Smart Contract Security Audit Report

For

Pointswap

Date Issued: Jun 21, 2023

Version: v1.0

Confidentiality Level: Public

Contents

1 Abstract

This report was prepared for Pointswap smart contract to identify issues and vulnerabilities in its smart contract source code. A thorough examination of Pointswap smart contracts was conducted through timely communication with Pointswap, 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 Name	Shield Security
Start date	Jun 15, 2023
End date	Jun 21, 2023
Contract type	DeFi
Language	Solidity
File	Achievement.sol, Family.sol, SwapERC20.sol, SwapFactory.sol, SwapPair.sol, SwapRouter.sol, PETToken.sol, WrappedCoin.sol, DividendPool.sol, Mapper.sol, StaticPool.sol

2.2 Audit Scope

File	SHA256
Achievement.sol	B792EDF386B2D3DAC9EE8589B2EE7A2F6D4DCC3E20DE3B6AEE6F079657CA7EED
Family.sol	8B888BF23EAD8C5F4EDF8DD4506324952D81B899A262F5F9E47E5767B7E7663B
SwapERC20.sol	4145192C308FECDC8C0175DB0ABABB3E951BFD98F78E018C1C91D038E6A27766
SwapFactory.sol	1DF6066FB5031C4BA9F33220317BBEC6529A37713E63E85BA9169D002A3CCC04
SwapPair.sol	8494766B92D3EC690C5456DAD5F228DF5E02B3ABCB96832634279D9B394FB0A8
SwapRouter.sol	3A6644CC62DCE3B9ADE8FC123E332E4A7579C1E26D1A4B31C6ABA5F6D2F1B91D
PETToken.sol	8E6388AE244EEDFCEDEB7D205245D57712B2DF11A8CA3F56599875C483549720
WrappedCoin.sol	1C3A6FF8AC10A8477BF7A6662DA58EB393C0F396783266DDAFF76BBEADDE3980
DividendPool.sol	8BF936D38071513660F9AFE9A487E31447D39F3E072C990CC9AFACE02F648160
Mapper.sol	B0FA3471D4991A9AFCD9B2B0C2F92AA7B2032F795E4C9D7040036B2AD9ABE556
StaticPool.sol	BA45F243F4B7FE85CEC7F67BF35A161715CFF54B2BFD3AEC20D85BF9AE2FBA9A

3 Project contract details

3.1 Contract Overview

StaticPool Contract

The contract provides a static staking pool. Users can pledge tokens to the contract, and obtain corresponding computing power according to the number of pledged tokens, thereby obtaining rewards. Users can withdraw the pledged principal and unclaimed rewards at any time. And it also realizes the calculation of rewards for users at the upper level, and the function of extracting dynamic dividend rewards for upper users.

DividendPool Contract

This contract implements a dividend contract for managing and distributing reward tokens to users. Users are allowed to deposit tokens and receive corresponding rewards based on their computing power.

Mapper Contract

This contract is a mapping contract, which is used to map old tokens to new tokens, and provides some auxiliary functions.

Migrations Contract

The contract mainly provides the function of setting the status of the migration completion, but also restricts that only the owner of the contract can call the function of setting the status.

Achievement Contract

The contract maintains data such as the amount of user funds deposited, the total amount deposited by the community, and the total amount of users at all levels. Implemented a function for calculating user performance and user rating. User upgrades meet various conditions set by the project party. And the administrator has the authority to clean up the level information of any user. Users can obtain different amounts of reward data by level. Admins can control reward parameters.

Family Contract

The contract maintains a tree-like structure, which is used to save the relationship between the user's subordinates and the depth data of the tree, and also supports the function of directly pushing the superior through the subordinate.

PETToken Contract

This contract implements the ERC20 standard Token token contract in the OpenZeppelin library. It has basic token functions, including transfer, balance inquiry and authorization, etc. The contract adds the collection of transaction fees. And implemented the blacklist restriction function.

SwapPair Contract

This contract implements a SwapPair contract that handles liquidity. The function of providing liquidity management is realized. The contract maintains the reserves (reserve0 and reserve1) of two tokens (token0 and token1), and calculates liquidity based on the reserves of the trading pair. Users can provide liquidity by calling the mint function and deposit tokens into the contract. The contract provides a swap function for performing token transactions. This function transfers tokens from one account to another and updates the reserve after certain conditions are met. Users do not call the contract directly, but indirectly through the SwapRouter contract.

SwapRouter Contract

The contract implements a trading mechanism based on SwapRouter, through which users can add and remove liquidity, as well as exchange tokens, while supporting special handling of farm coins.

3.2 Code Overview

Achievement Contract

Function Name	Visibility	Modifiers
initialize	Public	initializer
setLevelRewardPropsf	External	onlyRole(DEFAULT_ADMIN_ROLE)
clearUserLevel	External	onlyRole(DELEGATE_ROLE)
increaseDelegate	External	onlyRole(DELEGATE_ROLE)
levelOf	Public	-
distrubutionRewards	External	-
distrubutionsForefathers	Public	-
getUserLevelInfo	External	-
upUserLevel	External	-
_increase	Internal	-

Family Contract

Function Name	Visibility	Modifiers
initialize	Public	initializer
getForefathers	External	-
childrenOf	External	-
makeRelation	External	-
_makeRelationFrom	Internal	-

PETToken Contract

Function Name	Visibility	Modifiers
addPair	External	onlyOwner
removePair	External	onlyOwner
setSellFee	External	onlyOwner
setBuyFee	External	onlyOwner
setBuyPreAddress	External	onlyOwner
setSellPreAddress	External	onlyOwner
_checkOnSwapPair	Internal	-
_transfer	Internal	-
isGuardUser	External	-
isBlockUser	External	-
guardedView	External	-
blockedView	External	-
addGuarded	External	onlyOwner
removeGuarded	External	onlyOwner
addBlocked	External	onlyOwner
removeBlocked	External	onlyOwner
clim	External	-

DividendPool Contract

Function Name	Visibility	Modifiers
initialize	External	initializer
start	External	onlyRole(MANAGER_ROLE)
updatePool	Public	onlyRole(MANAGER_ROLE)
transport	External	-
userPower	External	-
earned	Public	-
deposit	External	onlyRole(DELEGATE_ROLE)
withdraw	External	onlyRole(DELEGATE_ROLE)
takeReward	External	-

Mapper Contract

Function Name	Visibility	Modifiers
setMapToken	External	onlyOwner
blockedView	External	-
oldTokenList	External	-
addBlocked	External	onlyOwner
removeBlocked	External	onlyOwner
mapToken	External	lock

StaticPool Contract

Function Name	Visibility	Modifiers
initialize	External	initializer
setMultiple	External	onlyRole(MANAGER_ROLE
setDayOutPut	External	onlyRole(MANAGER_ROLE
startEpoch	External	onlyRole(MANAGER_ROLE
getPowerByToken	Public	-
earned	Public	-
deposit	External	lock
withdraw	External	lock
takeReward	External	lock
distributeReward	Internal	-
takeDynamicReward	External	-

4 Audit results

4.1 Key messages

ID	Title	Severity	Status
01	A accTokenPerShare variable that is not updated results in rewards of zero	Informational	confirm
02	tokenInfo.isPg being true can cause users to continuously receive new Tokens	Low	confirm
03	Modifiable bonus factor	Low	confirm
04	Insecure funds transfer sequence when depositing	Informational	confirm
05	There is a blacklist address setting	Low	confirm
06	redundant code	Informational	confirm

4.2 Audit details

4.2.1 A accTokenPerShare variable that is not updated results in rewards of zero

ID	Severity	Location	Status
01	Informational	DividendPool.sol: 81, 103	confirm

Description

The deposit() method in the contract is used for deposit and will record the user's deposit information at the same time. The earned() method is used to calculate the user's available income. However, when calculating the income, it is calculated through accTokenPerShare. accTokenPerShare is mainly invoked by privileged roles. When the privileged role does not update the accTokenPerShare variable for a long time, the calculated benefit of this variable is always zero. It may result in users not getting benefits.

Code location:

```
81      function earned(address account) public view returns (uint256) {
82          UserInfo memory user = userPoolInfo[account];
83          if (totalPower == 0 || block.number <= startBlock || startBlock == 0) {
84              return 0;
85          }
86          return
87              user.reward +
88              (user.power * (accTokenPerShare - user.rewardDebt)) /
89              1e12;
90      }
91
92      function deposit(address account,uint256 amount ) external onlyRole(DELEGATE_ROLE) {
93          UserInfo storage user = userPoolInfo[account];
94
95          if (user.power != 0) {
96              user.reward = earned(account);
97          }
98          totalPower -= user.power;
99          user.power = amount;
100         totalPower += amount;
101         user.rewardDebt = accTokenPerShare;
102
103     }
```

Recommendation

It is recommended that the user's revenue calculation be calculated in real time based on time or blocks, so as to avoid the problem of updating privileged roles resulting in little or no revenue.

Status

The official confirms that the above content is for business needs.

4.2.2 tokenInfo.isPg being true can cause users to continuously receive new Tokens

ID	Severity	Location	Status
02	Low	Mapper.sol: 122, 163	confirm

Description

The mapToken() method is used to map tokens. When mapping tokens, the tokenInfo.isPg variable will be taken. When the value is true, the content in the first if statement will be executed, but the code in this statement is not the caller's The old token funds are sent to okenInfo.oldReceiver, but the caller can still get the new Token. At this time, if the user sends the old token to other addresses, other addresses can still use the old token to obtain new tokens.

Code location:

```
122     function mapToken(address token) external lock {
123         TokenInfo storage tokenInfo = tokenInfos[token];
124         require(tokenInfo.isOpened, "Mapper: closed!");
125         require(tokenInfo.newAddress != address(0), "Mapper: invalid token!");
126         uint256 amount = IERC20(token).balanceOf(msg.sender);
127         require(amount > 0, "Mapper: invalid zero!");
128         uint256 sendAmount;
129         if (tokenInfo.isPg) {
130             require(!blocked.contains(msg.sender), "Mapper: blocked!");
131             sendAmount = amount - userMaped[token][msg.sender];
132             userMaped[token][msg.sender] += sendAmount;
133             tokenInfo.totalSupply += sendAmount;
134         } else {
135             uint256 beforeBalance = IERC20(token).balanceOf(
136                 tokenInfo.oldReceiver
137             );
138             IERC20(token).safeTransferFrom(
139                 msg.sender,
140                 tokenInfo.oldReceiver,
141                 amount
142             );
143             uint256 afterBalance = IERC20(token).balanceOf(
144                 tokenInfo.oldReceiver
145             );
146             sendAmount = afterBalance - beforeBalance;
147             userMaped[token][msg.sender] += amount;
148             tokenInfo.totalSupply += amount;
149         }
150
151         tokenInfo.totalLimitl++;
152
153         IERC20(tokenInfo.newAddress).safeTransfer(msg.sender, sendAmount);
154
155         emit MapToken(
156             token,
157             tokenInfo.newAddress,
158             msg.sender,
159             amount,
160             sendAmount,
161             block.timestamp
162         );
163     }
```

Recommendation

It is recommended to record and transfer the old token that has been mapped in time when calling this method for token mapping, so as to prevent users from using the old token to obtain new tokens multiple times.

Status

The official confirms that the above content is for business needs.

4.2.3 Modifiable bonus factor

ID	Severity	Location	Status
03	Low	StaticPool.sol: 112, 120	confirm

Description

The multiple and dayOutPut variables both play a key role in the reward calculation. These two variables are modified by privileged roles. If the privileged role is maliciously controlled, the value of this variable may increase sharply, resulting in a sharp increase in rewards.

Code location:

```
112     function setMultiple(uint256 _multiple) external onlyRole(MANAGER_ROLE) {
113         multiple = _multiple;
114     }
115
116     function setDayOutPut(
117         uint256 _dayOutPut
118     ) external onlyRole(MANAGER_ROLE) updatePool {
119         dayOutPut = _dayOutPut;
120     }
```

Recommendation

It is recommended that privileged roles use multi-signature management.

Status

The official confirms that the above content is for business needs.

4.2.4 Insecure funds transfer sequence when depositing

ID	Severity	Location	Status
04	Informational	StaticPool.sol: 169, 197	confirm

Description

The deposit() method is used for pledge input. This method first records the amount that the user wants to pledge, and then transfers the user's funds. In order to avoid the security risk of reentry, it is necessary to use the mode of first transferring and then updating variables when depositing.

Code location:

```
169     function deposit(address account, uint256 amount) external lock updatePool {
170         require(amount > 0, "StaticPool: no zero");
171         UserInfo storage user = userInfoOf[account];
172         if (user.bookAddress == address(0)) {
173             user.bookAddress = address(new Vault(token));
174         }
175
176         uint256 power = getPowerByToken(amount);
177
178         if (user.power > 0) {
179             user.reward +=
180                 (user.power * (accTokenPerShare - user.rewardDebt)) /
181                 1e12;
182         }
183         user.supply += amount;
184         user.power += power;
185         user.rewardDebt = accTokenPerShare;
186         user.unLockAt = block.timestamp + 30 days;
187         totalPower += power;
188
189         achievement.increaseDelegate(account, amount);
190
191         IERC20Upgradeable(token).safeTransferFrom(
192             msg.sender,
193             user.bookAddress,
194             amount
195         );
196         emit Deposit(account, amount, power, block.timestamp);
197     }
```

Recommendation

It is recommended that the user transfer the funds first, and then update the user's deposit variable.

Status

The official confirms that the above content is for business needs.

4.2.5 There is a blacklist address setting

ID	Severity	Location	Status
05	Low	PETToken.sol: 75, 92	confirm

Description

The PETToken contract issued PET tokens, but there is a blacklist address in the contract. If the user address is included in the blacklist, the user will not be able to continue the transaction.

Code location:

```
75      function _transfer(
76          address from,
77          address to,
78          uint256 amount
79      ) internal override {
80          require(!blocked.contains(from) && !blocked.contains(to), "blocked!");
81
82          if (!guarded.contains(from) && !guarded.contains(to)) {
83              if (buyFee > 0 && _checkOnSwapPair(from)) {
84                  uint256 buyFeeAmount = (amount * buyFee) / 1e12;
85                  super._transfer(from, buyPreAddress, buyFeeAmount);
86                  amount -= buyFeeAmount;
87              } else if (sellFee > 0 && _checkOnSwapPair(to)) {
88                  uint256 sellFeeAmount = (amount * sellFee) / 1e12;
89                  super._transfer(from, sellPreAddress, sellFeeAmount);
90                  amount -= sellFeeAmount;
91              } else {}
92          }
```

Recommendation

It is recommended to remove the blacklist restriction.

Status

The official confirms that the above content is for business needs.

4.2.6 redundant code

ID	Severity	Location	Status
05	Informational	PETToken.sol: 75, 92 Migrations.sol: 8,19	confirm

Description

PETToken.sol

The _transfer() method is used for user transfer. There is an if-else statement in this method, and the last else statement does not have code logic. It is recommended to delete it.

Code location:

```
75      function _transfer(
76          address from,
77          address to,
78          uint256 amount
79      ) internal override {
80          require(!blocked.contains(from) && !blocked.contains(to), "blocked!");
81
82          if (!guarded.contains(from) && !guarded.contains(to)) {
83              if (buyFee > 0 && _checkOnSwapPair(from)) {
84                  uint256 buyFeeAmount = (amount * buyFee) / 1e12;
85                  super._transfer(from, buyPreAddress, buyFeeAmount);
86                  amount -= buyFeeAmount;
87              } else if (sellFee > 0 && _checkOnSwapPair(to)) {
88                  uint256 sellFeeAmount = (amount * sellFee) / 1e12;
89                  super._transfer(from, sellPreAddress, sellFeeAmount);
90                  amount -= sellFeeAmount;
91              } else {}
92          }
```

Migrations.sol

The setCompleted() method is used to update the last_completed_migration variabl e, which is not used in this project.

Code location:

```
8     modifier restricted() {
9       require(
10        msg.sender == owner,
11        "This function is restricted to the contract's owner"
12      );
13      _;
14    }
15
16    function setCompleted(uint completed) public restricted {
17      last_completed_migration = completed;
18    }
19  }
```

Recommendation

It is recommended to remove unused code.

Status

The official confirms that the above content is for business needs.

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.