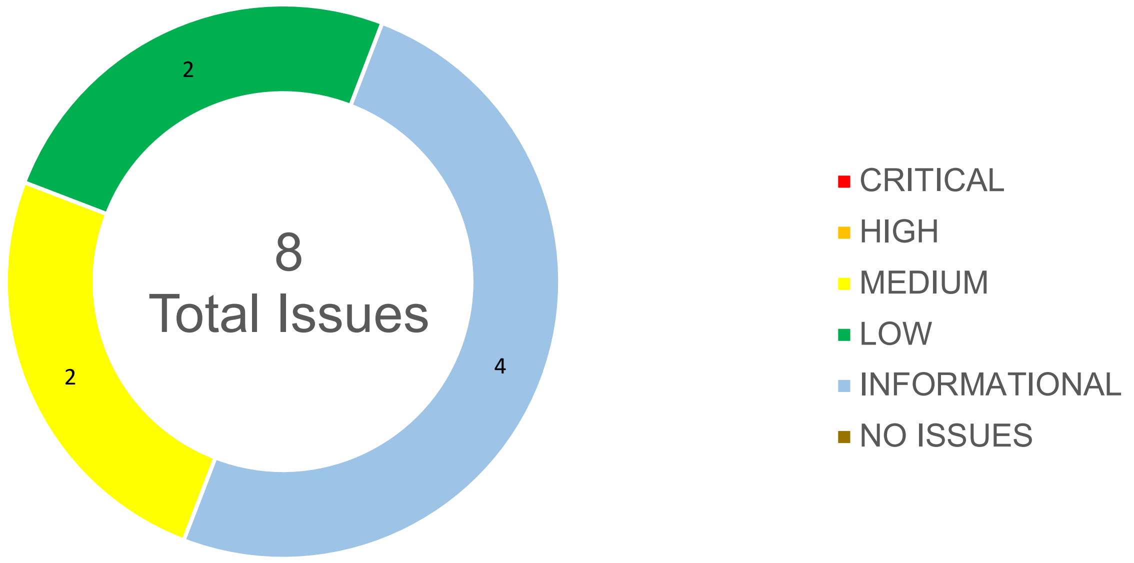
The InvestaX Team provided us with the files that needs to be tested. The scope of the audit is the primary Issuer contract.

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

1. The contract creators have implemented thorough access control mechanisms to prevent unauthorized actions.
2. The contract developers have made sure that the contract logic is gas-efficient and optimized to minimize the risk of running out of gas during contract execution.
3. Major functions of the contract working as expected, such as, Pre-/Public-sale creation, Whitelisting, Claim, Withdraw.
4. 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.

6.1 Findings Overview



No	Title	Severity	Status
6.2.1	Potential for Re-Entrancy Attacks	MEDIUM	FIXED
6.2.2	Misuse of Low-Level Calls	MEDIUM	FIXED
6.2.3	Excessive Permissions For Certain Functions	LOW	ACKNOWLEDGED
6.2.4	Misleading Function Naming	LOW	FIXED
6.2.5	Floating Compiler Version	INFORMATIONAL	FIXED
6.2.6	Missing Natspec Documentation	INFORMATIONAL	ACKNOWLEDGED
6.2.7	No Event Emitted After Changing Role	INFORMATIONAL	ACKNOWLEDGED
6.2.8	Unexpected Behavior of Whitelist Functions	INFORMATIONAL	ACKNOWLEDGED

6.2 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 issue** in the code of the smart contract.

6.2.1 Potential for Re-Entrancy Attacks

Severity: MEDUM
Status: FIXED
Code: CWE-839
File(s) affected: IxsSale.sol
Update: https://github.com/IX-Swap/ixs-primary-issuer-contracts/commit/c1b33c9ba549acb044ef1f3f2b6b163d9e9897ca

Attack / Description	The code is vulnerable to reentrancy attacks, where an external contract can maliciously call back into the contract before the ongoing execution is completed. This can result in unauthorized state changes or loss of funds. The vulnerable code patterns can be found in the investPublicSale and _invest functions. The vulnerability arises due to the lack of proper checks and ordering of state changes and external interactions. A potential attack scenario is when a malicious contract calls the investPublicSale function repeatedly within a single transaction, executing the callback before the internal state changes are completed. This can enable the attacker to repeatedly withdraw funds or perform unauthorized operations.

Code	Line 184 – 215 (IxsSale.sol)

```
function investPublicSale(
        uint256 _saleId,
        uint256 _amount,
        InvestStruct calldata _investData
    ) external payable dealExist(_saleId) validateData(_amount) {
        address sender = _msgSender();
        LaunchSale memory sale = onSales[_saleId];
        uint256 currentTime = block.timestamp;
        {
            require(
                !onSaleInvests[_saleId].forceFinalize,
                "IxsSale: SALE_FORCE_FINILIZED"
            );
            require(
                _verifySignature(_saleId, _amount, _investData),
                "IxsSale: INVALID_SIGNATURE"
            );
            require(
                currentTime >= sale.deadlines[1] &&
                    currentTime < sale.deadlines[2],
                "IxsSale: TIME_ERROR"
            );
            require(
                _amount >= sale.investLimits[2] &&
                _amount + investAmounts[_saleId][sender].amountPubSale <=
                    sale.investLimits[3] &&
                _amount + onSaleInvests[_saleId].dealBalance <=
                    sale.caps[2],
                "IxsSale: UNBOUNDED_INVESTMENT"
            );
        }
```

	`_invest(_amount, sender, sale.paymentToken);`

Result/Recommendation	It’s recommended to follow the steps below:
	
	1. Implement the "Checks-Effects-Interactions" pattern to mitigate reentrancy vulnerabilities. Ensure that all internal state changes are completed before making any external calls.
	
	2. Add a reentrancy guard at the beginning of each function to prevent multiple reentrant calls within the same transaction. OpenZeppelin provides a ReentrancyGuard contract that can be imported and used as a modifier on vulnerable functions to protect against reentrancy attacks. Consider importing and using the ReentrancyGuard contract from the OpenZeppelin library to add an extra layer of protection against reentrancy vulnerabilities.
	
	import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
	
	contract XTokenLightDistributionController is ReentrancyGuard {
	    // Contract code
	}
	
	contract IxsSale is ReentrancyGuard {
	    // Contract code
	}
	
	By using the ReentrancyGuard contract from OpenZeppelin, the modifier will automatically prevent reentrant calls and provide an added layer of security to the contract.

6.2.2 Misuse of Low-Level Calls

Severity: MEDIUM
Status: FIXED

Code: CWE-754
File(s) affected: IxsTreasury.sol
Update: https://github.com/IX-Swap/ixs-primary-issuer-contracts/commit/23303917f30e5b894e2ead20ddbfacd1b16b868b

Attack / Description	The contract uses send() function for transferring Ether. The send() function only forwards 2300 gas which might not be sufficient after the EIP-1884 update, where gas cost for certain operations was increased. This could result in a potential loss of funds if the called contract's fallback function consumes more than 2300 gas, causing the transaction to fail.

Code	Line 37 – 43 (IxsTreasury.sol)
	`function withdrawETH(address _to, uint256 _amount)`
	`    external`
	`    validateTx(_to, _amount)`
	`{`
	`    require(payable(_to).send(_amount), "IxsTreasury: ETH_TRANSFER_FAILED");`
	`    emit Withdraw(address(0), _to, _amount);`
	`}`

Result/Recommendation	Instead of using low-level calls like send() or transfer(), consider using the call() function which forwards all available gas. However, using call() also necessitates the implementation of reentrancy guards to prevent potential reentrancy attacks. The OpenZeppelin library provides a ReentrancyGuard contract that can be inherited to use the nonReentrant modifier.
	
	Here's an example of how the code can be refactored:
	
	import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
	
	contract IxsTreasury is IIxsTreasury, AccessControl, ReentrancyGuard {
	  ...
	  function withdrawETH(address payable _to, uint256 _amount) external validateTx(_to, _amount) nonReentrant {

	`(bool success, ) = _to.call{value: _amount}("");`
	`require(success, "IxsTreasury: ETH_TRANSFER_FAILED");`
	`emit Withdraw(address(0), _to, _amount);`
	`}`
	`...`
	`}`
	
	Source: https://consensys.net/diligence/blog/2019/09/stop-using-soliditys-transfer-now/

LOW ISSUES

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

6.2.3 Excessive Permissions For Certain Functions

Severity: LOW
Status: ACKNOWLEDGED
Code: CWE-284
File(s) affected: XTokenLightDistributionController.sol
Update: We will consider using a multisig wallet or timelock for executing critical functions

Attack / Description	There are functions which are only restricted by the OPERATOR_ROLE, which can execute critical functionality such as adding or removing from whitelist, forceWithdraw, forceFinalize. If the account with OPERATOR_ROLE gets compromised, the attacker could add arbitrary addresses to the whitelist or remove existing ones. Proof of Concept: addToWhitelist and removeFromWhitelist functions in the XTokenLightDistributionController contract are only restricted by the OPERATOR_ROLE.

Code	Line 29 – 47 (XTokenLightDistributionController.sol)
	`function addToWhitelist(address _to)`
	`        public`
	`        override`
	`        onlyRole(OPERATOR_ROLE)`
	`    {`
	`        require(`
	`            !IXTokenLite(token).ifWhitelisted(_to),`
	`            "XTokenLightDistributionController: ALLREADY_WHITELISTED"`
	`        );`
	`        IXTokenLiteProxy(proxyAddr).addWhitelistInvestor(_to, _getId(_to));`
	`    }`
	
	`    function removeFromWhitelist(address _to)`
	`        external`
	`        override`
	`        onlyRole(OPERATOR_ROLE)`
	`    {`
	`        IXTokenLiteProxy(proxyAddr).removeWhitelistInvestor(_to);`
	`    }`

Result/Recommendation	Consider implementing a multisig wallet or timelock for executing critical functions.

6.2.4 Misleading Function Naming

Severity: LOW
Status: FIXED
Code: CWE-685
File(s) affected: IxsSale.sol
Update: https://github.com/IX-Swap/ixs-primary-issuer-contracts/commit/374c438209e841a34907443e2da07843b99e3c6a

Attack / Description	The function getAmountOut might be misleading since it doesn't actually get an amount out. Instead, it calculates and returns an output amount based on an input amount and a conversion rate.
Code	Line 373 – 390 (IxsSale.sol)

```
  function getAmountOut(
        uint256 _saleId,
        uint256 _amount
    ) public view returns(uint256 _amountOut){
        _amountOut = _amount * onSales[_saleId].rate * IXS_AMOUNT_OUT_PRECISION; // calculate how much claim secTokens
        address secTokenAddr = IERC20DistributionController(onSales[_saleId].controller).token();
        uint256 decimalsOfSec = IERC20Metadata(secTokenAddr).decimals();
        uint256 decimalsOfPay = onSales[_saleId].paymentToken == WETH
            ?
            18
            :
            IERC20Metadata(onSales[_saleId].paymentToken).decimals();
        if (decimalsOfSec >= decimalsOfPay){
            _amountOut = (_amountOut * 10 ** (decimalsOfSec - decimalsOfPay)) / IXS_AMOUNT_OUT_PRECISION**2;
        } else if(decimalsOfSec < decimalsOfPay){
            _amountOut = (_amountOut / 10 ** (decimalsOfPay - decimalsOfSec)) / IXS_AMOUNT_OUT_PRECISION**2;
        }
    }
```

Result/Recommendation	Rename getAmountOut to something more descriptive like calculateOutputAmount or getConvertedAmount.

INFORMATIONAL ISSUES

During the audit, Chainsulting's experts found **4 Informational issues** in the code of the smart contract

6.2.5 Floating Compiler Version

Severity: INFORMATIONAL
Status: FIXED
Code: SWC-103
File(s) affected: All
Update: https://github.com/IX-Swap/ixs-primary-issuer-contracts/commit/a020c31524e87f3eb0c3a7acacb57179648bec77

Attack / Description	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.
Code	`pragma solidity ^0.8.17;`
Result/Recommendation	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 0.8.17 See SWC-103: https://swcregistry.io/docs/SWC-103

6.2.6 Missing Natspec Documentation

Severity: INFORMATIONAL
Status: ACKNOWLEDGED
Code: CWE-1059

File(s) affected: All
Update: We have a detailed document of the smart contract, we will release to the public if necessary

Attack / Description	The Solidity contract does not include Natspec documentation, which is a special form of comments used to provide rich documentation for functions, return variables, and other contract elements. Natspec documentation enhances the readability and understandability of the contract code, making it easier for developers and auditors to review and analyze the contract's functionality.

Code	`/**`
	` * @dev Transfer tokens from the caller's address to a specified recipient.`
	` * @param recipient The address of the recipient.`
	` * @param amount The amount of tokens to transfer.`
	` * @return A boolean indicating whether the transfer was successful.`
	` */`
	`function transfer(address recipient, uint256 amount) external returns (bool) {`
	`    // Function implementation`
	`    // ...`
	`}`

Result/Recommendation	It is recommended to include Natspec documentation for each function, including a brief description of its purpose, input parameters, return values, and any important considerations. Use the doxygen-style format for Natspec comments, including tags such as @title, @notice, @dev, @param, and @return, to provide clear and structured documentation.

6.2.7 No Event Emitted After Changing Role

Severity: INFORMATIONAL
Status: ACKNOWLEDGED
Code: NA
File(s) affected: XTokenLightDistributionController.sol
Update: AccessControl library of Openzeppelin already emits event RoleGranted(bytes32 indexed role, address indexed account, address indexed sender)

Attack / Description	The contract code does not emit an event when the operator role has been changed. While this is not a security issue per se, it's a good practice to do so. Events provide an easier way to monitor state changes and any action taken on the contract, especially when they concern critical operations like changing roles. This is particularly important for off-chain services and interfaces interacting with your contract.

Code	Line 18 – 27 (XTokenLightDistributionController.sol)
	`constructor(address _token, address _proxyAddr) {`
	`        require(`
	`            _proxyAddr != address(0) && _token != address(0),`
	`            "XTokenLightDistributionController: ZERO_ADDRESS"`
	`        );`
	`        _grantRole(DEFAULT_ADMIN_ROLE, _msgSender());`
	`        _grantRole(OPERATOR_ROLE, _msgSender());`
	`        proxyAddr = _proxyAddr;`
	`        token = _token;`
	`    }`

Result/Recommendation	In the constructor, you can emit a custom event to signal that the roles have been assigned. You need to define the event and emit it when the change occurs. Here is an example of how you can define and emit an event when a role is granted:
	
	// Event declaration
	event RoleGranted(bytes32 indexed role, address indexed account);
	
	// Emitting the event when a role is granted
	constructor(address _token, address _proxyAddr) {
	    require(
	        _proxyAddr != address(0) && _token != address(0),
	        "XTokenLightDistributionController: ZERO_ADDRESS"
	    );

	`_grantRole(DEFAULT_ADMIN_ROLE, _msgSender());`
	`_grantRole(OPERATOR_ROLE, _msgSender());`
	`proxyAddr = _proxyAddr;`
	`token = _token;`
	
	`emit RoleGranted(DEFAULT_ADMIN_ROLE, _msgSender());`
	`emit RoleGranted(OPERATOR_ROLE, _msgSender());`
	`}`
	
	With these changes, every time a role is granted, an event will be emitted which can be tracked by off-chain services and interfaces.

6.2.8 Unexpected Behavior of Whitelist Functions

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

Attack / Description	The current implementation of addToWhitelist and removeFromWhitelist in XTokenLightDistributionController contract can have some unexpected behavior. Currently, you're calling the ifWhitelisted function to check if an address is whitelisted before adding to the whitelist. However, the same is not done before removing from the whitelist. As a result, if the removeFromWhitelist function is called with an address that is not whitelisted, it will not revert, even though the function does not make any state changes. This might lead to confusion for the users of your contract. Proof of concept: Call removeFromWhitelist with an address that is not whitelisted. It will not revert

	even though it does not make any state changes.

Code	Line 41 – 47 (XTokenLightDistributionController.sol)
	
	`function removeFromWhitelist(address _to)`
	`        external`
	`        override`
	`        onlyRole(OPERATOR_ROLE)`
	`    {`
	`        IXTokenLiteProxy(proxyAddr).removeWhitelistInvestor(_to);`
	`    }`

Result/Recommendation	You can use a similar pattern as you've used in addToWhitelist function, by first checking if the address is whitelisted before attempting to remove it from the whitelist.
	
	`function removeFromWhitelist(address _to)`
	`        external`
	`        override`
	`        onlyRole(OPERATOR_ROLE)`
	`{`
	`    require(`
	`        IXTokenLite(token).ifWhitelisted(_to),`
	`        "XTokenLightDistributionController: NOT_WHITELISTED"`
	`    );`
	`    IXTokenLiteProxy(proxyAddr).removeWhitelistInvestor(_to);`
	`}`
	
	This change will ensure that the function only executes if it will make a state change, and it reverts otherwise, reducing the potential for confusion.

6.3 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	✅

6.4 Verify Claims

6.4.1 The contract creators have implemented thorough access control mechanisms to prevent unauthorized actions.
Status: tested and verified ✅

6.4.2 The contract developers have made sure that the contract logic is gas-efficient and optimized to minimize the risk of running out of gas during contract execution.
Status: tested and verified ✅

6.4.3 Major functions of the contract working as expected, such as, Pre-/Public-sale creation, Whitelisting, Claim, Withdraw.
Status: tested and verified ✅

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

6.5 Unit Tests

```
Compiled 39 Solidity files successfully

  ERC20DistributionController
    ✓ should not accept zero address in constructor
    ✓ prepare & deploy
    ✓ should set contract's fields properly
    ✓ should set roles corectly
    transferController
      ✓ should be done only by operator
      ✓ should not accept zero amount/address
      ✓ should fail to transfer if balance is insufficient
      ✓ should succeed to transfer is balance is sufficient

  XTokenLightDistributionController
    ✓ should not accept zero address in constructor
    ✓ prepare & deploy
    ✓ should set contract's fields properly
    ✓ should set roles corectly
    addToWhitelist
      ✓ should be done only by operator
      ✓ should add new whitelist investor
      ✓ should revert if investor is already whitelisted
    removeFromWhitelist
      ✓ should be done only be operator
      ✓ should remove existent whitelisted investor
    transferController
      ✓ should be done only by operator
      ✓ should not accept zero amount/address
      ✓ should add to whitelist automatically
      ✓ should set the corresponding id
      ✓ should mint XTokenLite tokens to the receiver
```

```
IxsSale
  ✓ should not accept zero addresses in constructor
  ✓ prepare & deploy
  ✓ should set roles and contract parameters correctly
  launchSale
    ✓ prepare sale data
    ✓ should be done only by operator
    ✓ should not accept zero address
    ✓ should not accept zero rate, fee, caps, and invalid deadlines
    ✓ should not accept zero investment limits
    ✓ should set new sale
  investPreSale
    ✓ deal should exist
    ✓ amount should not be zero
    ✓ should not allow to invest if sale is finalized forcely
    ✓ should accept only whitelisted members
    ✓ the investments should be bounded
    ✓ should allow investments in erc20 tokens
    ✓ should allow investments in native assets
    ✓ should be available only for pre-sale
  investPublicSale
    ✓ deal should exist
    ✓ amount should not be zero
    ✓ should not allow to invest if sale is finalized forcely
    ✓ should not accept invalid signatures
    ✓ the investments should be bounded
    ✓ should allow investments in erc20 tokens
    ✓ should allow investments in native assets
    ✓ should meet the sale deadlines
  forceFinalize
    ✓ should not allow to finalize again
    ✓ should be available only for existing deals
    ✓ should be done only by the sale initiator/operator
    ✓ should be finalized only when soft cap is reached
    ✓ should finalize the sale
```

```
forceWithdraw
  ✓ should be done only by operator
  ✓ should not accept invalid arguments
  ✓ should not be done if issuer already claimed and balance is zero
  ✓ should claim correct amount out
batchClaim
  ✓ should be done for existing sales
  ✓ access should be restricted to the sale initiator/operator
  ✓ should not go out of gas
  ✓ should allow batchClaim only after end sale date
  ✓ should not accept receiver with zero address
  ✓ erc20 investment token, sale failed
  ✓ native investment token, sale failed
  ✓ erc20 investment token, sale success (pre & public sale, soft cap reached)
  ✓ native investment token, sale success (pre & public sale, hard cap reached)
  ✓ native investment token, sale success (only public sale, soft cap reached)
withdrawIssuer
  ✓ should be available only for existing sales
  ✓ should be done only by sale initiator
  ✓ should be disponible only for successful sales
  ✓ should not be available if not all investors claimed or issuer already claimed
  ✓ should return unsold security tokens back
  ✓ should return invested funds with substracted fee (if exists)
claim
  ✓ should be available only for existing sales
  ✓ should not accept invalid arguments
  ✓ should be available only when sale is unsuccessful
  ✓ should return funds back to investor
setWhitelistRoot
  ✓ should not accept zero root
  ✓ should set whitelist root
getAmountOut
  ✓ security token decimals > payment token decimals
  ✓ security token decimals < payment token decimals
  ✓ security token decimals === payment token decimals
```

```
      ✓ payment token is native token
    setNewDeadline
      ✓ should not accept index out of bounds
      ✓ should not accept deadlines smaller than the old ones
      ✓ updated deadline should be greater than the previous one
      ✓ updates sale deadlines

  IxsTreasury
    ✓ prepare & deploy
    ✓ should receive ETH
    setSale
      ✓ should be limited to operator
      ✓ should not accept zero sale address
      ✓ set correctly sale address
    withdrawETH
      ✓ should not accept zero arguments
      ✓ should accept only calls from the sale contract
      ✓ should transfer ETH to receiver
      ✓ should revert if transfer failed
    withdraw
      ✓ should not accept zero arguments
      ✓ should accept only calls from the sale contract
      ✓ should transfer erc20 tokens


  98 passing (13s)
```

6.6 Coverage

37.45% Statements `97/259`
26.47% Branches `90/340`
45.35% Functions `39/86`
39.06% Lines `125/320`

File		Statements		Branches		Functions		Lines	
contracts/		25%	34/136	27.84%	49/176	41.18%	14/34	27.27%	45/165
contracts/interfaces/		100%	0/0	100%	0/0	100%	0/0	100%	0/0
contracts/test/		66.15%	43/65	32.56%	28/86	64%	16/25	66.25%	53/80
contracts/test/utils/		34.48%	20/58	16.67%	13/78	33.33%	9/27	36%	27/75

7. Executive Summary

Two independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase provided by Investax. The main objective of the audit was to verify the security and functionality claims of the smart contract. The audit process involved a thorough manual code review and automated security testing.

Overall, the audit identified a total of 8 issues, classified as follows:
- No critical issues were found.
- No high severity issues were found.
- Two medium severity issues were identified, including the potential vulnerability related to reentrancy attacks and the use of low-level calls like `send` and `transfer`.
- Two low severity issues were discovered, including the excessive permissions and misleading function naming.
- Four informational issues were identified, including code optimizations.

The audit report provides detailed descriptions of each identified issue, including severity levels, CWE classifications, and recommendations for mitigation. It also includes code snippets, where applicable, to demonstrate the issues and suggest possible fixes.

The Investax team is advised to address the identified issues and implement the recommended changes to enhance the code's security and readability. By resolving these issues, the smart contract will be better protected against potential vulnerabilities and will adhere to best practices in Solidity development.

It is crucial for the Investax team to carefully review the audit report, prioritize the issues based on their severity, and allocate resources to implement the recommended fixes. Regular code reviews, security testing, bug bounty and adherence to coding best practices are strongly encouraged to ensure the ongoing security and integrity of the smart contract.

Update (02.06.2023): All important findings have been addressed and re-check by the auditor has been successfully done.

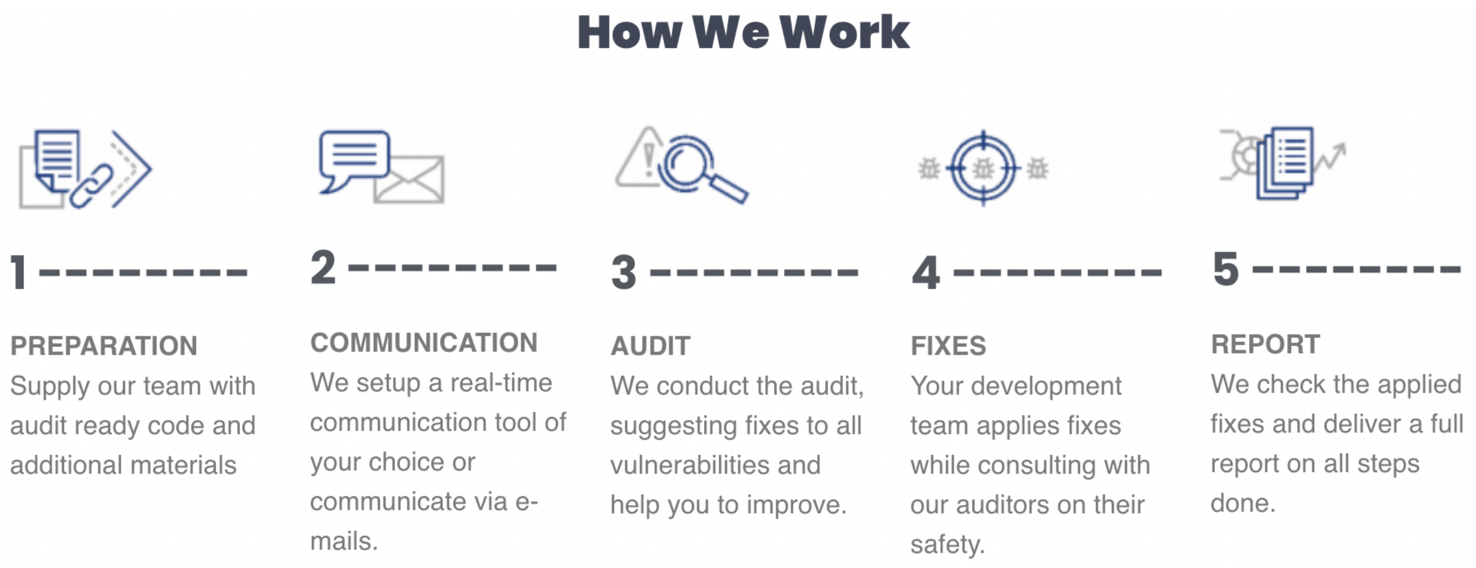
8. About the Auditor

Chainsulting is a professional software development firm, founded in 2017 and based in Germany. They show ways, opportunities, risks and offer comprehensive Web3 solutions. Their services include Web3 development, security and consulting.

Chainsulting conducts code audits on market-leading blockchains such as Solana, Tezos, Ethereum, Binance Smart Chain, and Polygon to mitigate risk and instil trust and transparency into the vibrant crypto community. They have also reviewed and secure the smart contracts of many 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 web3 sector to deliver top-notch smart contract audit solutions, tailored to the clients' evolving business needs.

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



How We Work

1 PREPARATION
Supply our team with audit ready code and additional materials

2 COMMUNICATION
We setup a real-time communication tool of your choice or communicate via e-mails.

3 AUDIT
We conduct the audit, suggesting fixes to all vulnerabilities and help you to improve.

4 FIXES
Your development team applies fixes while consulting with our auditors on their safety.

5 REPORT
We check the applied fixes and deliver a full report on all steps done.