XTON Core

SMART CONTRACT AUDIT

13.03.2024

Made in Germany by Softstack.io

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 XTON INNOVATIONS LLC. 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 (27.02.2024)	Layout
0.4 (05.03.2024)	Automated Security Testing Manual Security Testing
0.5 (08.03.2024)	Verify Claims and Test Deployment
0.9 (12.03.2024)	Summary and Recommendation
1.0 (13.03.2024)	Final document
1.1 (20.03.2024)	Re-check

2. About the Project and Company

Company address:

XTON INNOVATIONS LLC
Reg.No.: 3392 LLC 2024
Euro House, VC, Richmond Hill Rd
Kingstown VC0100
St Vincent & the Grenadines

Website: https://www.xton.org

Twitter (X): https://twitter.com/xton_official

LinkedIn: https://t.me/xton_official

2.1 Project Overview

XTON is a launchpad designed to bridge the gap between blockchain technology and the widespread user base of Telegram, leveraging the liquidity of Ethereum Virtual Machine (EVM) and Telegram Open Network (TON). This platform aims to facilitate the seamless launch of blockchain projects by providing a unique ecosystem that integrates advanced tools for community engagement and innovative multichain functionality.

The platform strategically utilizes Telegram's 800 million users and aligns with the TON Foundation's goal to introduce 40% of these users to TON within three to five years. This integration not only expands the potential audience for blockchain projects but also introduces a new demographic to the possibilities of Web3 and decentralized technologies. Community engagement is a cornerstone of XTON, offering tools that foster interaction and education within Telegram. This includes interactive quests and educational materials that help grow and sustain user bases for launching projects. By tapping into an existing, engaged community, XTON enables projects to leverage collective interest and support. XTON distinguishes itself with its multichain functionality, allowing startups to choose their preferred chains for payment and token allocation. This feature not only enhances liquidity flow across blockchains but also provides flexibility in token distribution strategies. Such an approach is pivotal in accommodating the diverse needs of blockchain startups and their potential investors.

Another key aspect of XTON is its focus on customized investor pools through XTON tokens. These tokens grant holders exclusive access to token sales, with benefits that include tailored rules, allocation sizes, and participation chances. This system incentivizes community engagement and investment in the platform, aligning the interests of startups and investors.

In summary, XTON presents an innovative approach to blockchain project launches by integrating Telegram's user base, multichain functionality, and comprehensive support systems.

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 softstack 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 softstack 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

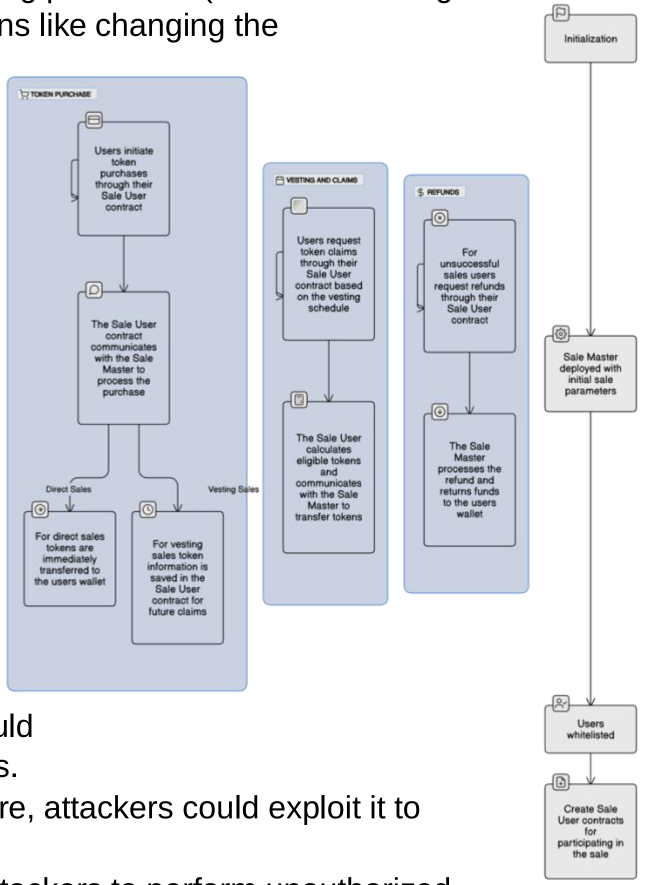
The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability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

File	Fingerprint (MD5)
./core-fc/contracts/constants/constants.fc	58106433a105393e7d3ebc8501f6063d
./core-fc/contracts/constants/fees.fc	13e06449d929afe249fbf4e29e59a74d
./core-fc/contracts/constants/errors.fc	cf6e07987bcf2181bc3b16e6349c02c3
./core-fc/contracts/constants/op-codes.fc	f6053a22687e0121e56bcab0a14d6263
./core-fc/contracts/sale_user.fc	160efeb18662d3f47c28cedf3e5ae741
./core-fc/contracts/logic/utils.fc	3485db29afcf5e68e530f1be020cf0db
./core-fc/contracts/logic/messages.fc	201392be4d1639c42beb0d0e587e61a0
./core-fc/contracts/logic/price_manager.fc	38cf32992a28da46907a940cf4645a64
./core-fc/contracts/imports/stdlib.fc	f54c5dc7eebe7e11f70592b7f1285a18
./core-fc/contracts/sale.tlb	8c1c64e0e8f01ce0b0ac4519c64fcf0b
./core-fc/contracts/sale_master.fc	2f04da8fe8ae986abc83969bfc15c881
./eth-wallet-main/contracts/imports/stdlib.fc	f54c5dc7eebe7e11f70592b7f1285a18
./eth-wallet-main/contracts/errors.fc	de0aefb8c7dbbace1f3c1a2f2896ab7f
./eth-wallet-main/contracts/wallet_eth.fc	ed59aa38e4fe75d47ac6f80e231b6954
./xton-contracts-main/src/XTONSaleV1.sol	3490dfa0d82be85166b762741b1642c4
./xton-contracts-main/src/ISaleV1.sol	8132dbf286e134f9fa6ad3f66efe637d
./xton-contracts-main/src/WhitelistedPoolVerifier.sol	285016053380714b6af38cd64177a870

5.2 Business Logic

Sales Core Contracts (FunC)

- Sale Master Contract: Manages the overall sale process, including initializing sales, handling purchases (direct and through bridges), managing refunds, and processing claims. It also supports administrative functions like changing the admin address, bridge public key, and cap information.
- Sale User Contract: Represents individual users participating in the sale. It supports buying tokens (directly and through bridges), claiming tokens (for vesting sales), and requesting refunds for unsuccessful sales.
- Constants and Errors: Define various constants used across contracts (e.g., flags, sale types, error codes) and specific error codes for different failure conditions.
- Fees and Op-Codes: Specify the fees for different operations and operation codes used for internal messaging between contracts.
- Utilities: Provide utility functions for creating user contracts, calculating contract addresses, and handling jetton transfers.

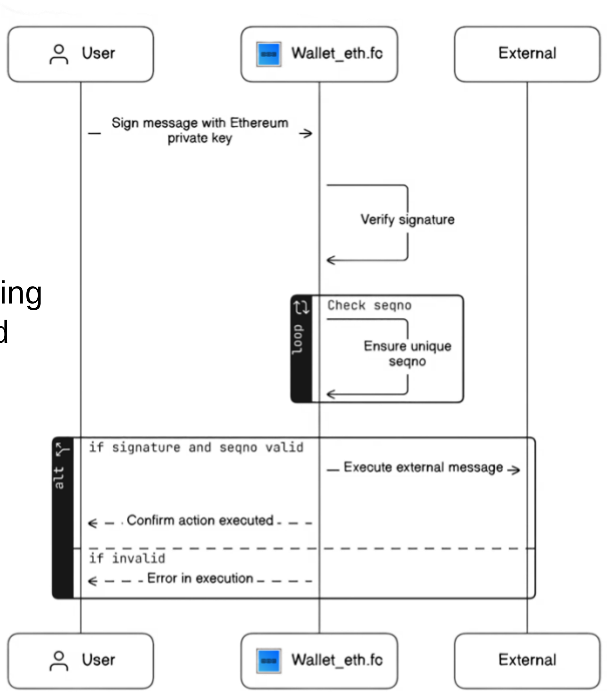

TOKEN PURCHASE
Users initiate token purchases through their Sale User contract
The Sale User contract communicates with the Sale Master to process the purchase
Direct Sales
Vesting Sales
For direct sales tokens are immediately transferred to the users wallet
For vesting sales token information is saved in the Sale User contract for future claims
VESTING AND CLAIMS
Users request token claims through their Sale User contract based on the vesting schedule
The Sale User calculates eligible tokens and communicates with the Sale Master to transfer tokens
REFUNDS
For unsuccessful sales users request refunds through their Sale User contract
The Sale Master processes the refund and returns funds to the users wallet
Initialization
Sale Master deployed with initial sale parameters
Users whitelisted
Create Sale User contracts for participating in the sale

Potential Attack Vectors

- Man-in-the-Middle (MITM) Attacks: Attackers could intercept or manipulate messages between Sale User and Sale Master contracts or between the TON blockchain and EVM chains.
- Replay Attacks: If nonce or timestamp checks are not properly implemented, attackers could replay transaction messages to either double-spend or cause unauthorized token transfers.
- Bridge Vulnerabilities: If the bridge mechanism between TON and EVM chains is not secure, attackers could exploit it to manipulate token purchases or transfers across chains.
- Permission Escalation: Flaws in user authentication or contract permissions could allow attackers to perform unauthorized operations, such as initiating refunds, changing sale parameters, or claiming tokens they are not entitled to.

Wallet Contracts (FunC)

- Signature Verification: The contract includes functionality to recover the signer of a message (similar to Ethereum's ecrecover), ensuring that messages can be securely signed and verified. This is crucial for operations that require authentication, such as transactions or message passing between TON and EVM.
- Sequence Numbers: Utilizing sequence numbers (seqno) for messages prevents replay attacks by ensuring each message is unique and only processed once.
- External Message Execution: The contract can send external messages, potentially allowing for cross-chain interactions or message passing within the TON network based on verified signatures and valid sequence numbers.

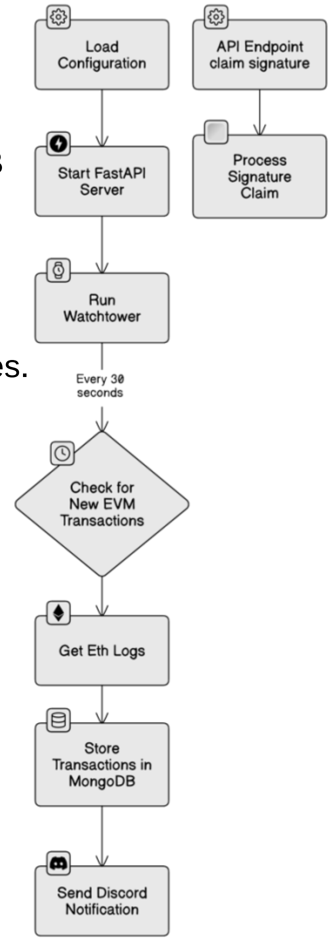


Potential Attack Vectors

- Signature Replay Attacks: Even with sequence numbers, if the sequence number implementation is flawed or if there's a way to manipulate it, attackers might replay messages for unauthorized actions.
- Signature Forgery: If the ecrecover implementation has weaknesses or if there's insufficient validation of the recovered address, attackers might forge signatures to impersonate other users.
- External Message Manipulation: Since the contract can send external messages based on received commands, a vulnerability in message processing could allow attackers to send unauthorized external messages, leading to unintended actions within the TON network or cross-chain.

EVM Watcher (Helios)

- Configuration Loading: The system loads configuration from an .env file, including API tokens, blockchain addresses, and database URI, among others.
- EVM Event Watching: The system uses a Web3Service to listen for specific events on the Ethereum blockchain (e.g., cross chain sale transactions). It filters these events based on predefined criteria (address and event signature).
- Database Interaction: When new transactions are detected, the system stores transaction details in a MongoDB database for later retrieval and analysis.
- Notification: Upon detecting new transactions, the system can send alerts via Discord, notifying interested parties about the transaction details.
- API Endpoints: The system exposes API endpoints, such as /claim_signature, allowing external entities to interact with it, potentially to claim participation in cross chain sales or retrieve whitelisted participation signatures.



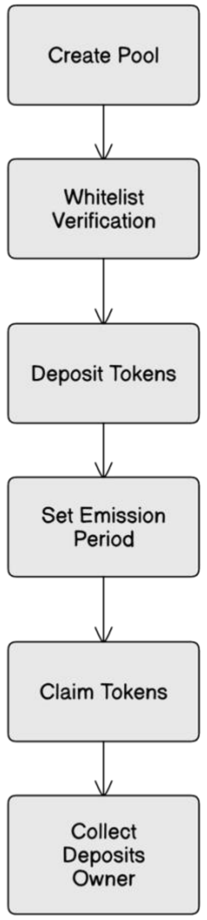
Load Configuration
API Endpoint claim signature
Start FastAPI Server
Process Signature Claim
Run Watchtower
Every 30 seconds
Check for New EVM Transactions
Get Eth Logs
Store Transactions in MongoDB
Send Discord Notification

Potential Attack Vectors

- Replay Attacks: If the system does not adequately validate the uniqueness of transactions or events, attackers could replay old transactions to trigger false alerts or database entries.
- Signature Spoofing: The /claim_signature endpoint could be vulnerable to attacks where an attacker spoofs or forges signatures to falsely claim participation in a sale or whitelist.
- Man-in-the-Middle (MITM) Attacks: Without proper encryption and authentication, data transmitted between the system components (e.g., between the FastAPI server and the MongoDB database or the Discord service) could be intercepted and manipulated by attackers.
- Denial of Service (DoS): Attackers could flood the system's API endpoints with excessive requests, potentially overwhelming the server and causing legitimate requests to be dropped or delayed.
- Database Injection: If the system does not properly sanitize inputs before storing them in the database, attackers could execute database injection attacks, leading to unauthorized data access or manipulation.

XTON EVM Contracts (Solidity)

- Pool Creation: The contract owner can create multiple sale pools, each with specific parameters such as deposit token, deposit limits, sale price, and vesting details.
- Whitelist Verification: Participants in a sale pool must have their participation request signed by a trusted signer. This ensures only whitelisted addresses can deposit tokens.
- Token Deposits: Whitelisted users can deposit ERC20 tokens into a pool within a specified timeframe. Deposits can be made directly by participants or on behalf of others.
- Emissions and Vesting: The contract supports setting emission periods for each pool, defining when and how tokens will be vested and claimable by participants.
- Claiming and Collecting: Participants can claim their vested tokens based on the emission schedule. The contract owner can collect deposited funds after the deposit period ends.
- Cross-Chain Participation: The system allows for cross-chain sales, where participants can buy tokens on one chain (e.g., Ethereum) and claim them on another (e.g., TON).



Potential Attack Vectors

- Signature Spoofing: If the private key of the signer is compromised, attackers could whitelist any address, bypassing the intended restrictions.
- Reentrancy Attacks: Although the contract uses ReentrancyGuard, it's essential to ensure all external calls are non-reentrant, especially in functions like claim and depositFor, to prevent unexpected behavior.
- Front-Running: Malicious actors could observe pending transactions (e.g., deposit or claim) and attempt to front-run them for profit, especially in scenarios involving price-sensitive actions.
- DoS with Block Gas Limit: By creating many pools or making excessive deposits, an attacker could potentially make the contract's functions require more gas than the block gas limit, preventing further interactions.
- Chain Risks: The security of cross-chain operations depends on the bridge mechanism used. Vulnerabilities in the bridge or incorrect handling of cross-chain messages could lead to loss or duplication of tokens.

Throughout the manual testing stage, potential attack vectors were rigorously evaluated, underscoring the importance of continuously addressing these vulnerabilities in future development phases to enhance project security.

5.3 Source Unites in Scope

Source: https://github.com/tokenova-fi/xton-contracts

Total : 3 files, 323 codes, 72 comments, 84 blanks, all 479 lines

language	files	code	comment	blank	total
Solidity	3	323	72	84	479

Source: https://github.com/tokenova-fi/evm-watcher

Total : 14 files, 398 codes, 5 comments, 135 blanks, all 538 lines

language	files	code	comment	blank	total
Python	14	398	5	135	538

Source: https://github.com/tokenova-fi/core-fc

Total : 10 files, 1062 codes, 326 comments, 256 blanks, all 1644 lines

language	files	code	comment	blank	total
FunC	10	1,062	326	256	1,644

Source: https://github.com/tokenova-fi/eth-wallet

Total : 3 files, 254 codes, 317 comments, 150 blanks, all 721 lines

language	files	code	comment	blank	total
FunC	3	254	317	150	721

6. Scope of Work

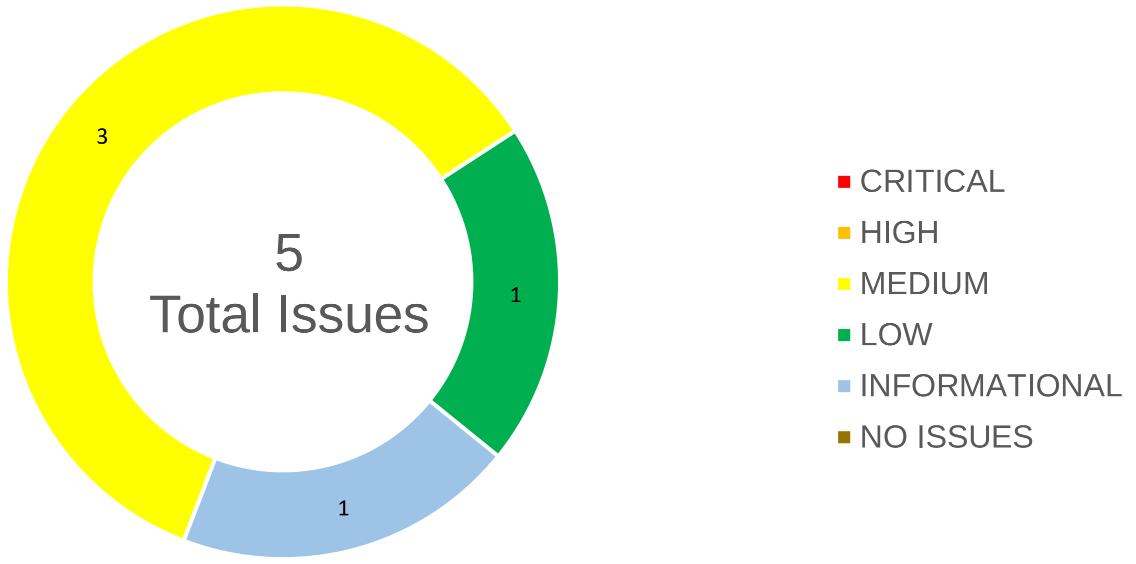
The XTON Development Team provided us with the files that needs to be tested. The scope of the audit is the XTON EVM and TON Core contracts.

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

1. Compliance with Best Practices: Verification that the contracts adhere to established smart contract best practices, with a focus on preventing common vulnerabilities such as reentrancy attacks, overflow/underflow issues, and adherence to the check-effects-interactions pattern.
2. Authentication and Authorization Security: Assessment of the implementation of role-based access controls and permissions to ensure only authorized addresses can execute sensitive functions, mitigating the risk of unauthorized actions and permission escalation.
3. Nonce and Timestamp Security: Examination of nonce and timestamp utilization to protect against replay attacks, ensuring transaction messages are processed only once and cannot be replayed for double-spending or unauthorized token transfers.
4. Bridge Mechanism Integrity: Scrutiny of the security of the bridge mechanisms between TON and EVM chains to prevent exploitation that could lead to manipulation of token purchases or transfers across chains.
5. External Message and Signature Verification: Evaluation of the contracts' mechanisms for external message execution and signature verification, specifically looking for vulnerabilities that could allow for signature forgery or manipulation of external messages leading to unauthorized cross-chain interactions.

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	Partial Execution of Transactions	MEDIUM	ACKNOWLEDGED
6.2.2	Inability to Recover ETH in recoverAnyTokens Function	MEDIUM	ACKNOWLEDGED
6.2.3	Precision Loss in claimableTokens Function	MEDIUM	ACKNOWLEDGED
6.2.4	Lack of Input Validation for _tg_address in depositFor Function	LOW	ACKNOWLEDGED
6.2.5	Inconsistent Solidity Pragma Directives	INFORMATIONAL	ACKNOWLEDGED

6.2 Manual and Automated Vulnerability Test

CRITICAL ISSUES

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

HIGH ISSUES

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

MEDIUM ISSUES

During the audit, softstack‘s experts found **three Medium issues** in the code of the smart contract.

6.2.1 Partial Execution of Transactions

Severity: MEDIUM
Status: ACKNOWLEDGED
File(s) affected: sale_user.fc
Update: We believe that the provided snippet is not related to the issue, but we investigated the contract for the mentioned issue carefully and think that this kind of partial execution is not possible due to the design we have: The only state change occurring in sale-user contract is related to the state lock and prevention of signature replays. When we enter “op::try_to_buy” and “op::try_to_buy_bridge” in master contract, the state changes will not be integral if and only if one of these two messages fail. There may be a loss of consistency if jetton-send fails due to low balance, and this should be ensured at the beginning of the sale to prevent it from happening. Our team will look more closely at this issue and try to develop a patch.

Attack / Description	The smart contract handles operations including op::try_to_buy for direct token purchases and op::try_to_buy_bridge for bridge token purchases. In both scenarios, the token amount is deducted or accounted for before the confirmation of successful transaction completion. This premature deduction poses a risk if subsequent steps fail after this initial modification, potentially leading to tokens being deducted without completing the transfer or associated operations. Technical Analysis:

	• Direct and Bridge Token Purchases: In the recv_internal function, operations related to token purchases deduct token amounts prior to ensuring the success of the entire transaction sequence. If a failure occurs in subsequent steps, such as message sending or external contract interactions failing, the initial deduction remains, creating a discrepancy in the token balance without achieving the intended outcome. • Lack of Revert Mechanism: The smart contract does not implement a mechanism to revert changes if a part of the transaction fails after the initial state modifications. This absence can lead to scenarios where initial deductions or operations are not reversed if later steps fail, leading to partial execution.
Code	Line 133 – 147:

```
() recv_internal(int my_balance, int msg_value, cell in_msg_full, slice in_msg_body) impure {
    load_data();
    slice cs = in_msg_full.begin_parse();
    int flags = cs~load_uint(4);
    slice sender_address = cs~load_msg_addr();
    if (flags & 1) {
        in_msg_body~skip_bits(32);
        int op = in_msg_body~load_uint(32);
        if (op == op::try_to_buy) {
            messages::send_simple_non_bounceable(storage::user_address, 0, null(), null(), mode::remaining_amount);
            storage::locked? = false;
            save_data();
            return ();
        }
    }
```

Result/Recommendation	Proposed Solutions: Staging Changes: Implement a staging mechanism for changes, where state modifications are temporarily held until the entire transaction sequence is verified to succeed. Only then are changes committed atomically to the contract state. Explicit Revert or Compensation Logic: In cases where later steps in a transaction fail, incorporate explicit logic to revert earlier changes or provide compensation logic to restore the initial state. This can be achieved through custom error handling and state management logic.

6.2.2 Inability to Recover ETH in recoverAnyTokens Function

Severity: MEDIUM
Status: ACKNOWLEDGED
File(s) affected: XTONSaleV1.sol

Attack / Description	The recoverAnyTokens function is designed to allow the recovery of excess tokens from the contract. However, it does not account for the recovery of Ether (ETH), potentially leading to trapped ETH within the contract. The recoverAnyTokens function iterates through pools to calculate the amount of each token that is reserved and should not be withdrawn. It then transfers the excess tokens to the owner. This function, however, only supports ERC20 tokens and does not include logic to handle or recover Ether, which might also be sent to or stored in the contract.

Code	Line 210 - 228:
	
	`function recoverAnyTokens(address token) external onlyOwner {`
	`    require(token != address(0), "ERR_ZERO_ADDRESS");`

```
  uint256 reservedTokens = 0;
  uint256 pids = poolInfo.length;
  for (uint256 pid = 0; pid < pids; pid++) {
    PoolInfo memory pool = poolInfo[pid];
    if (token == pool.depositToken && !pool.collected) {
      reservedTokens += pool.depositedAmount;
    } else if (token == launchToken) {
      uint256 totalSold = pool.depositedAmount * launchTokenPrecision / pool.price;
      reservedTokens += totalSold - pool.totalClaimed;
    }
  }
  uint256 currentTokenBalance = IERC20(token).balanceOf(address(this));
  require(currentTokenBalance > reservedTokens, "ERR_NO_EXCESS_TOKENS");
  uint256 excessTokens = currentTokenBalance - reservedTokens;
  IERC20(token).safeTransfer(msg.sender, excessTokens);
}
```

Result/Recommendation

Implement an additional function or modify the existing recoverAnyTokens function to allow the recovery of ETH. This could involve checking the contract's ETH balance and enabling a transfer of ETH to the owner's address. For example:

```
if(address(this).balance > 0) {
    (bool success, ) = (msg.sender).call{value: address(this).balance}("");
    require(success, "ERR_TRANSFER_ETH");
```

	` }` this prevents funds from being unintentionally trapped.

6.2.3 Precision Loss in claimableTokens Function

Severity: MEDIUM
Status: ACKNOWLEDGED
File(s) affected: XTONSaleV1.sol

Attack / Description	The claimableTokens function calculates the amount of launch tokens a user can claim based on their deposited amount. This calculation involves converting the deposited amount into an equivalent amount of launch tokens based on the pool's price. The issue arises in the line where division occurs before multiplication, leading to potential precision loss: uint256 boughtAmount = user.deposited * launchTokenPrecision / pool.price; Solidity performs integer division, which truncates the result. Consequently, dividing before multiplying can cause a loss of precision, especially when pool.price is significantly larger than user.deposited. This could lead to users receiving fewer launch tokens than they are entitled to based on their deposited amount, affecting the fairness of the token distribution process. Consider a scenario where the user's deposited amount (user.deposited) is much smaller compared to the pool's price (pool.price), and the calculation of boughtAmount involves a division that truncates the result to a lower value before it is multiplied by launchTokenPrecision. Users might end up receiving fewer launch tokens than expected.

Code	Line 361 – 388 :
	
	`function claimableTokens(uint256 _pid, address _user) public view returns (uint256 claimableAmount) {`
	`    require(_pid < poolInfo.length, "ERR_POOLID");`
	`    PoolInfo memory pool = poolInfo[_pid];`

	```
	  if (pool.targetChain != 0) {
	    // cross-chain pools cannot be claimed, just claim with signature is possible on the target chain!
	    return 0;
	  }
	  if (pool.emissionPeriod.startOfEmissions == 0
	      || pool.emissionPeriod.startOfEmissions > block.timestamp) {
	     return 0;
	  }
	  UserInfo memory user = userInfo[_pid][_user];
	  uint256 boughtAmount = user.deposited * launchTokenPrecision / pool.price;
	  uint256 instantUnlockedAmount = boughtAmount * pool.instantUnlockRatio / MAX_BASIS_POINTS;
	  uint256 vestedAmount = boughtAmount - instantUnlockedAmount;
	  uint256 startOfEmissions = pool.emissionPeriod.vestingCliffAccrues
	                             ? pool.emissionPeriod.startOfEmissions
	                             : pool.emissionPeriod.endOfVestingCliff;
	  uint256 totalEmissionSeconds = pool.emissionPeriod.endOfEmissions - startOfEmissions;
	  uint256 emissionPassed = (block.timestamp < pool.emissionPeriod.endOfEmissions)
	                           ? (block.timestamp > startOfEmissions ? block.timestamp - startOfEmissions : 0)
	                           : totalEmissionSeconds;
	  uint256 vestedUnlockedAmount = (block.timestamp >= pool.emissionPeriod.endOfVestingCliff)
	                                 ? vestedAmount * emissionPassed / totalEmissionSeconds
	                                 : 0;
	  uint256 unlockedAmount = instantUnlockedAmount + vestedUnlockedAmount;
	  claimableAmount = unlockedAmount - user.claimed;
	}
	```

Result/Recommendation	To avoid precision loss, it's recommended to adjust the arithmetic operations to perform multiplication before division. This can be done by reordering operations and adjusting the calculation of instantUnlockedAmount as follows:
	
	uint256 boughtAmount = user.deposited * launchTokenPrecision;
	uint256 instantUnlockedAmount = boughtAmount * pool.instantUnlockRatio / (MAX_BASIS_POINTS * pool.price);

LOW ISSUES

During the audit, softstack‘s experts found **one Low issue** in the code of the smart contract

6.2.4 Lack of Input Validation for _tg_address in depositFor Function

Severity: LOW
Status: ACKNOWLEDGED
Code: CWE-20: Improper Input Validation
File(s) affected: XTONSaleV1.sol
Update: Acknowledged, but a little bit not straightforward to address. The reason is that tg_address may belong to different chains (non-EVM, like TON for our case), which introduces cost overhead for validation, also making it not flexible enough to use for other chains. We choose to heavily validate addresses in our dApp that is using this function.

Attack / Description	The depositFor function in the XTONSaleV1 smart contract is designed to allow users to deposit tokens for participating in a token sale. This function accepts several parameters, including _tg_address, which is intended for specifying a target address for cross-chain participation. However, the function does not perform any validation on the _tg_address input. This oversight could potentially lead to various issues, including but not limited to, operational inefficiencies and incorrect cross-chain transactions.

Code	Line 258 - 265:
	`function deposit (uint256 _pid,`
	`                 uint256 _amount,`

	`string calldata _salt,`
	`bytes memory _signature,`
	`string calldata _tg_address)`
	`external {`
	`depositFor(_pid, _amount, msg.sender, _salt, _signature, _tg_address);`
	`}`

Result/Recommendation	Implement input validation for the _tg_address parameter within the depositFor function. The validation logic should check for the correct format and other criteria relevant to the intended use case of the _tg_address. For Ethereum addresses, common validations include checking the address length and applying checksum validation. For cross-chain scenarios, additional validations may be required based on the target blockchain's address format.

INFORMATIONAL ISSUES

During the audit, softstack‘s experts found **one Informational issue** in the code of the smart contract.

6.2.5 Inconsistent Solidity Pragma Directives

Severity: INFORMATIONAL
Status: ACKNOWLEDGED
File(s) affected: WhitelisedPoolVerifier.sol, XTONSaleV1.sol, ISaleV1.sol

Attack / Description	The use of different Solidity pragma versions (^0.8.13, ^0.8.19, ^0.8.20) across various contracts and libraries introduces a risk of compatibility issues. Solidity frequently releases updates that include new features, optimizations, and important security fixes. The specified pragma versions suggest that different contracts within the suite might not be taking full advantage of these updates. Furthermore, using multiple versions can complicate the deployment process and make the codebase harder to maintain.
Code	WhitelistedPoolVerifier.sol uses ^0.8.13 XTONSaleV1.sol uses ^0.8.13

	ISaleV1.sol uses ^0.8.19 OpenZeppelin contracts (e.g., Ownable.sol, IERC20.sol, SafeERC20.sol, etc.) use ^0.8.20
Result/Recommendation	Aim to use a single, consistent Solidity version across the entire contract suite. This standardization helps ensure that all contracts are compiled with the same compiler version, reducing the risk of unexpected behavior and making the codebase easier to manage. If possible, upgrade to the latest stable version of Solidity to benefit from the latest language features and security improvements.

6.3 Verify Claims

6.3.1 Compliance with Best Practices: Verification that the contracts adhere to established smart contract best practices, with a focus on preventing common vulnerabilities such as reentrancy attacks, overflow/underflow issues, and adherence to the check-effects-interactions pattern.
Status: tested and verified ✓

6.3.2 Authentication and Authorization Security: Assessment of the implementation of role-based access controls and permissions to ensure only authorized addresses can execute sensitive functions, mitigating the risk of unauthorized actions and permission escalation.
Status: tested and verified ✓

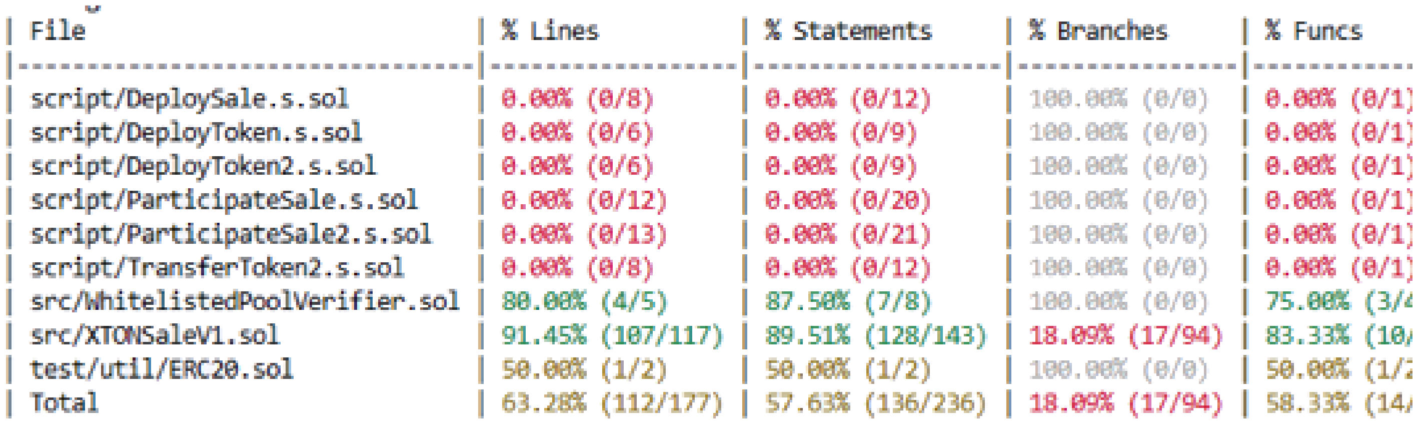
6.3.3 Nonce and Timestamp Security: Examination of nonce and timestamp utilization to protect against replay attacks, ensuring transaction messages are processed only once and cannot be replayed for double-spending or unauthorized token transfers.
Status: tested and verified ✓

6.3.4 Bridge Mechanism Integrity: Scrutiny of the security of the bridge mechanisms between TON and EVM chains to prevent exploitation that could lead to manipulation of token purchases or transfers across chains.
Status: tested and verified ✅

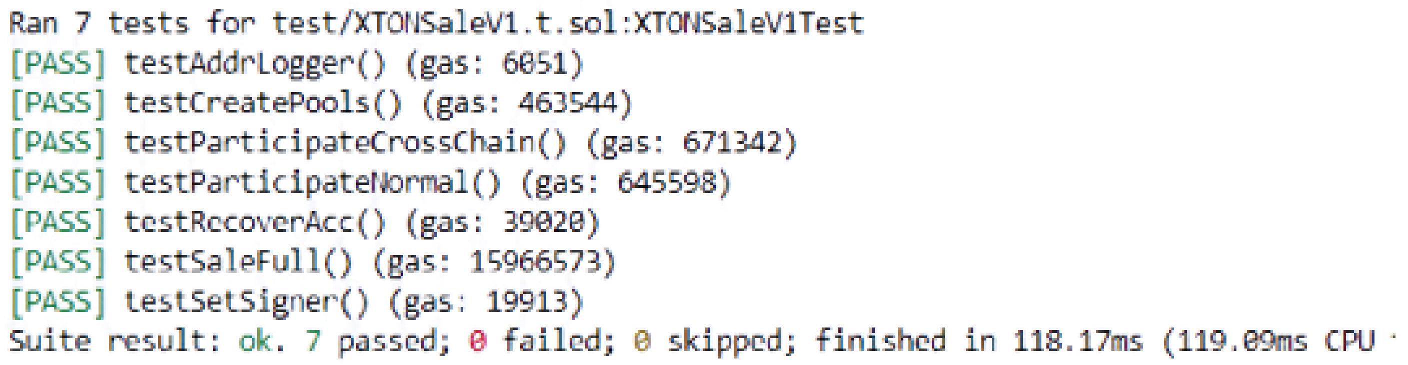
6.3.5 External Message and Signature Verification: Evaluation of the contracts' mechanisms for external message execution and signature verification, specifically looking for vulnerabilities that could allow for signature forgery or manipulation of external messages leading to unauthorized cross-chain interactions.
Status: tested and verified ✅

6.4 Unit Tests

XTON EVM Contracts (Solidity)

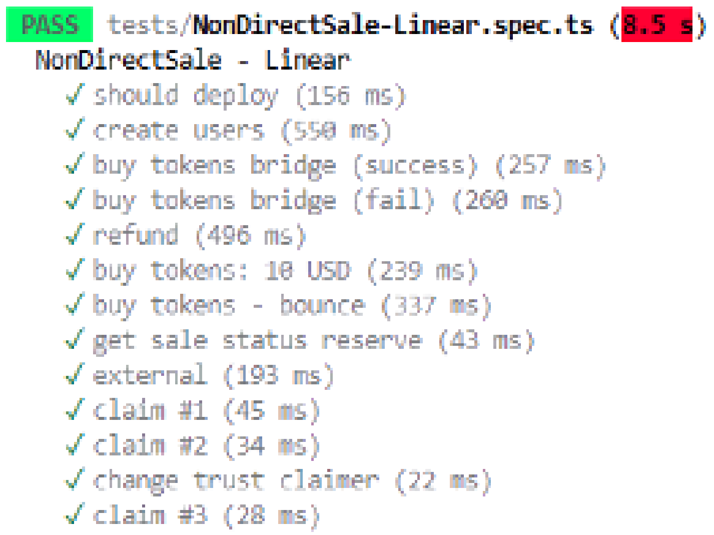


```
| File                            | % Lines          | % Statements     | % Branches      | % Funcs
|---------------------------------|------------------|------------------|-----------------|------------
| script/DeploySale.s.sol         | 0.00% (0/8)      | 0.00% (0/12)     | 100.00% (0/0)   | 0.00% (0/1)
| script/DeployToken.s.sol        | 0.00% (0/6)      | 0.00% (0/9)      | 100.00% (0/0)   | 0.00% (0/1)
| script/DeployToken2.s.sol       | 0.00% (0/6)      | 0.00% (0/9)      | 100.00% (0/0)   | 0.00% (0/1)
| script/ParticipateSale.s.sol    | 0.00% (0/12)     | 0.00% (0/20)     | 100.00% (0/0)   | 0.00% (0/1)
| script/ParticipateSale2.s.sol   | 0.00% (0/13)     | 0.00% (0/21)     | 100.00% (0/0)   | 0.00% (0/1)
| script/TransferToken2.s.sol     | 0.00% (0/8)      | 0.00% (0/12)     | 100.00% (0/0)   | 0.00% (0/1)
| src/WhitelistedPoolVerifier.sol | 80.00% (4/5)     | 87.50% (7/8)     | 100.00% (0/0)   | 75.00% (3/
| src/XTONSaleV1.sol              | 91.45% (107/117) | 89.51% (128/143) | 18.09% (17/94)  | 83.33% (10/
| test/util/ERC20.sol             | 50.00% (1/2)     | 50.00% (1/2)     | 100.00% (0/0)   | 50.00% (1/
| Total                           | 63.28% (112/177) | 57.63% (136/236) | 18.09% (17/94)  | 58.33% (14/
```

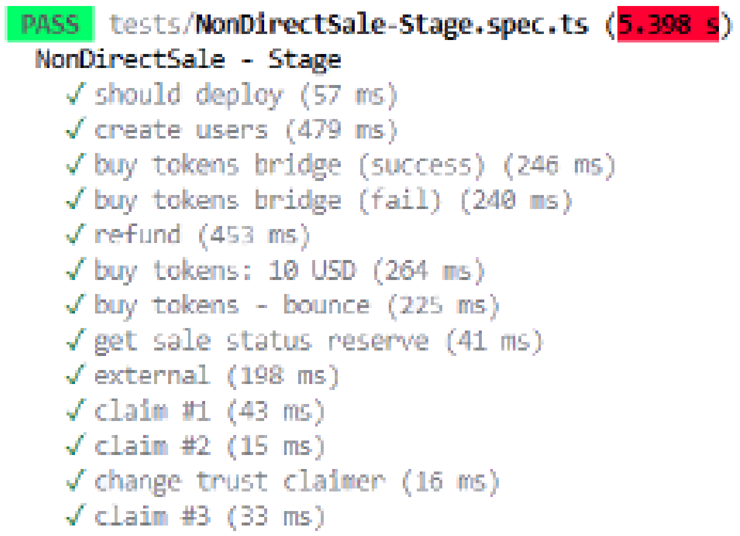


```
Ran 7 tests for test/XTONSaleV1.t.sol:XTONSaleV1Test
[PASS] testAddrLogger() (gas: 6051)
[PASS] testCreatePools() (gas: 463544)
[PASS] testParticipateCrossChain() (gas: 671342)
[PASS] testParticipateNormal() (gas: 645598)
[PASS] testRecoverAcc() (gas: 39020)
[PASS] testSaleFull() (gas: 15966573)
[PASS] testSetSigner() (gas: 19913)
Suite result: ok. 7 passed; 0 failed; 0 skipped; finished in 118.17ms (119.09ms CPU
```

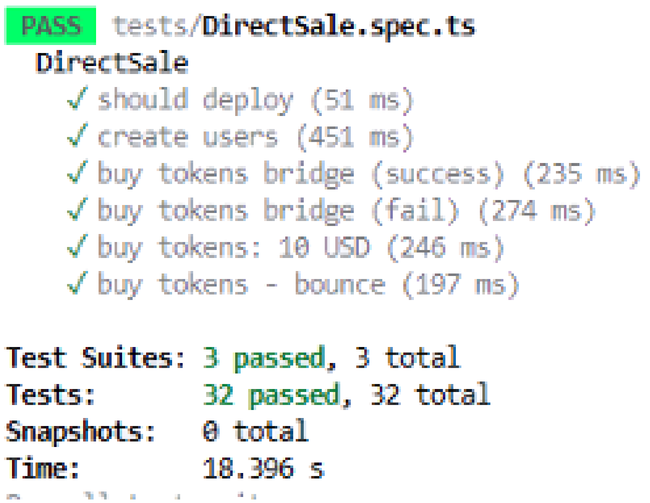
XTON Sales Core Contracts (FunC)



```
PASS tests/NonDirectSale-Linear.spec.ts (8.5 s)
  NonDirectSale - Linear
    ✓ should deploy (156 ms)
    ✓ create users (550 ms)
    ✓ buy tokens bridge (success) (257 ms)
    ✓ buy tokens bridge (fail) (260 ms)
    ✓ refund (496 ms)
    ✓ buy tokens: 10 USD (239 ms)
    ✓ buy tokens - bounce (337 ms)
    ✓ get sale status reserve (43 ms)
    ✓ external (193 ms)
    ✓ claim #1 (45 ms)
    ✓ claim #2 (34 ms)
    ✓ change trust claimer (22 ms)
    ✓ claim #3 (28 ms)
```



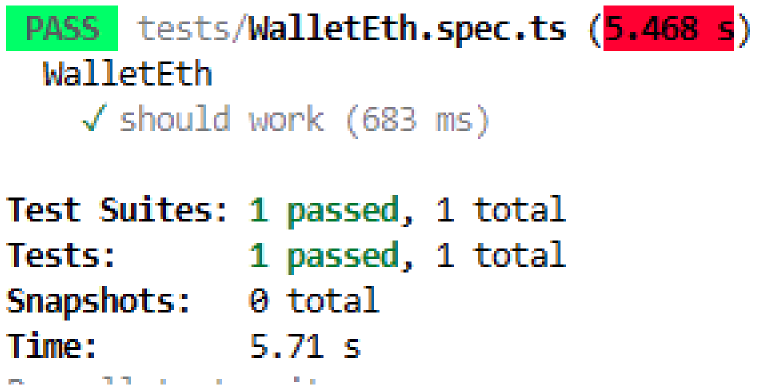
```
PASS tests/NonDirectSale-Stage.spec.ts (5.398 s)
  NonDirectSale - Stage
    ✓ should deploy (57 ms)
    ✓ create users (479 ms)
    ✓ buy tokens bridge (success) (246 ms)
    ✓ buy tokens bridge (fail) (240 ms)
    ✓ refund (453 ms)
    ✓ buy tokens: 10 USD (264 ms)
    ✓ buy tokens - bounce (225 ms)
    ✓ get sale status reserve (41 ms)
    ✓ external (198 ms)
    ✓ claim #1 (43 ms)
    ✓ claim #2 (15 ms)
    ✓ change trust claimer (16 ms)
    ✓ claim #3 (33 ms)
```



```
PASS  tests/DirectSale.spec.ts
  DirectSale
    ✓ should deploy (51 ms)
    ✓ create users (451 ms)
    ✓ buy tokens bridge (success) (235 ms)
    ✓ buy tokens bridge (fail) (274 ms)
    ✓ buy tokens: 10 USD (246 ms)
    ✓ buy tokens - bounce (197 ms)

Test Suites: 3 passed, 3 total
Tests:       32 passed, 32 total
Snapshots:   0 total
Time:        18.396 s
```

Wallet Contracts (FunC)



```
PASS  tests/WalletEth.spec.ts (5.468 s)
  WalletEth
    ✓ should work (683 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        5.71 s
```

7. Executive Summary

Two independent softstack experts performed an unbiased and isolated audit of the smart contract codebase provided by the XTON Development Team. 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 6 issues, classified as follows:
- No critical issues were found.
- One high severity issue was found.
- Three medium severity issues were found.
- One low severity issue was discovered
- One informational issue was identified

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. We advise the XTON Development Team to implement the recommendations to further enhance the code's security and readability.

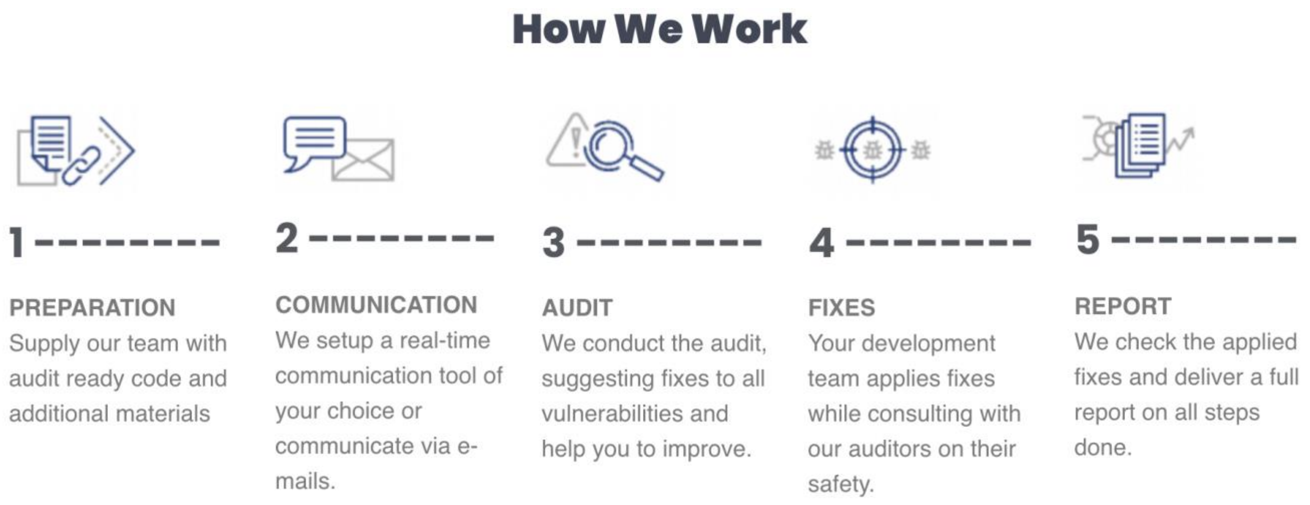
8. About the Auditor

Established in 2017 under the name Chainsulting, and rebranded as softstack GmbH in 2023, softstack has been a trusted name in Web3 Security space. Within the rapidly growing Web3 industry, softstack provides a comprehensive range of offerings that include software development, cybersecurity, and consulting services. Softstack's competency extends across the security landscape of prominent blockchains like Solana, Tezos, TON, Ethereum and Polygon. The company is widely recognized for conducting thorough code audits aimed at mitigating risk and promoting transparency.

The firm's proficiency lies particularly in assessing and fortifying smart contracts of leading DeFi projects, a testament to their commitment to maintaining the integrity of these innovative financial platforms. To date, softstack plays a crucial role in safeguarding over $100 billion worth of user funds in various DeFi protocols.

Underpinned by a team of industry veterans possessing robust technical knowledge in the Web3 domain, softstack offers industry-leading smart contract audit services. Committed to evolving with their clients' ever-changing business needs, softstack's approach is as dynamic and innovative as the industry it serves.

Check our website for further information: https://softstack.io



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.