Smart Contract Security Audit Report

For

X-star

Date Issued: Mar.19, 2023

Version: v2.0

Confidentiality Level: Public

Contents

1 Abstract

This report was prepared for X-star smart contract to identify issues and vulnerabilities in its smart contract source code. A thorough examination of X-star smart contracts was conducted through timely communication with X-star, static analysis using multiple audit tools and manual auditing of their smart contract source code.

The audit process paid particular attention to the following considerations.

- A thorough review of the smart contract logic flow
- Assessment of the code base to ensure compliance with current best practice and industry standards
- Ensured the contract logic met the client's specifications and intent
- Internal vulnerability scanning tools tested for common risks and writing errors
- Testing smart contracts for common attack vectors
- Test smart contracts for known vulnerability risks
- Conduct a thorough line-by-line manual review of the entire code base

As a result of the security assessment, issues ranging from critical to informational were identified. We recommend that these issues are addressed to ensure a high level of security standards and industry practice. The recommendations we made could have better served the project from a security perspective.

- Enhance general coding practices to improve the structure of the source code.
- Provide more comments for each function to improve readability.
- Provide more transparency of privileged activities once the agreement is in place.

2 Overview

2.1 Project Summary

Project Summary	Project Information
Name	X-star
Start date	Mar.8, 2024
End date	Mar.19, 2024
Contract type	Token,NFT,DeFi
Language	Solidity
Code	X-star-contract-main.zip

2.2 Report HASH

Name	HASH
X-star-contract-main.zip	BAF4213E25A2E5B99A43799EF673AC0485E8BE4D15E83990FCCFED19893AFBF7

3 Project contract details

3.1 Contract Overview

XToken.sol

This contract is an ERC20 token contract that inherits OpenZeppelin's ERC20, Ownable and ERC20Burnable contracts. This token has some special features such as buying and selling fees, burning mechanism, and integration with PancakeSwap exchange.

Main features and functions:

Handling fee: This contract sets a handling fee for buying and selling operations. The purchase fee (buyFee) and sales fee (sellFee) can be set by the contract owner. Part of the sales fee can be set to be burned (sellBurnFee).

Trading Pairs and Liquidity: The contract integrates with PancakeSwap, allowing added liquidity and trading. The pair variable stores the PancakeSwap trading pair address related to the token.

Protection mechanism: The contract contains two mappings, isBlockedOf and isGuardedOf, which are used to prevent certain addresses from conducting transactions and protect certain addresses from handling fees respectively.

Blacklist: Through the addBlocked and removeBlocked functions, the contract owner can add or remove blacklist addresses, and blacklist addresses cannot conduct transactions.

Automatic liquidity provision: The contract's built-in mechanism automatically adds liquidity to PancakeSwap trading pairs through transaction fees.

Burning mechanism: This token implements a burning mechanism that allows tokens to be destroyed through the _burn function, reducing the total supply.

Start time: The startTime variable is used to track the time when the contract starts activity. Based on this time, the contract executes specific logic.

CardNFT.sol

This contract is an ERC721 NFT contract based on the OpenZeppelin upgraded version contract. It integrates a series of ERC721 function extensions, including enumerability (ERC721EnumerableUpgradeable), URI storage (ERC721URIStorageUpgradeable) and burnability (ERC721BurnableUpgradeable). In addition, the contract also introduces custom permission control (PermissionControl) and random number generator (IRandom) to enhance its functionality.

Key features and functionality

Permission control: Through PermissionControl and AccessControlUpgradeable, a role-based permission control mechanism is provided for contract operations.

NFT Mining: Provides safeMint and minInit methods, allowing the creation of new NFTs. In particular, the minInit function requires MANAGER_ROLE permission, while safeMint seems to be open to all users.

Random number generation: Use the random instance of the IRandom interface to generate a random number when minting an NFT, which is used to determine the ID of the new NFT.

URI Management: Allows setting the base URL (baseUrl) and generating and setting the URI for each NFT when minting.

DynamicFarm.sol

The contract DynamicFarm is a dynamic farm contract for decentralized finance (DeFi) scenarios, used to manage the contributions of different users in DeFi projects and distribute rewards. Add security via PermissionControl and ReentrancyGuardUpgradeable, and use the SafeERC20Upgradeable library to handle ERC20 token operations.

Key features and functionality:

Dynamic reward distribution: The contract supports different types of reward distribution, including preaching rewards, low node rewards and high node rewards, and uses different computing power (powerA, powerB, powerT, powerN) to calculate the rewards that users deserve.

N value and node level: By maintaining the total computing power, average computing power and number of users of the node, and setting the N value threshold for low nodes and high nodes, users are dynamically allocated to nodes of different levels to obtain rewards in different proportions. .

Reward accumulation and collection: Users can accumulate rewards in different periods (Epoch) and claim these rewards when they meet the conditions.

Permission management: The contract implements role-based permission control. Only specific roles (DEFAULT_ADMIN_ROLE, MANAGER_ROLE) can perform sensitive operations, such as setting cycle controllers, adjusting N value thresholds, etc.

EpochController.sol

The contract EpochController is designed as a period controller that controls and distributes token rewards. It distributes rewards by interacting with different pools (such as static pools, alliance pools, dynamic farms, etc.). Additionally, it supports token buyback and destruction through PancakeSwap to increase token value.

Main functions and features

Periodic reward distribution: Check regularly (such as daily) whether reward distribution is possible, and automatically distribute rewards when conditions are met.

Dynamically set parameters: Parameters such as the ratio of removing liquidity (removeRadio) and the repurchase ratio (xRadio) can be adjusted by the contract owner or specific roles.

Token repurchase and destruction: Part of the rewards are used to repurchase and destroy tokens, aiming to reduce the circulating supply and increase the value of tokens.

Interact with multiple pools: By interacting with contracts such as static pools, alliance pools, and dynamic farms, reward distribution in different scenarios can be achieved.

Family.sol

The contract Family is used to build a tree structure representing family relationships, including root addresses, parent-child relationships between addresses, and depth information. This contract implements permission control through the PermissionControl module, allowing the management and query of the family tree structure.

Main functions and features

Initialization: Set the root address and associated ICard contract address through the initialize method.

Establish relationships: makeRelation allows users to mint a new identity for an NFT that has not yet been minted and add it to the family tree, or establish a superior-subordinate relationship for an existing identity.

Query family members: The getForefathers and childrenOf functions allow querying the direct superiors and direct subordinates of an address.

Genesis.sol

Contract Genesis is an initialization contract designed to allow users to purchase computing power with USDT through the genesis subscription event. The contract is deployed on the Ethereum smart contract platform and provides a decentralized way to initiate the Initial Token Distribution (IDO) of a new project.

The main function

Initialization settings: Set the USDT address, static pool address and USDT receiving address through the initialize function.

Subscription startup: Start the IDO process through the start function, which can only be called by users with specific roles.

Purchase of computing power: Through the subscribe function, users can spend USDT to purchase computing power. Each identity NFT address (cardAddr) can only be purchased once in each cycle (epoch), and the purchase amount has an upper and lower limit.

Calculation of computing power: The getPowerByUSDT function calculates the computing power that should be obtained based on the amount of USDT invested and the current period. The computing power decreases over time, and purchases are supported in the first three cycles.

LeaguePool.sol

The contract LeaguePool is a smart contract used to distribute rewards. It uses Initializable, PermissionControl, and ReentrancyGuardUpgradeable, which are upgrade and security modules from OpenZeppelin. The goal of this contract is to manage the distribution of reward tokens to incentivize participants for their contributions.

Main functions and features

Initialization settings: Set the reward token, identity card contract address and time node epoch contract address through the initialize function.

Distribute rewards: The distribute function allows the contract to receive notifications from the epoch contract to update the reward distribution pool. This function calculates the reward (accTokenPerShare) for each computing power based on the current total computing power (totalPower) and the new reward.

Receive rewards: The takeReward function allows users to receive corresponding rewards based on their share of computing power.

NewPool.sol

Contract NewPool is a contract used for new user reward distribution. It inherits Initializable, PermissionControl, and ReentrancyGuardUpgradeable contracts provided by OpenZeppelin for upgrades, permission management, and prevention of reentrancy attacks.

The main function

Initialization: Set the addresses of reward token, card contract and time node epoch contract through the initialize method.

Deposit (computing power increase): The deposit method allows specific characters to deposit computing power into the contract, which will affect the user's total computing power in the current cycle and update the prize pool information.

Earning query: The earn method allows users to query the rewards they have accumulated since the last time they withdrawn rewards.

Withdraw rewards: The takeReward method allows users to withdraw their rewards.

Distribute rewards: The distribute method is used to distribute rewards to users at the end of each period. This method is called by the epoch contract and allocates rewards according to the user's computing power.

ProxyRouter.sol

The contract ProxyRouter is a proxy contract that helps users add liquidity through a specific router contract. It inherits Initializable and PermissionControl and uses the SafeERC20Upgradeable library to safely interact with ERC20 tokens.

The main function

Initialization: Set the router's address through the initialize method.

Add liquidity: The addLiquidity method allows users to add tokens tokenA and tokenB to the liquidity pool specified by the router through the proxy contract. During method execution, the required tokens will be transferred from the user address to the ProxyRouter contract, and then the router will be approved to take these tokens from the ProxyRouter to add liquidity.

Random.sol

The contract Random is used to generate random numbers and has permission control capabilities. It inherits PermissionControl, ensuring a certain degree of permission management.

The main function

Initialization: Initialize the contract in the initialize method and set an internal random seed _internalRandomSeed.

Generate random numbers: The seed method generates random numbers based on the current timestamp, the internal seed, and an auto-incremented seed_seed. This approach attempts to increase randomness by combining multiple variables.

Update seed: The update method allows updating the internal seed_seed, incrementing it with each call, thus introducing changes in subsequent random numbers generated.

RemoteControl.sol

The contract RemoteControl provides a mechanism that allows remote activation of the functions of other contracts through permission control, while also handling tasks such as adding token liquidity. It inherits Initializable for initialization, PermissionControl for permission management, and ReentrancyGuardUpgradeable to prevent reentrancy attacks.

The main function

Initialization: The initialize method is used to set the initial state of the contract, including the token address, USDT address, period (epoch) contract address, LP receiver address and router address. It also sets up the router's authorization for the relevant tokens.

Set Genesis contract: The setGenesis method allows administrators to set the genesis contract address.

Start the Genesis contract: The startGenesis method allows users with MANAGER_ROLE permissions to start the genesis contract.

Start the Epoch contract: The startEpoch method handles adding liquidity to the token and USDT and starting the epoch contract. It first checks if there is enough USDT balance, then adds liquidity to two different receiver addresses through router and pancakeRouter, and finally transfers the tokens to the lpReceiver address and starts the epoch contract.

Robot.sol

Contract Robot is designed as an automated tool for handling token swaps and liquidity additions. It inherits Initializable for contract initialization and PermissionControl for permission management. Below is a detailed analysis and risk assessment of the contract:

Functional Analysis

Initialization: The initialize method initializes the contract and sets the token address, USDT address, exchange routing address and payee address. At the same time, unlimited quotas are authorized to the router, allowing it to exchange and add liquidity from the Robot contract address.

Receiving ETH: Through the receive function, the contract can receive ETH, but since there is no function to directly process ETH in the contract, this function may be redundant.

Set the payee: The setReceiptor method allows users with MANAGER_ROLE permission to change the payee address. However, the payee address is not used elsewhere in the contract, which may be redundant code.

Withdraw USDT: The takeUsdt method allows authorized users to transfer a specified amount of USDT from the contract to a specified account, providing a way to manually manage contract funds.

Add liquidity: The addLiquidity method automatically uses the tokens in the contract to exchange USDT and adds liquidity together with the remaining tokens. The process first exchanges a portion of the tokens for USDT, and then adds the exchanged USDT to the liquidity pool along with the remaining tokens.

StaticPool.sol

The contract StaticPool is a staking pool that allows users to deposit USDT to receive rewards. The following is an analysis of the contract's main functions and potential risk points:

Functional Analysis

Initialization: Set the initial state of the contract through the initialize method, including reward tokens, USDT tokens, routers, etc.

Set Xswap purchase ratio: The setXRadio method allows administrators to set the ratio used to purchase Xswap tokens each time a token is invested.

Set the computing power expansion index (powRadio): This function can only be called by addresses that have been granted the MANAGER_ROLE role. The check ensures that the computing power expansion index is set within reasonable expectations and prevents invalid or potentially harmful input.

Distribute rewards: The distribute method calculates and distributes rewards to stakers. Rewards are based on each user's computing power and the ratio of total computing power.

Pledge investment: The deposit method allows users to pledge by depositing USDT and obtain corresponding computing power based on the deposit amount. This process involves allocating USDT to PancakeSwap and Xswap for trading.

Receive rewards: The takeReward method allows users to claim rewards based on their staked share and accumulated yield.

WeekPool.sol

The contract WeekPool implements a periodic reward distribution mechanism, which is mainly used to rank user contributions and distribute rewards on a periodic basis. The following is the main functions and risk analysis of the contract:

The main function

Initialization: The initialize method is used to set initial configurations such as reward tokens, recipients of unowned assets, and card NFT contract addresses.

Get the current cycle index: The currentIndex method returns the current cycle index, calculated based on the contract activation time and the current time.

Get the top address information of a specific period: the topAddressInfoAtEpoch method provides the top address of the specified period and its corresponding amount.

Distributing rewards: The distrubtionAtEpoch method is used to distribute rewards to the top 10 addresses at the end of a specific period. This method ensures that rewards can only be distributed for past periods and only once.

Cumulative rewards: The distribute method is used to accumulate the reward amount of the current period.

Deposit: The deposit method allows users (or agents) to increase contribution value to a specific address, thereby participating in ranking and reward distribution.

4 Audit results

4.1 Key messages

ID	Title	Severity	Status
01	Call minting from any address	Informational	Confirmed
02	Privileged characters can mint coins at will	Low	Confirmed
03	Possible arbitrage attacks	Informational	Confirmed
04	You can add superiors at will to mint coins	Informational	Confirmed
05	Redundant code	Informational	Confirmed
06	afterSell can be called from any address	Informational	Confirmed
07	Excessive deposit amount may lead to arbitrage	Informational	Confirmed
08	Calculation accuracy issues	Informational	Confirmed
09	Privileged roles can set multiple key variables	Low	Confirmed
10	The issue of destroying trading pair tokens	Informational	Confirmed
11	Controllable random numbers	Low	Confirmed

4.2 Audit details

4.2.1 Call minting from any address

ID	Severity	Location	Status
01	Informational	CardNFT.sol: 42, 53	Confirmed

Description

Any address can call the safeMint method to mint coins, and the mint amount can exceed the NFT quantity limit of 12385.

Code location:

```
42      function safeMint(address to) external returns (address) {
43          uint256 tokenId = currentTokenId;
44          _safeMint(to, tokenId);
45          _setTokenURI(
46              tokenId,
47              string(abi.encodePacked(tokenId.toString(), ".json"))
48          );
49          random.update();
50          uint256 add = random.seed() % 100;
51          currentTokenId += add > 0 ? add : 1;
52          return address(uint160(tokenId));
53      }
```

Recommendation

It is recommended to add a role call, and the currency can be minted by any address.

Status

Confirmed.

Business needs are like this.

4.2.2 Privileged characters can mint coins at will

ID	Severity	Location	Status
02	Low	CardNFT.sol: 55,61	Confirmed

Description

The MANAGER_ROLE privileged role address can mint any coins, and the coin limit is 12385.

Code location:

```
55      function minInit(
56          address to,
57          uint256 tokenId
58      ) external onlyRole(MANAGER_ROLE) {
59          require(tokenId < 12385, "error");
60          _safeMint(to, tokenId);
61      }
```

Recommendation

It is recommended that privileged roles be managed using multi-signatures. If the EOA is purely a privileged role, the private key is more likely to be stolen or lost. There are also corresponding privileged roles in several other contracts, and multi-signature management is recommended here.

Status

Confirmed.

Privileged roles only exist temporarily, and all permissions will be destroyed after the project is online and stable.

4.2.3 Possible arbitrage attacks

ID	Severity	Location	Status
03	Informational	EpochController.sol: 177,197	Confirmed

Description

The shareOutBonus method will repurchase the liquidity after the liquidity is released. If the attacker exchanges a large amount of U funds for Token before operating the shareOutBonus method, when the repurchase is completed, the Token price will rise, and the attacker will exchange the U through Token to complete the acquisition. profit.

Code location:

```
177         // Xswap回购x
178         if (xRadio > 0) {
179             uint256 usdtBalance = IERC20Upgradeable(usdt).balanceOf(address(this));
180             IPancakeRouter(router).swapExactTokensForTokens(
181                 (usdtBalance * xRadio) / 1e12,
182                 0,
183                 buyPath,
184                 address(this),
185                 block.timestamp
186             );
187         }
188         // pancke回购x
189         if (IERC20Upgradeable(usdt).balanceOf(address(this)) > 0) {
190             IPancakeRouter(pancakeRouter).swapExactTokensForTokens(
191                 IERC20Upgradeable(usdt).balanceOf(address(this)),
192                 0,
193                 buyPath,
194                 address(this),
195                 block.timestamp
196             );
197         }
```

Recommendation

It is recommended not to use this method for repurchase to avoid being exploited by attackers.

Status

Confirmed.

xSwap's internal purchase adopts a whitelist mechanism. Except for privileged characters, others have no right to purchase, so there is no arbitrage attack.

4.2.4 You can add superiors at will to mint coins

ID	Severity	Location	Status
04	Informational	Family.sol: 78, 107	Confirmed

Description

Any address can call the makeRelation method, use the cardAddr parameter to address (0) to mint coins, and add a superior.

Code location:

```
78      function makeRelation(address parentCardAddr, address cardAddr) external {
79          address ownerCard = cardAddr;
80          if (ownerCard == address(0)) {
81              ownerCard = ICard(card).safeMint(msg.sender);
82          }
83
84          require(
85              ICard(card).ownerOfAddr(ownerCard) == msg.sender,
86              "invalid cardAddr"
87          );
88          _makeRelationFrom(parentCardAddr, ownerCard);
89      }
90
91      function _makeRelationFrom(address parent, address child) internal {
92          require(depthOf[parent] > 0, "invalid parent");
93          require(depthOf[child] == 0, "invalid child");
94
95          // 累加数量
96          totalAddresses++;
97
98          // 上级检索
99          parentOf[child] = parent;
100
101         // 深度记录
102         depthOf[child] = depthOf[parent] + 1;
103
104         // 下级检索
105         _childrenMapping[parent].push(child);
106     }
107 }
```

Recommendation

It is recommended to add restrictions to avoid arbitrary minting.

Status

Confirmed.

Business needs are like this.

4.2.5 Redundant code

ID	Severity	Location	Status
05	Informational	Robot.sol: 33, 36	Confirmed

Description

The receiptor is not used in the entire project. There is redundant code and whether there is any unimplemented logic.

Code location:

```
33      function setReceiptor(address account) external onlyRole(MANAGER_ROLE) {
34          require(account != address(0), "not zero");
35          receiptor = account;
36      }
```

Recommendation

If it is redundant code, it is recommended to delete it.

Status

Confirmed.

Redundant code removed.

4.2.6 afterSell can be called from any address

ID	Severity	Location	Status
06	Informational	StaticPool.sol: 272, 305	Confirmed

Description

The afterSell method can be called from any address to sell reinvestment, but this method does not determine the caller's address. When any address can be called, it will result in a loss of contract funds. The operator parameter passed in here serves as the operator, but it does not have any logic. Is there any unimplemented logic?

Code location:

```
272     function afterSell(
273         address operator,
274         address to,
275         uint256 tokenId
276     ) external override nonReentrant {
277         operator;
278         address[] memory tmpPath = new address[](2);
279         tmpPath[0] = usdt;
280         tmpPath[1] = rewardToken;
281         require(ICard(card).ownerOf(tokenId) != address(0), "invalid cardAddr");
282         uint256 usdtAmount = IERC20Upgradeable(usdt).balanceOf(address(this));
283         require(usdtAmount > 0, "FarmSellProvider: NO_SELL");
284         // 卖出所得usdt45%给用户
285         IERC20Upgradeable(usdt).safeTransfer(to, (usdtAmount * 0.45e12) / 1e12);
286 
287         // 卖出所得usdt55%回购token销毁
288         IPancakeRouter(router).swapExactTokensForTokens(
289             IERC20Upgradeable(usdt).balanceOf(address(this)),
290             0,
291             tmpPath,
292             address(this),
293             block.timestamp
294         );
295         if (tokenId > 0) {
296             address cardAddr = address(uint160(tokenId));
297             IERC20Burn(rewardToken).burn(IERC20Upgradeable(rewardToken).balanceOf(address(this)));
298             uint256 power = getPowerByUSDT((usdtAmount * 0.55e12) / 1e12);
299 
300             _deposit(cardAddr, power);
301 
302             emit Deposit(cardAddr, usdtAmount, power, block.timestamp);
303         }
304     }
305 }
```

Recommendation

Determine the calling address.

Status

Confirmed.

The afterSell code logic ensures that calling this method must ensure that the current contract holds usdt, so there is no risk.

4.2.7 Excessive deposit amount may lead to arbitrage

ID	Severity	Location	Status
07	Informational	StaticPool.sol: 173, 205	Confirmed

Description

The deposit method is used for pledge. When pledging, U will be converted into Token to add liquidity. If the attacker exchanges a large amount of U funds for Token before the deposit is made, when the deposit here is completed, the Token price will rise, and the attacker will use Token to exchange it for Token. U is redeemed to make a profit.

Code location:

```
173     function deposit(address cardAddr, uint256 amount) external nonReentrant {
174         require(ICard(card).ownerOfAddr(cardAddr) != address(0),"invalid cardAddr");
175         require(amount >= 100e18, "StaticPool: Too Low");
176 
177         IERC20Upgradeable(usdt).safeTransferFrom(msg.sender,address(this),amount);
178 
179         //pancake
180         uint256 usdtPanckAmount = (amount * (1e12 - xRadio)) / 1e12;
181         if (usdtPanckAmount > 0) {
182             _panckeAddLiquidity(usdtPanckAmount / 2);
183         }
184 
185         //Xswap
186         uint256 usdtXAmount = IERC20Upgradeable(usdt).balanceOf(address(this));
187         if (usdtXAmount > 0) {
188             IPancakeRouter(router).swapExactTokensForTokens(
189                 usdtXAmount / 2,
190                 0,
191                 buyPath,
192                 address(this),
193                 block.timestamp
194             );
195             IPancakeRouter(router).addLiquidity(
196                 address(rewardToken),
197                 usdt,
198                 IERC20Upgradeable(rewardToken).balanceOf(address(this)),
199                 IERC20Upgradeable(usdt).balanceOf(address(this)),
200                 0,
201                 0,
202                 address(0),
203                 block.timestamp
204             );
205         }
```

Recommendation

Avoid arbitrage caused by excessive quantity.

Status

Confirmed.

xtoken tokens have a selling fee of up to 20%, so there is no arbitrage space for actual traders.

4.2.8 Calculation accuracy issues

ID	Severity	Location	Status
08	Informational	XToken.sol: 113, 126	Confirmed

Description

The calculation of burnFee is based on 3 hours. The calculation result here contains zero decimals. In subsequent startTime updates, the startTime value will be smaller than the current time.

Code location:

```
113        uint256 balancePair = IERC20(pair).totalSupply();
114        uint256 burnFee = startTime > 0 ? (block.timestamp - startTime) / 3 hours : 0;
115        //
116        if (burnFee > 0 && balancePair > 0) {
117            uint256 burnAmount;
118            uint256 balance = balanceOf(pair);
119            for (uint i = 0; i < burnFee; i++) {
120                burnAmount += balance / 100;
121                balance -= balance / 100;
122            }
123            _burn(pair, burnAmount);
124            Ipair(pair).sync();
125            startTime += 3 hours * burnFee;
126        }
```

Recommendation

It is recommended to assign the current time to startTime.

Status

Confirmed.

Business needs are like this.

4.2.9 Privileged roles can set multiple key variables

ID	Severity	Location	Status
09	Low	XToken.sol: 148, 170	Confirmed

Description

Privileged roles can set blacklists and other attribute addresses. If any address is set to a whitelist, transactions will be impossible.

Code location:

```
148     function addGuarded(address account) external onlyOwner {
149         require(!isGuardedOf[account], "account already exist");
150         isGuardedOf[account] = true;
151         emit Guarded(account, block.timestamp, true);
152     }
153
154     function removeGuarded(address account) external onlyOwner {
155         require(isGuardedOf[account], "account not exist");
156         isGuardedOf[account] = false;
157         emit Guarded(account, block.timestamp, false);
158     }
159
160     function addBlocked(address account) external onlyOwner {
161         require(!isBlockedOf[account], "account already exist");
162         isBlockedOf[account] = true;
163         emit Blocked(account, block.timestamp, true);
164     }
165
166     function removeBlocked(address account) external onlyOwner {
167         require(isBlockedOf[account], "account not exist");
168         isBlockedOf[account] = false;
169         emit Blocked(account, block.timestamp, false);
170     }
```

Recommendation

It is recommended to cancel the black and white list attribute, and it is recommended to use multi-signature management for privileged roles.

Status

Confirmed.

Privileged roles only exist temporarily, and all permissions will be destroyed after the project is online and stable.

4.2.10 The issue of destroying trading pair tokens

ID	Severity	Location	Status
10	Informational	XToken.sol: 100, 140	Confirmed

Description

When transferring, if the to address is a trading pair, the token will be destroyed, but here, funds can be exchanged through flash loans, and then the _transfer method is called to reduce the pair pool tokens, which affects the transaction comparison ratio, and then the flash loan funds can be exchanged back to make a profit.

Code location:

```
100     function _transfer(
101         address from,
102         address to,
103         uint256 amount
104     ) internal override {
105         require(!isBlockedOf[from] && !isBlockedOf[to], "blocked!");
106 
107         if (!isGuardedOf[from] && !isGuardedOf[to]) {
108             if (buyFee > 0 && isPairsOf(from)) {
109                 uint256 buyFeeAmount = (amount * buyFee) / 1e12;
110                 super._transfer(from, buyPreAddress, buyFeeAmount);
111                 amount -= buyFeeAmount;
112             } else if (sellFee > 0 && isPairsOf(to)) {
113                 uint256 balancePair = IERC20(pair).totalSupply();
114                 uint256 burnFee = startTime > 0 ? (block.timestamp - startTime) / 3 hours : 0;
115                 //
116                 if (burnFee > 0 && balancePair > 0) {
117                     uint256 burnAmount;
118                     uint256 balance = balanceOf(pair);
119                     for (uint i = 0; i < burnFee; i++) {
120                         burnAmount += balance / 100;
121                         balance -= balance / 100;
122                     }
123                     _burn(pair, burnAmount);
124                     Ipair(pair).sync();
125                     startTime += 3 hours * burnFee;
126                 }
127 
128                 uint256 sellFeeAmount = (amount * sellFee) / 1e12;
129                 super._transfer(from, robot, sellFeeAmount);
130                 amount -= sellFeeAmount;
131 
132                 if (balanceOf(robot) >= 1e18 && balancePair > 0) {
133                     IRobot(robot).addLiquidity();
134                 }
135 
136                 if ((balanceOf(from) * 9999) / 10000 <= amount) {
137                     amount = (balanceOf(from) * 9999) / 10000;
138                 }
139             }
140         }
```

Recommendation

It is recommended not to destroy the trading pair funds when transferring.

Status

Confirmed.

Business needs are like this.

4.2.11 Controllable random numbers

ID	Severity	Location	Status
11	Low	Random.sol: 78, 107	Confirmed

Description

The generation of random numbers depends on block.timestamp. Since _seed and _internalRandomSeed are updated and initialized inside the contract, the generated random numbers are predictable. The value of _seed can be increased through the update() function. Without limiting the caller, this function can be called from any address, resulting in a controllable _seed value.

Code location:

```
19      function seed() external view returns (uint256) {
20          return
21              uint256(
22                  keccak256(
23                      abi.encodePacked(
24                          block.timestamp,
25                          _seed,
26                          _internalRandomSeed >> (_seed % 32)
27                      )
28                  )
29              );
30      }
31
32      function update() external {
33          _seed++;
34      }
35  }
```

Recommendation

Use a random number generation service (such as Chainlink VRF) as the source of random numbers, and add update() function access control to prevent arbitrary address calls from causing _seed value manipulation.

Status

Confirmed.

This random number is indeed controllable. However, the only application of random numbers in this project lies in the span of each increment of nft's tokenId. This has no impact on the business process and should be a very, very minor problem.

5 Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a struct assignment operation affecting an in-memory struct rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as constant contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Disclaimer

This report is issued in response to facts that occurred or existed prior to the issuance of this report, and liability is assumed only on that basis.
Shield Security cannot determine the security status of this program and assumes no responsibility for facts occurring or existing after the date of this report. The security audit analysis and other content in this report is based on documents and materials provided to Shield Security by the information provider through the date of the insurance report. in Shield Security's opinion. The information provided is not missing, falsified, deleted or concealed. If the information provided is missing, altered, deleted, concealed or not in accordance with the actual circumstances, Shield Security shall not be liable for any loss or adverse effect resulting therefrom. shield Security will only carry out the agreed security audit of the security status of the project and issue this report. shield Security is not responsible for the background and other circumstances of the project. Shield Security is not responsible for the background and other circumstances of the project.