BEOSIN
Blockchain Security

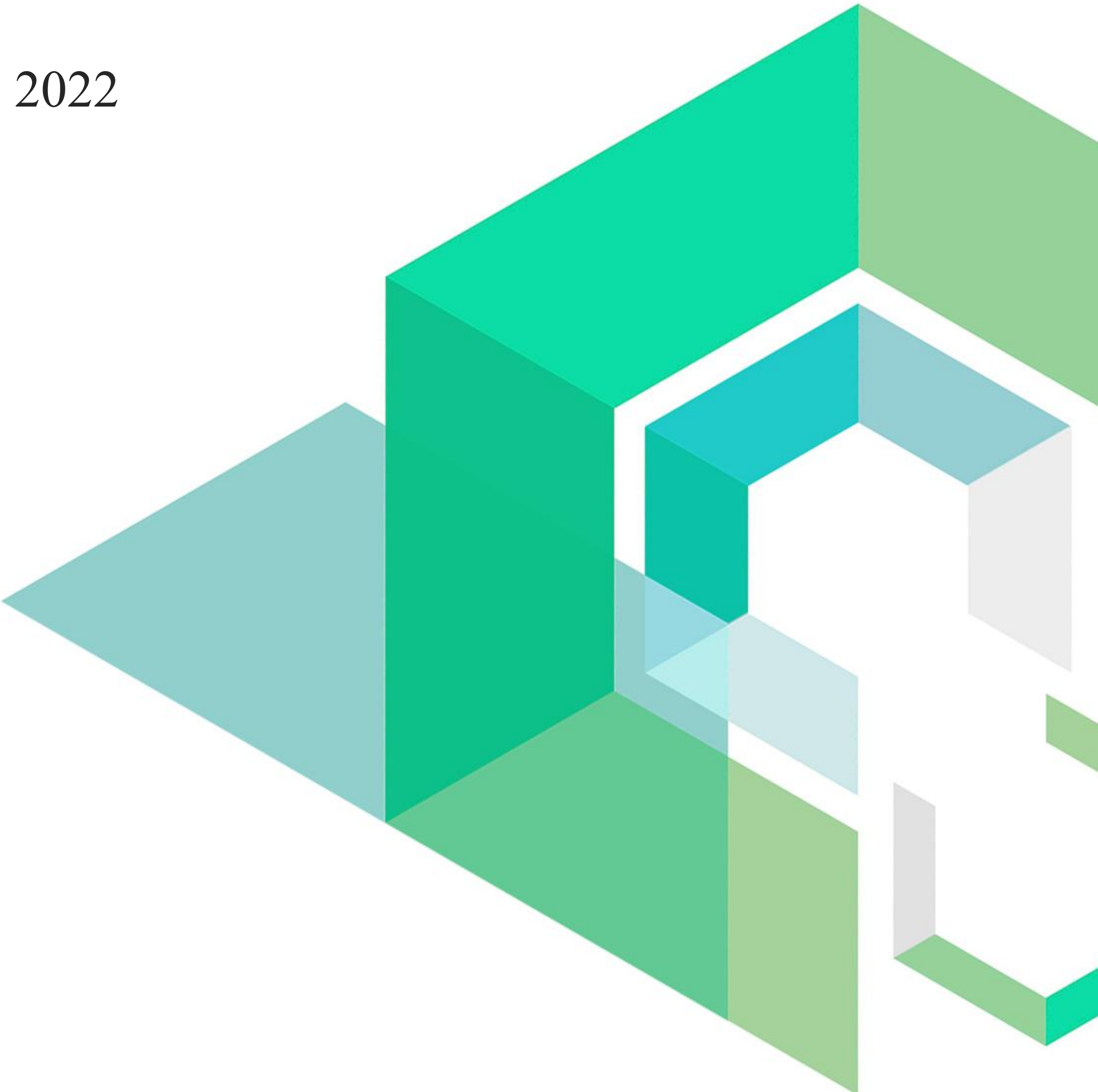
PMT

Smart Contract Security Audit

V1.0

No. 202211241633

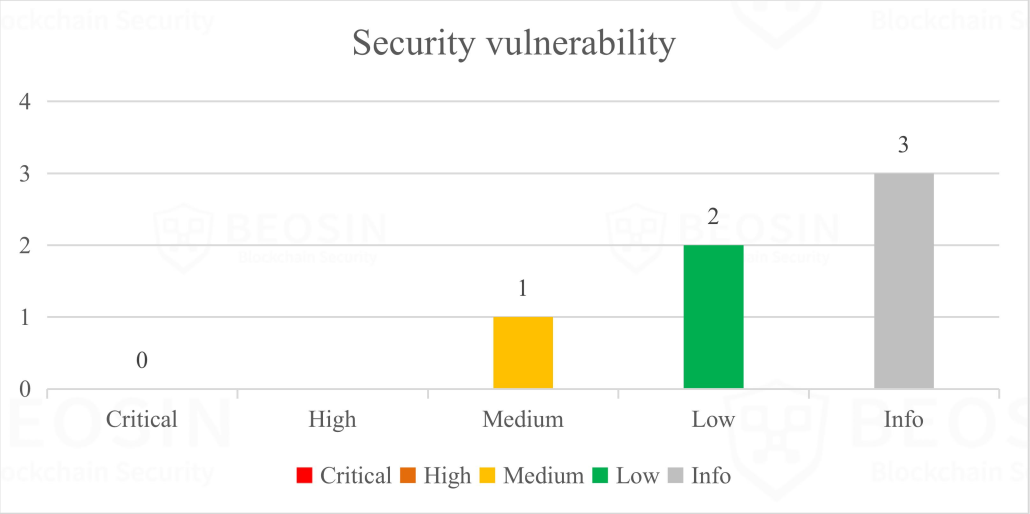
Nov 24th, 2022



Contents

Summary of Audit Results

After auditing, 1 Medium-risk, 2 Low-risk and 3 Info-risk items were identified in the PMT project. Specific audit details will be presented in the **Findings** section. Users should pay attention to the following aspects when interacting with this project:



***Notes:**

- **Risk Description:**

1. There is asset centralization risk that this project mint all token to owner address.

- **Project Description:**

1. Basic Token Information

Token name	Pyramid Management Trading
Token symbol	PMT
Decimals	18
Pre-mint	0 (All to deployer)
Total supply	209,992,355 (Burnable)
Token type	BEP-20

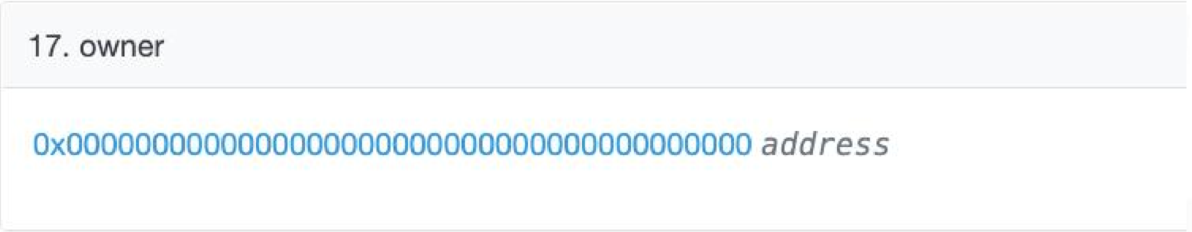
Table 1 Basic information of PMT

2. Business overview

The PMT project is BEP-20 token issued on BNB chain. PMT can be minted and can be burned (reduce totalSupply_ but not transfer to zero address), the max supply of PMT is 210 million. Based on BNB chain data, PMT token was minted with max supply amount to owner address after contract creation and mint function can't be used anymore. The deployer will be granted owner permission when the contract is deployed. The owner has the right to mint token and can set value of setIsExcludedFromFee, setMarketPairStatus.

Transfer operation have different fees conditions. There will be no fee in transfer if sender and recipient is not MarketPair address. If the sender or recipient address is in isExcludedFromFee list, it will perform the transfer operation without fee. If sender or recipient is MarketPair address, 5% fee or 2% fee will be triggered (when the sender is MarketPair address, the _buyTax fee is 5%; while the recipient is MarketPair address, the fee is 2%, which include 5% _sellTax fee minus 3% _sellBurn) which will send to contract address. After that sending 1% rake back to walletMarket and walletFund address respectively from contract address balance, 3% liquidity provider fee to walletLp address from contract address balance, and update balance in this circumstance.

This project was already deployed on BNB chain. Based on BNB chain data, the owner address had renounced ownership, and *setMarketPairStatus* function can't be used anymore which means isMarketPair status can't be changed. As well as *setIsExcludedFromFee* function that can't add new address in isExcludedFromFee list for fee waive.



17. owner

0x0000000000000000000000000000000000000000 *address*

The owner address of PMT contract

1 Overview

1.1 Project Overview

Project Name	PMT
Platform	BNB Chain
Contract Address	Initial 0xb34baae2cc8aac788dd468c13b4f34669979207e Finally 0x09566fd86533832017dc7c45d570dc51403547d4

1.2 Audit Overview

Audit work duration: November 15, 2022 – November 24, 2022

Audit methods: Formal Verification, Static Analysis, Typical Case Testing and Manual Review.

Audit team: Beosin Security Team

2 Findings

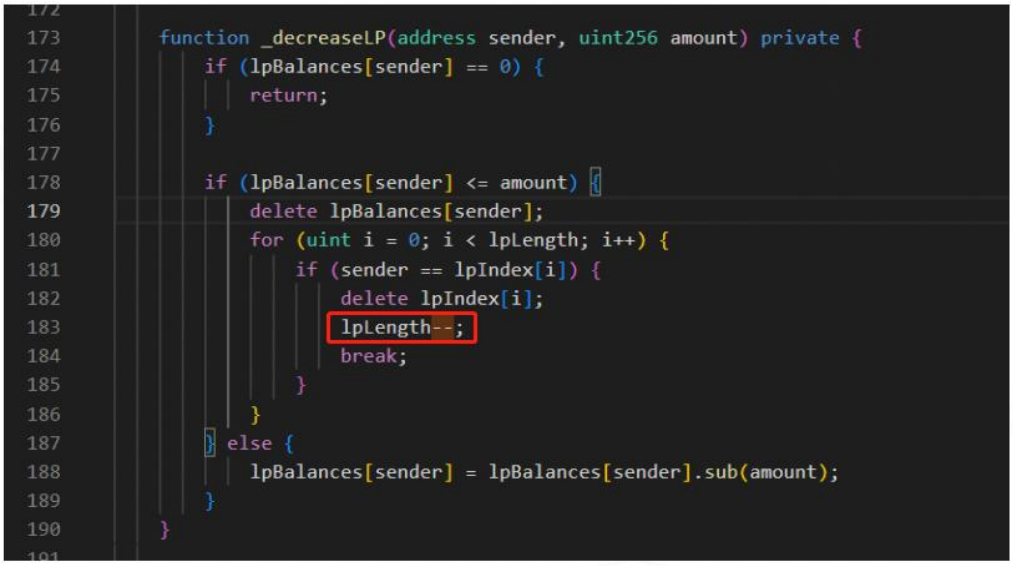
Index	Risk description	Severity level	Status
PMT-1	*_decreaseLP* function design logic error	Medium	Fixed
PMT-2	Centralization risk	Low	Acknowledged
PMT-3	Flash loan risk	Low	Fixed
PMT-4	Missing trigger event	Info	Acknowledged
PMT-5	Redundant code	Info	Partially Fixed
PMT-6	Insufficient gas causes the operation to fail	Info	Fixed

Status Notes:

- PMT-2 is unfixed and has centralization risk.
- PMT-4 is unfixed and will not cause any issue.
- PMT-5 is partially fixed and will not cause any issue.

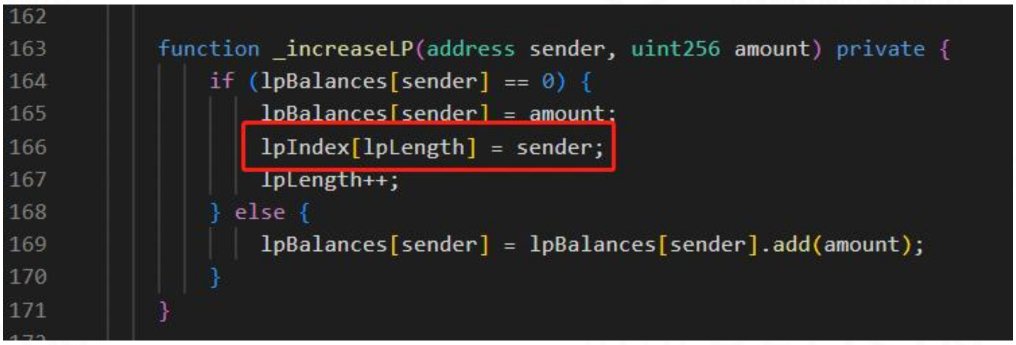
[PMT-1] *_decreaseLP* function design logic error

Severity Level	**Medium**
Type	Business Security
Lines	PMT.sol #L163-170, L173-190 , L213-234
Description	When delete the specified address in the for loop of *_decreaseLP* function, it performs the lpLength-- operation. This will cause address to be overwritten for accessing *_increaseLP* function and assigning new address to lpIndex[lpLength]. Meanwhile, it also cause the profit calculated incorrectly in *_buyRakeBack* function, and the last address of lpIndex may not get profit.



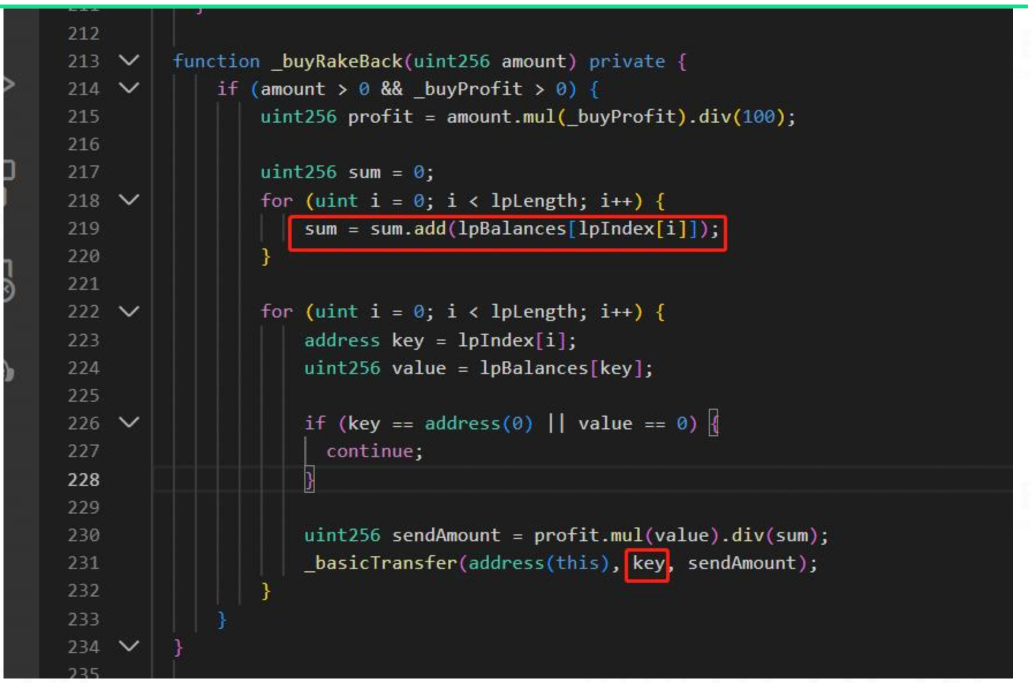
```
173     function _decreaseLP(address sender, uint256 amount) private {
174         if (lpBalances[sender] == 0) {
175             return;
176         }
177 
178         if (lpBalances[sender] <= amount) {
179             delete lpBalances[sender];
180             for (uint i = 0; i < lpLength; i++) {
181                 if (sender == lpIndex[i]) {
182                     delete lpIndex[i];
183                     lpLength--;
184                     break;
185                 }
186             }
187         } else {
188             lpBalances[sender] = lpBalances[sender].sub(amount);
189         }
190     }
```

Figure 1 Source code of *_decreaseLP* function



```
162 
163     function _increaseLP(address sender, uint256 amount) private {
164         if (lpBalances[sender] == 0) {
165             lpBalances[sender] = amount;
166             lpIndex[lpLength] = sender;
167             lpLength++;
168         } else {
169             lpBalances[sender] = lpBalances[sender].add(amount);
170         }
171     }
```

Figure 2 Source code of *_increaseLP* function

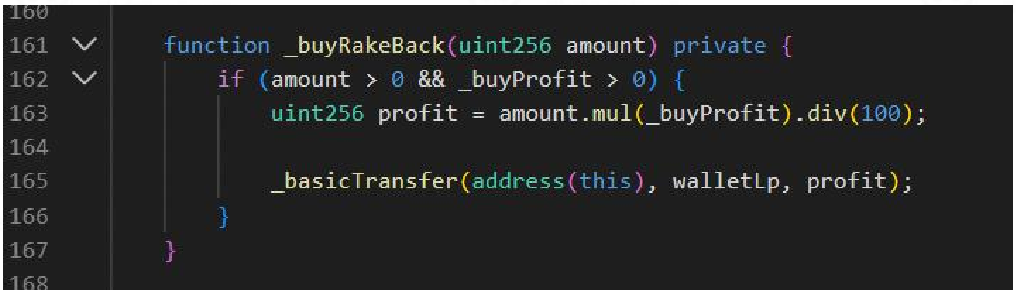


```
function _buyRakeBack(uint256 amount) private {
    if (amount > 0 && _buyProfit > 0) {
        uint256 profit = amount.mul(_buyProfit).div(100);

        uint256 sum = 0;
        for (uint i = 0; i < lpLength; i++) {
            sum = sum.add(lpBalances[lpIndex[i]]);
        }

        for (uint i = 0; i < lpLength; i++) {
            address key = lpIndex[i];
            uint256 value = lpBalances[key];

            if (key == address(0) || value == 0) {
                continue;
            }

            uint256 sendAmount = profit.mul(value).div(sum);
            _basicTransfer(address(this), key, sendAmount);
        }
    }
}
```

Figure 3 Source code of *_buyRakeBack* function

Recommendations It is recommended to delete lpLength--.

Status Fixed. The project team deleted related the code of design logic.



```
function _buyRakeBack(uint256 amount) private {
    if (amount > 0 && _buyProfit > 0) {
        uint256 profit = amount.mul(_buyProfit).div(100);

        _basicTransfer(address(this), walletLp, profit);
    }
}
```

Figure 4 Source code of *_buyRakeBack* function

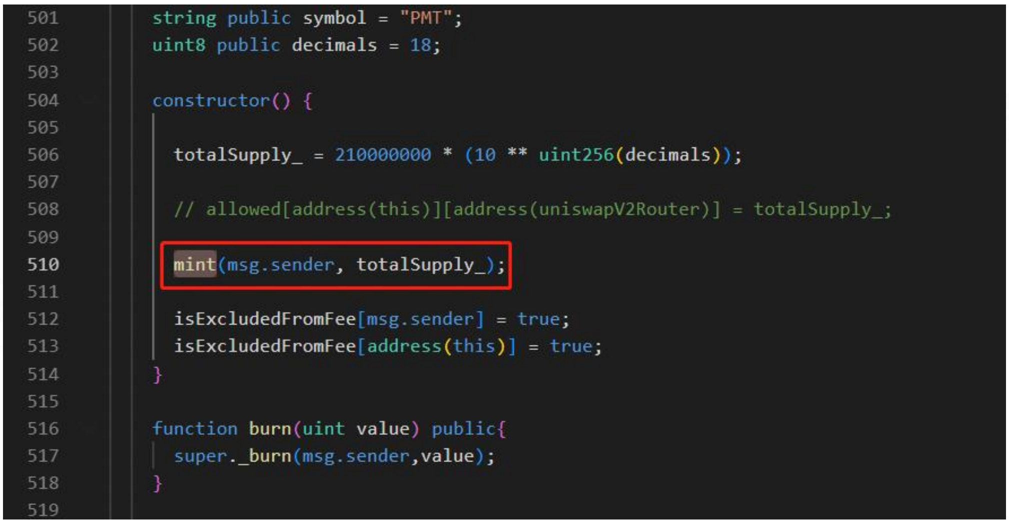
[PMT-2] Centralization risk

Severity Level Low

Type Business Security

Lines PMT.sol #L510

Description After contract creation, all tokens are allocated to the owner account through the *mint* function, which has the risk of centralization of token allocation.

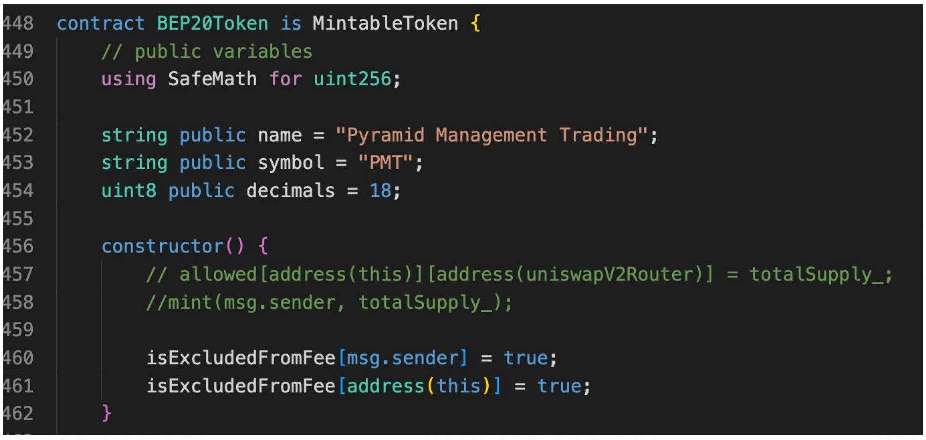


```
501     string public symbol = "PMT";
502     uint8 public decimals = 18;
503 
504     constructor() {
505 
506       totalSupply_ = 210000000 * (10 ** uint256(decimals));
507 
508       // allowed[address(this)][address(uniswapV2Router)] = totalSupply_;
509 
510       mint(msg.sender, totalSupply_);
511 
512       isExcludedFromFee[msg.sender] = true;
513       isExcludedFromFee[address(this)] = true;
514     }
515 
516     function burn(uint value) public{
517       super._burn(msg.sender,value);
518     }
519 
```

Figure 5 Source code of *constructor* function

Recommendations It is recommended to use multi-signature wallet, DAO or TimeLock to manage the pre-mint token.

Status Acknowledged. The project team deleted *mint* function in constructor, however, it still has centralization risk because owner mint all token to owner's address based on BNB chain data.



```
contract BEP20Token is MintableToken {
    // public variables
    using SafeMath for uint256;

    string public name = "Pyramid Management Trading";
    string public symbol = "PMT";
    uint8 public decimals = 18;

    constructor() {
        // allowed[address(this)][address(uniswapV2Router)] = totalSupply_;
        //mint(msg.sender, totalSupply_);

        isExcludedFromFee[msg.sender] = true;
        isExcludedFromFee[address(this)] = true;
    }
```

Figure 6 Source code of *constructor* function

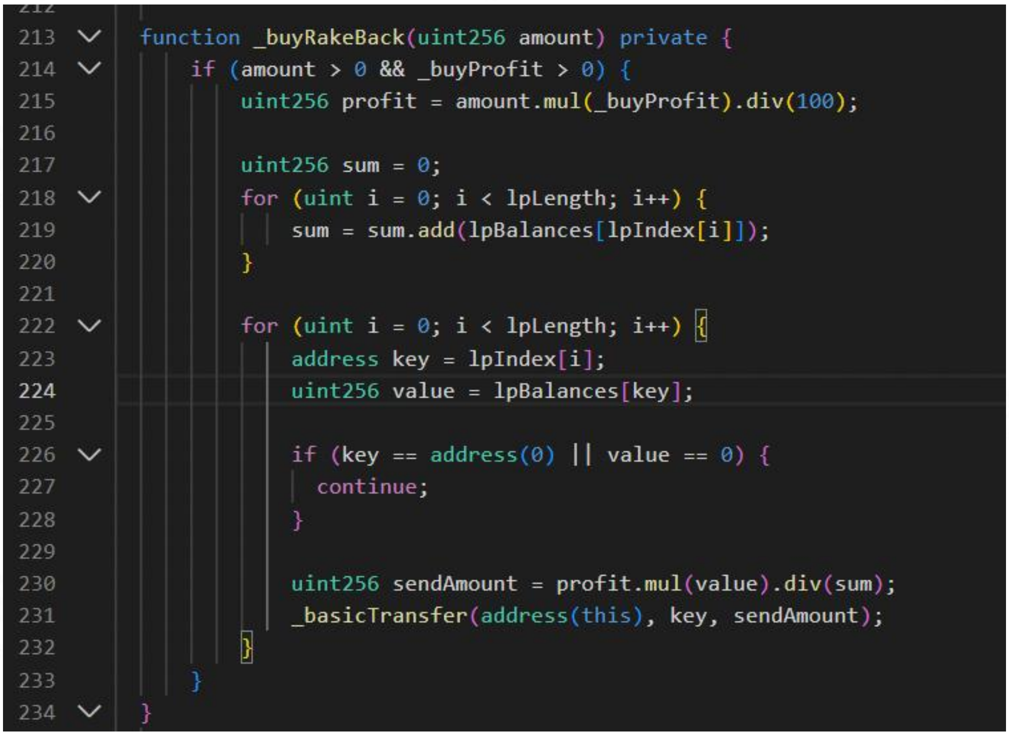
[PMT-3] Flash loan risk

Severity Level Low

Type Business Security

Lines PMT.sol #L213-234

Description If the attacker obtains a large amount of PMT tokens through flash loans and then returns them to the pool, it will be considered as adding liquidity. Then the attacker's lpbalance will increase heavily which will lead to a high proportion of liquidity rewards. The attacker will get a large percentage of rewards from every transaction purchase tokens from the liquidity pool.

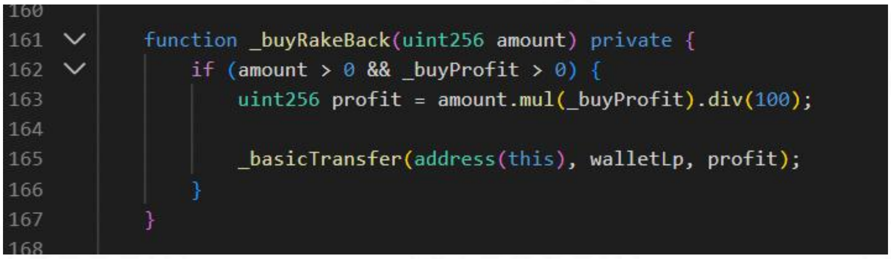


```
213  function _buyRakeBack(uint256 amount) private {
214      if (amount > 0 && _buyProfit > 0) {
215          uint256 profit = amount.mul(_buyProfit).div(100);
216
217          uint256 sum = 0;
218          for (uint i = 0; i < lpLength; i++) {
219              sum = sum.add(lpBalances[lpIndex[i]]);
220          }
221
222          for (uint i = 0; i < lpLength; i++) {
223              address key = lpIndex[i];
224              uint256 value = lpBalances[key];
225
226              if (key == address(0) || value == 0) {
227                  continue;
228              }
229
230              uint256 sendAmount = profit.mul(value).div(sum);
231              _basicTransfer(address(this), key, sendAmount);
232          }
233      }
234  }
```

Figure 7 Source code of *_buyRakeBack* function

Recommendations It is recommended to prohibit contract address participation to update lpbalance list.

Status Fixed. The project team deleted related the code of design logic.



```
161  function _buyRakeBack(uint256 amount) private {
162      if (amount > 0 && _buyProfit > 0) {
163          uint256 profit = amount.mul(_buyProfit).div(100);
164
165          _basicTransfer(address(this), walletLp, profit);
166      }
167  }
```

Figure 8 Source code of *_buyRakeBack* function

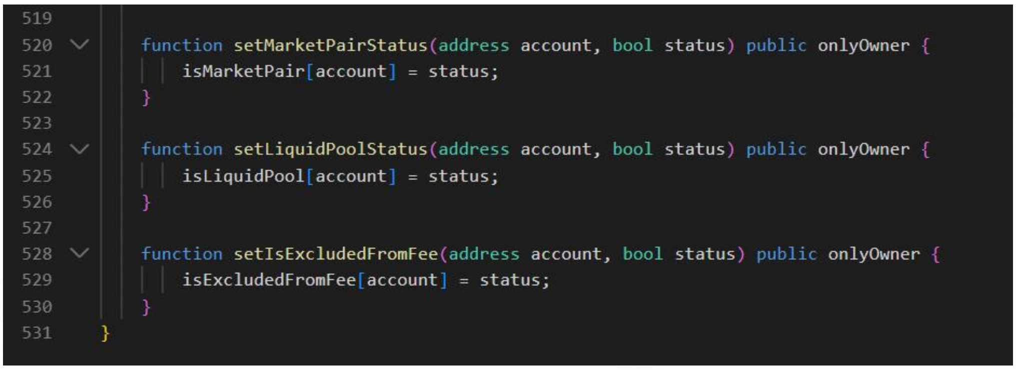
[PMT-4] Missing trigger event

Severity Level Info

Type Coding Conventions

Lines PMT.sol #L520-531

Description Setting functions without emit event. These functions are: *setMarketPairStatus*, *setLiquidPoolStatus*, *setIsExcludedFromFee*



```
519
520         function setMarketPairStatus(address account, bool status) public onlyOwner {
521             isMarketPair[account] = status;
522         }
523
524         function setLiquidPoolStatus(address account, bool status) public onlyOwner {
525             isLiquidPool[account] = status;
526         }
527
528         function setIsExcludedFromFee(address account, bool status) public onlyOwner {
529             isExcludedFromFee[account] = status;
530         }
531     }
```

Figure 9 Source code of related function

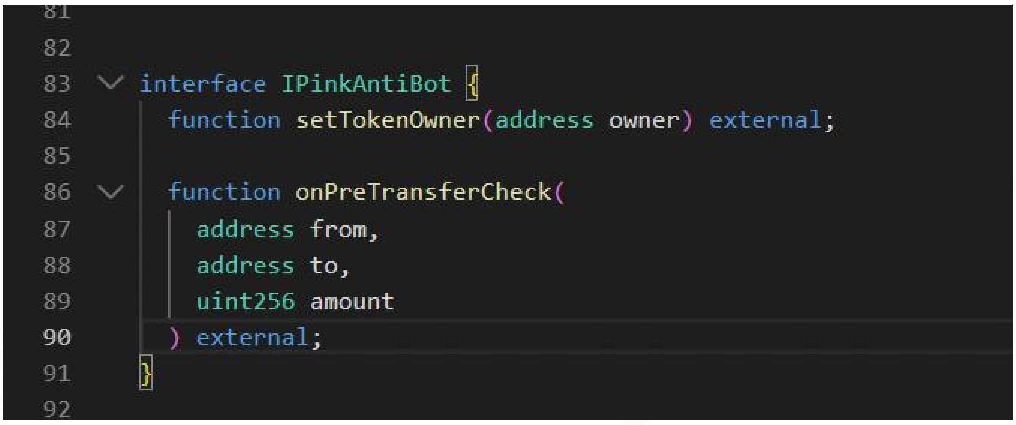
Recommendations It is recommended to declare and trigger the corresponding event.

Status Acknowledged. The project team deleted *setLiquidPoolStatus* function.

[PMT-5] Redundant code

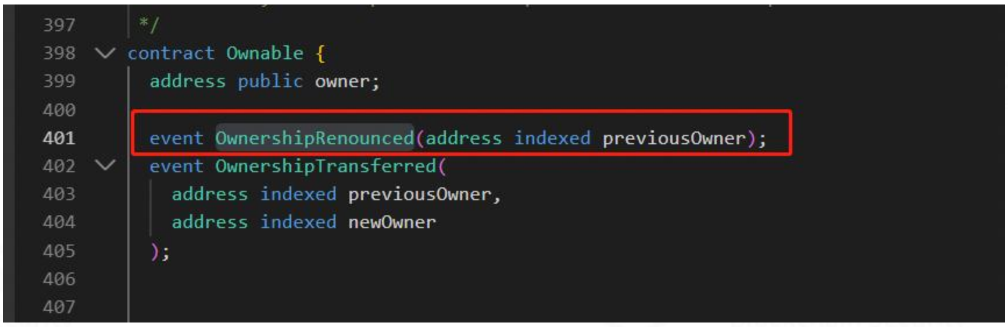
Severity Level	Info
Type	Coding Conventions
Lines	PMT.sol #L83-91, L401

Description Interface *IPinkA ntiBot* is not used. OwnershipRenounced event is useless.



```
81
82
83  interface IPinkAntiBot {
84      function setTokenOwner(address owner) external;
85
86      function onPreTransferCheck(
87          address from,
88          address to,
89          uint256 amount
90      ) external;
91  }
92
```

Figure 10 Source code of IPinkAntiBot interface



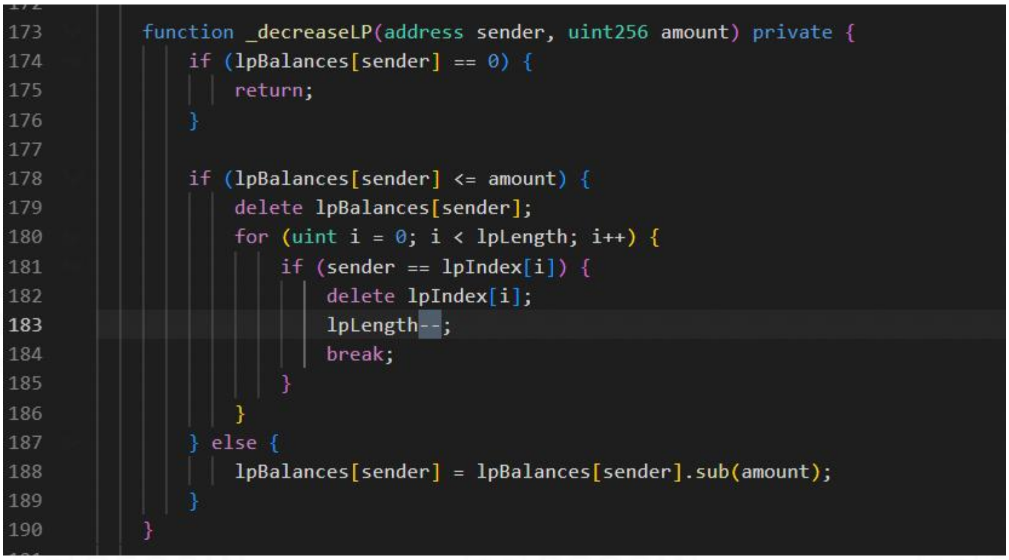
```
397  */
398  contract Ownable {
399    address public owner;
400
401    event OwnershipRenounced(address indexed previousOwner);
402    event OwnershipTransferred(
403      address indexed previousOwner,
404      address indexed newOwner
405    );
406
407
```

Figure 11 Source code of IPinkAntiBot interface

Recommendations	It is recommended to delete redundant code.
Status	Partially Fixed. Interface *IPinkA ntiBot have been deleted.*

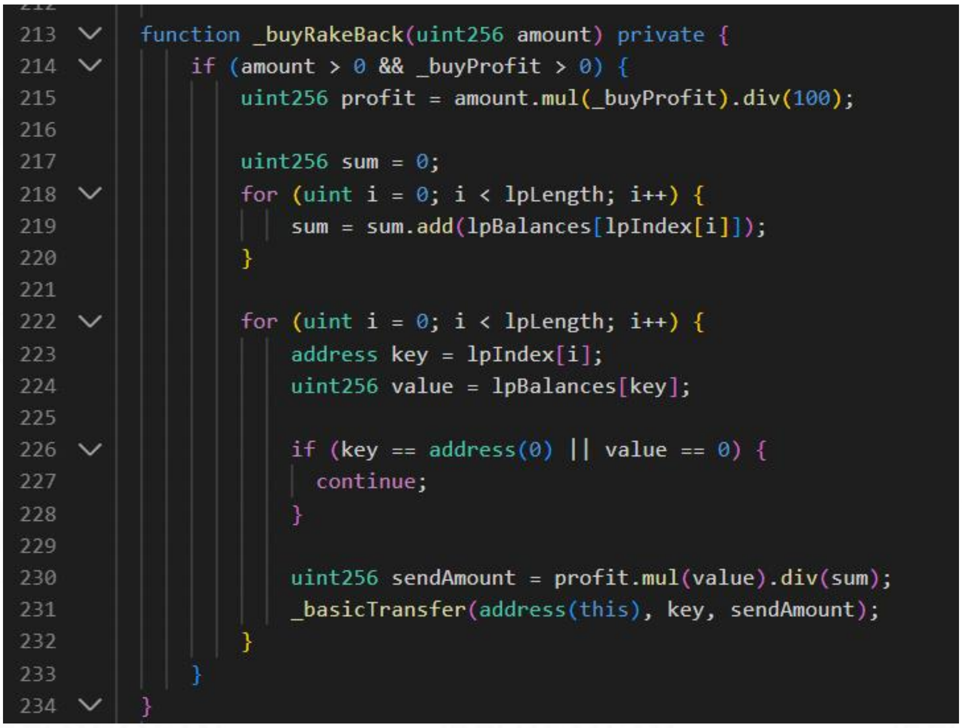
[PMT-6] Insufficient gas causes the operation to fail

Severity Level	Info
Type	Business Security
Lines	PMTToken.sol #L173-190, L213-233
Description	For loop in *_decreaseLP* and *_buyRakeBack* function will have insufficient gas problem because of large lpLength. Insufficient gas may cause the operation to fail.



```
173     function _decreaseLP(address sender, uint256 amount) private {
174         if (lpBalances[sender] == 0) {
175             return;
176         }
177 
178         if (lpBalances[sender] <= amount) {
179             delete lpBalances[sender];
180             for (uint i = 0; i < lpLength; i++) {
181                 if (sender == lpIndex[i]) {
182                     delete lpIndex[i];
183                     lpLength--;
184                     break;
185                 }
186             }
187         } else {
188             lpBalances[sender] = lpBalances[sender].sub(amount);
189         }
190     }
```

Figure 12 Source code of *_decreaseLP* function

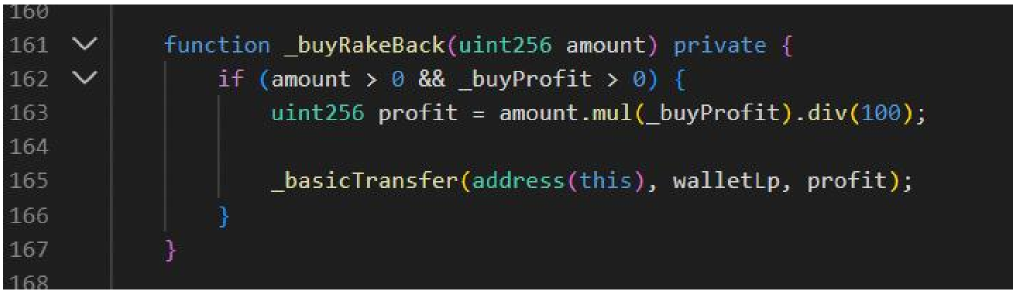


```
213     function _buyRakeBack(uint256 amount) private {
214         if (amount > 0 && _buyProfit > 0) {
215             uint256 profit = amount.mul(_buyProfit).div(100);
216 
217             uint256 sum = 0;
218             for (uint i = 0; i < lpLength; i++) {
219                 sum = sum.add(lpBalances[lpIndex[i]]);
220             }
221 
222             for (uint i = 0; i < lpLength; i++) {
223                 address key = lpIndex[i];
224                 uint256 value = lpBalances[key];
225 
226                 if (key == address(0) || value == 0) {
227                     continue;
228                 }
229 
230                 uint256 sendAmount = profit.mul(value).div(sum);
231                 _basicTransfer(address(this), key, sendAmount);
232             }
233         }
234     }
```

Figure 13 Source *code of _buyRakeBack func*tion

Recommendations It is recommended to limit the number of loop to a reasonable range.

Status Fixed. The project team deleted related the code of design logic.



```
    function _buyRakeBack(uint256 amount) private {
        if (amount > 0 && _buyProfit > 0) {
            uint256 profit = amount.mul(_buyProfit).div(100);

            _basicTransfer(address(this), walletLp, profit);
        }
    }
```

Figure 14 Source c*ode of _buyRakeBack func*tion

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1 (Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Impact Likelihood	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	High	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.4 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.5 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Compiler Version Security
		Deprecated Items
		Redundant Code
		require/assert Usage
		Gas Consumption
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		call/delegatecall Security
		Returned Value Security
		tx.origin Usage
		Replay Attack
		Overriding Variables
		Third-party Protocol Interface Consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Asset Tradability
		Arbitrage Attack

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides invesPMT nt advice on any project, nor should it be utilized as invesPMT nt suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in Blockchain.

3.4 About BEOSIN

BEOSIN is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions.BEOSIN has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, BEOSIN has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

BEOSIN
Blockchain Security

Official Website

https://www.beosin.com

Telegram

https://t.me/+dD8Bnqd133RmNWNl

Twitter

https://twitter.com/Beosin_com

Email

Contact@beosin.com