Piston

Smart Contract Security Audit

Prepared by ShellBoxes

March 28th, 2022 - April 1st, 2022

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	Piston
Version	1.0
Classification	Public

Scope

The Piston Contract in the Piston Repository

Repo	Commit Hash
https://github.com/mydiamondteam/immunefi	e74b794772ce5190851ce4449f80e3519c86585c

Re-Audit

Repo	Commit Hash
https://github.com/mydiamondteam/immunefi	d7a03d3a8a48d7a725a709b94695da8358b8e0a2

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction

Piston engaged ShellBoxes to conduct a security assessment on the Piston beginning on March 28th, 2022 and ending April 1st, 2022. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About Piston

Piston Token is a real deflationary token that allows you to gain 1% per day up to 365% of your deposit. Just compounding you can 5x your investment and withdraw 365% of it.

Issuer	Piston
Website	`https://piston-token.com/`
Type	Solidity Smart Contract
Audit Method	Whitebox

1.2 Approach & Methodology

ShellBoxes used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

1.2.1 Risk Methodology

Vulnerabilities or bugs identified by ShellBoxes are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This frame-

work is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.

Impact				
	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low

Likelihood

2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the Piston implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Disclaimer

Except for two issues Labeled (A1 and B2), none of the issues raised in this report were resolved following the re-audit. Piston team labeled the issues as unlikely to cause harm; by doing so, they accept full responsibility in case that any of the issues described herein occur in practice.

2.3 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include , 6 high-severity, 5 medium-severity, 15 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
Fees Can Be Bypassed	HIGH	Fixed
Owner Can Disable Transfers	HIGH	Acknowledged
The Panbusdswap Pair Can Be Removed From Automatedmarketmakerpairs	HIGH	Acknowledged
The Owner Can Take The Race Contract And The Ecosystem Fees	HIGH	Acknowledged
Missing Verification In The Transfer Calls	HIGH	Fixed
The Owner Can Control The Price	HIGH	Acknowledged

Missing Value Verification	MEDIUM	Acknowledged
mintMaster Can Be Set to Any Address	MEDIUM	Acknowledged
Old Controllers Are Not Included Back In The Fees	MEDIUM	Acknowledged
Race Condition	MEDIUM	Acknowledged
Owner Can Deny The Users From deposits_BUSD	MEDIUM	Acknowledged
Missing Address Verification	LOW	Acknowledged
Approve Race	LOW	Acknowledged
For Loop Over Dynamic Array	LOW	Acknowledged
Renounce Ownership	LOW	Acknowledged
Floating Pragma	LOW	Acknowledged
Missing Address Verification	LOW	Acknowledged
Approve Race	LOW	Acknowledged
Renounce Ownership	LOW	Acknowledged
Floating Pragma	LOW	Acknowledged
Missing Address Verification	LOW	Acknowledged
Renounce Ownership	LOW	Acknowledged
Ref_Depth Should Be Lower Than 255	LOW	Acknowledged
Missing Value Verification	LOW	Acknowledged
Renounce Ownership	LOW	Acknowledged
Floating Pragma	LOW	Acknowledged

3 Finding Details

A PistonToken.sol

A.1 Fees Can Be Bypassed [HIGH]

Description:

When the fees are enabled, in every transfer, the contract takes a portion of the transferred amount as fees. Sending an amount that is lower than the totalFees and extraSellfee variables will generate a type conversion which makes the fees value equal to 0. Therefore the users can bypass the fees.

Code:

Listing 1: PistonToken.sol

```
139 if(takeFee) {
140  uint256 fees = amount.mul(totalFees).div(100);
141  if(automatedMarketMakerPairs[to]){
142   fees += amount.mul(extraSellFee).div(100);
143  }
144  amount = amount.sub(fees);
145  super._transfer(from, address(this), fees);
146 }
```

Risk Level:

Likelihood – 4

Impact - 5

Recommendation:

It is recommended to require the transferred amount to be higher than both the totalFees and extraSellfee variables.

Status - Fixed

The Piston team has fixed the issue by adding a require statement to make sure the amount is higher than the sum of the fees.

A.2 Owner Can Disable Transfers [HIGH]

Description:

The owner has the ability to disable the transfers of the users that are not excluded from the fees through the variable tradingEnabled. This represents a centralization risk where the owner have too much power over the contract.

Code:

Listing 2: PistonToken.sol

```
149 function _transfer(
150 address from,
151 address to,
152 uint256 amount
153 ) internal override {
154  require(from != address(0), "ERC20: transfer from the zero address");
155  require(to != address(0), "ERC20: transfer to the zero address");
156  require(!_isBlacklisted[from] && !_isBlacklisted[to], 'Blacklisted
        ↪ address');

158  if(!_isExcludedFromFees[from] && !_isExcludedFromFees[to]){
159  require(tradingEnabled, "Trading not enabled");
160  }
```

Listing 3: PistonToken.sol

```
110 function setTradingEnabled(bool _enabled) external onlyOwner{
111  tradingEnabled = _enabled;
112  swapEnabled = _enabled;
113 }
```

Risk Level:

Likelihood – 3
Impact - 5

Recommendation:

It is recommended to only allow the transition of the tradingEnabled variable from false to true and prevent the other way around.

Status - Acknowledged

The Piston team has acknowledged the risk.

A.3 The Panbusdswap Pair Can Be Removed From Automatedmarketmakerpairs [HIGH]

Description:

The setAutomatedMarketMakerPair() function contains a restriction that issupposed to prevent the owner from removing the panbusdswap pair. Therestriction can be bypassed by modifying the uniswapV2Pair to another address, then the owner can remove the panbusdswap pair.

Code:

Listing 4: PistonToken.sol

```
100 function setAutomatedMarketMakerPair(address pair, bool value) public
        ↪ onlyOwner {
101  require(pair != uniswapV2Pair, "PISTON: The PanBUSDSwap pair cannot be
        ↪ removed from automatedMarketMakerPairs");
102  _setAutomatedMarketMakerPair(pair, value);
103 }
```

Listing 5: PistonToken.sol

```
125 function setUniswapV2PairAndController(address _uniswapV2Pair, address
        ↪ _controller) external onlyOwner{
126  uniswapV2Pair=address(_uniswapV2Pair);
127  controller=address(_controller);
128  excludeFromFees(controller, true);
129 }
```

Risk Level:

Likelihood – 3
Impact - 4

Recommendation:

It is recommended to get the address of the pair in uniswap using create2 function and remove the setter as the value will not change over time.

Status - Acknowledged

The Piston team has acknowledged the risk.

A.4 Missing Value Verification [MEDIUM]

Description:

Certain functions lack a safety check in the values, the values of the arguments should be verified to allow only the ones that go with the contract's logic. ThemaxBuyAmount, maxWalletBalance and maxSellAmount variables should be different from zero, otherwise users will not be able to transfer tokens.

Code:

Listing 6: PistonToken.sol

```
124 function setMaxBuyAmount(uint256 amount) external onlyOwner{
125  maxBuyAmount = amount * 10**18;
126 }

128 function setMaxWalletBalance(uint256 amount) external onlyOwner{
129  maxWalletBalance = amount * 10**18;
130 }

132 function setMaxSellAmount(uint256 amount) external onlyOwner{
133  maxSellAmount = amount * 10**18;
134 }
```

Risk Level:

Likelihood – 2
Impact - 4

Recommendation:

It is recommended to verify the values provided in the arguments. The concerns can be resolved by utilizing a require statement.

Status - Acknowledged

The Piston team has acknowledged the risk.

A.5 mintMaster Can Be Set to Any Address [MEDIUM]

Description:

It is mentioned in the spec that the mintAddress will be initialized with the owner's address, and later it will be modified to the piston race contract. The code does not match the spec, the code only covers the modification of the mintAddress and nothing ensure that the address will be the address of the race contract.

Code:

Listing 7: PistonToken.sol

```
144 function setMintMasterAddress(address _value) external {
145  require(msg.sender == mintMaster, "only the current mint master is
        ↪ allowed to do this");
146  mintMaster = _value;
147 }
```

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

It is recommended to ensure that the mintAddress can only be modified to the race contract address.

Status - Acknowledged

The Piston team has acknowledged the risk.

A.6 Old Controllers Are Not Included Back In The Fees [MEDIUM]

Description:

When modifying the controller address the new controller address is excluded from the fees, but the previous controller is still excluded from the fees.

Code:

Listing 8: PistonToken.sol

```
86 function setUniswapV2PairAndController(address _uniswapV2Pair, address
       ↪ _controller) external onlyOwner{
87  uniswapV2Pair=address(_uniswapV2Pair);
88  controller=address(_controller);
89  excludeFromFees(controller, true);
90 }
```

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

It is recommended to include the previous controller back in the fees.

Status - Acknowledged

The Piston team has acknowledged the risk

A.7 Missing Address Verification [LOW]

Description:

Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, some of the contract's functionality may become inaccessible.

Code:

Listing 9: PistonToken.sol

```
86 function setUniswapV2PairAndController(address _uniswapV2Pair, address
       ↪ _controller) external onlyOwner{
```

```
87  uniswapV2Pair=address(_uniswapV2Pair);
88  controller=address(_controller);
89  excludeFromFees(controller, true);
90 }
```

Listing 10: PistonToken.sol

```
144 function setMintMasterAddress(address _value) external {
145  require(msg.sender == mintMaster, "only the current mint master is
        ↪ allowed to do this");
146  mintMaster = _value;
147 }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

It is recommended to make sure the addresses provided in the arguments are different from the address(0).

Status - Acknowledged

The Piston team has acknowledged the risk.

A.8 Approve Race [LOW]

Description:

The standard ERC20 implementation contains a widely-known racing condition in its approve function, wherein a spender is able to witness the token owner broadcast a transaction altering their approval and quickly sign and broadcast a transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the

spender's transaction is validated before the owner's, the spender will be able to get both approval amounts of both transactions.

Code:

Listing 11: PistonToken.sol

```
contract PistonToken is Initializable, ERC20Upgradeable,
    ↪ OwnableUpgradeable {
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

It is recommended to use the increaseAllowance() and decreaseAllowance() function to override the approval amount instead of the approve() function.

Status - Acknowledged

The Piston team has acknowledged the risk.

A.9 For Loop Over Dynamic Array [LOW]

Description:

When smart contracts are deployed or their associated functions are invoked, the execution of these operations always consumes a certain quantity of gas, according to the amount of computation required to accomplish them. Modifying an unknown-size array that grows in size over time can result in a Denial of Service attack. Simply by having an excessively huge array, users can exceed the gas limit, therefore preventing the transaction from ever succeeding.

Code:

Listing 12: PistonToken.sol

```
79 function excludeMultipleAccountsFromFees(address[] calldata accounts,
       ↪ bool excluded) public onlyOwner {
80  for(uint256 i = 0; i < accounts.length; i++) {
81   _isExcludedFromFees[accounts[i]] = excluded;
82  }
83  emit ExcludeMultipleAccountsFromFees(accounts, excluded);
84 }
```

Risk Level:

Likelihood – 2
Impact - 2

Recommendation:

Avoid actions that involve looping across the entire data structure. If you really must loop over an array of unknown size, arrange for it to consume many blocs and thus multiple transactions.

Status - Acknowledged

The Piston team has acknowledged the risk

A.10 Renounce Ownership [LOW]

Description:

Typically, the contract's owner is the account that deploys the contract. As a result, the owner can perform certain privileged activities. The renounceOwnership function is used in smart contracts to renounce ownership. However, if the contract's ownership has never been transferred before renouncing it, it will never have an Owner, which may result in a denial of service.

Code:

Listing 13: PistonToken.sol

```
9 contract PistonToken is Initializable, ERC20Upgradeable,
      ↪ OwnableUpgradeable {
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

It is advised that the Owner cannot call renounceOwnership without first transferring ownership to a different address. Additionally, if a multi-signature wallet is utilized, executing the renounceOwnership method will require two or more users to sign the transaction. Alternatively, the Renounce Ownership functionality can be disabled by overriding it.

Status - Acknowledged

The Piston team has acknowledged the risk.

A.11 Floating Pragma [LOW]

Description:

The contract makes use of the floating-point pragma 0.8.9. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version, that may introduce issues in the contract system.

Code:

Listing 14: PistonToken.sol

```
1 // SPDX-License-Identifier: MIT
```

```
2 pragma solidity ^0.8.9;
```

Risk Level:

Likelihood – 2
Impact - 2

Recommendation:

Consider locking the pragma version. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status - Acknowledged

The Piston team has acknowledged the risk.

B PistonTokenController.sol

B.1 The Owner Can Take The Race Contract And The Ecosystem Fees [HIGH]

Description:

The swapAndLiquify() function transfers the fees to the ecosystem and the race contract. However, the addresses of the ecosystem wallet and the race contract can be modified by the owner. Thus, the owner can set the addresses to his own wallet and get 70% of the fees.

Code:

Listing 15: PistonTokenController.sol

```
86 IERC20Upgradeable(BUSD).transfer(_ecosystemWalletAddress,
       ↪ IERC20Upgradeable(BUSD).balanceOf(address(this)));
```

```
88 // send 10% to burn
89 pistonToken.transfer(deadWallet, forBurn);

91 //remaining tokens to race contract - around 40%
92 pistonToken.transfer(_raceContractAddress,
```

Listing 16: PistonTokenController.sol

```
61 function setContracts(address ecosystemWalletAddress, address
       ↪ raceContractAddress) external onlyOwner {
62  _ecosystemWalletAddress = ecosystemWalletAddress;
63  _raceContractAddress = raceContractAddress;
64 }
```

Risk Level:

Likelihood – 3
Impact - 5

Recommendation:

It is recommended to initialize the addresses correctly in the constructor, then remove the setters to prevent changing these addresses.

Status - Acknowledged

The Piston team has acknowledged the risk.

B.2 Missing Verification In The Transfer Calls [HIGH]

Description:

The ERC20 standard token implementation functions return the transaction status as a Boolean. It is a good practice to check for the return status of the function call to ensure that the transaction has passed successfully. It is the developer's responsibility to enclose these function calls with require() to ensure that, when the intended ERC20 function call

returns false, the caller transaction also fails. However, it is mostly missed by developers when they carry out checks in effect, the transaction would always succeed, even if the token transfer did not.

Code:

Listing 17: PistonTokenController.sol

```
86 IERC20Upgradeable(BUSD).transfer(_ecosystemWalletAddress,
       ↪ IERC20Upgradeable(BUSD).balanceOf(address(this)));

88 // send 10% to burn
89 pistonToken.transfer(deadWallet, forBurn);

91 //remaining tokens to race contract - around 40%
92 pistonToken.transfer(_raceContractAddress,
```

Risk Level:

Likelihood – 3
Impact - 5

Recommendation:

Use the safeTransfer function from the safeERC20 implementation, or put the transfer call inside an assert or require statement to verify that the transfer has passed successfully.

Status - Fixed

The Piston team has fixed the issue by wrapping the transfer calls inside a require statement to make sure the transfer has passed successfully.

B.3 Missing Address Verification [LOW]

Description:

Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, some of the contract's functionality may become inaccessible.

Code:

Listing 18: PistonTokenController.sol

```
34 function initialize(address _Piston) public virtual initializer {

36  __Ownable_init();

38  pistonToken = IToken(address(_Piston));
```

Listing 19: PistonTokenController.sol

```
61 function setContracts(address ecosystemWalletAddress, address
      ↪ raceContractAddress) external onlyOwner {
62  _ecosystemWalletAddress = ecosystemWalletAddress;
63  _raceContractAddress = raceContractAddress;
64 }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

It is recommended to make sure the addresses provided in the arguments are different from the address(0).

Status - Acknowledged

The Piston team has acknowledged the risk.

B.4 Approve Race [LOW]

Description:

The standard ERC20 implementation contains a widely-known racing condition in its approve function, wherein a spender is able to witness the token owner broadcast a transaction altering their approval and quickly sign and broadcast a transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the spender's transaction is validated before the owner's, the spender will be able to get both approval amounts of both transactions.

Code:

Listing 20: PistonTokenController.sol

```
11 contract PistonTokenController is Initializable, ERC20Upgradeable,
       ↪ OwnableUpgradeable {
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

It is recommended to use the increaseAllowance() and decreaseAllowance() function to override the approval amount instead of the approve() function.

Status - Acknowledged

The Piston team has acknowledged the risk.

B.5 Renounce Ownership [LOW]

Description:

Typically, the contract's owner is the account that deploys the contract. As a result, the owner can perform certain privileged activities. The renounceOwnership function is used in smart contracts to renounce ownership. However, if the contract's ownership has never been transferred before renouncing it, it will never have an Owner, which may result in a denial of service.

Code:

Listing 21: PistonTokenController.sol

```
11 contract PistonTokenController is Initializable, ERC20Upgradeable,
       ↪ OwnableUpgradeable {
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

It is advised that the Owner cannot call renounceOwnership without first transferring ownership to a different address. Additionally, if a multi-signature wallet is utilized, executing the renounceOwnership method will require two or more users to sign the transaction. Alternatively, the Renounce Ownership functionality can be disabled by overriding it.

Status - Acknowledged

The Piston team has acknowledged the risk.

B.6 Floating Pragma [LOW]

Description:

The contract makes use of the floating-point pragma 0.8.9. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version, that may introduce issues in the contract system.

Code:

Listing 22: PistonTokenController.sol

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.9;
```

Risk Level:

Likelihood – 2
Impact - 2

Recommendation:

Consider locking the pragma version. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status - Acknowledged

The Piston team has acknowledged the risk.

C PistonPriceFeed.sol

C.1 Missing Address Verification [LOW]

Description:

Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, some of the contract's functionality may become inaccessible.

Code:

Listing 23: PistonPriceFeed.sol

```
16 function setMarketPair(address value) external {
17     require(msg.sender == owner, "owner only");
18     marketPairAddress = value;
19 }
```

Listing 24: PistonPriceFeed.sol

```
21 function setOwner(address value) external {
22     require(msg.sender == owner, "owner only");
23     owner = value;
24 }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

It is recommended to make sure the addresses provided in the arguments are different from the address(0).

Status - Acknowledged

The Piston team has acknowledged the risk.

C.2 Renounce Ownership [LOW]

Description:

Typically, the contract's owner is the account that deploys the contract. As a result, the owner can perform certain privileged activities. The setOwner() function can be used to renounce the ownership if the value argument is set to address(0). Therefore, it will never have an Owner, which may result in a denial of service.

Code:

Listing 25: PistonPriceFeed.sol

```
21 function setOwner(address value) external {
22     require(msg.sender == owner, "owner only");
23     owner = value;
24 }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

It is recommended to make sure the owner can only transfer his ownership and cannot renounce it.

Status - Acknowledged

The Piston team has acknowledged the risk

D PistonRace.sol

D.1 The Owner Can Control The Price [HIGH]

Description:

Whenever the contract needs the price it calls the getPrice() function from the PistonPriceFeed contract, this contract address is modifiable by the owner. Therefore, the owner have the ability to provide the price by creating another contract and setting its address using updatePistonTokenPriceFeed() function.

Code:

Listing 26: PistonRace.sol

```
189 function updatePistonTokenPriceFeed(address priceFeedAddress, bool
    ↪ _store_busd_enabled) public onlyOwner {
190     pistonTokenPriceFeed = ITokenPriceFeed(priceFeedAddress);
191     STORE_BUSD_VALUE = _store_busd_enabled;
192 }
```

Risk Level:

Likelihood – 3
Impact - 5

Recommendation:

It is recommended to initialize the pistonTokenPriceFeed variable to the PistonPriceFeed address in the constructor and remove the setter to prevent modifying it.

Status - Acknowledged

The Piston team has acknowledged the risk.

D.2 Race Condition [MEDIUM]

Description:

The ref_bonus variable is utilized to determine the bonus value. If the user calls the deposit() function then the owner modifies the ref_bonus, there is a possibility that the owner's transaction will get mined first. If that happens the deposit function will get executed with the new value of bonus which will generate unexpected behavior from the user side.

Code:

Listing 27: PistonRace.sol

```
209 function updateRefBonus(uint256 _newRefBonus) public onlyOwner {
210     ref_bonus = _newRefBonus;
211 }
```

Risk Level:

Likelihood – 2
Impact - 4

Recommendation:

It is recommended to notify the users before modifying the ref_bonus or to add it in the arguments with a require which makes sure that the one in the contract is the same as the one provided in the arguments.

Status - Acknowledged

The Piston team has acknowledged the risk.

D.3 Owner Can Deny The Users From deposits_BUSD [MEDIUM]

Description:

The deposits_BUSD associated to the users is only incremented if the STORE_BUSD_VALUE variable is set to true. The owner can change this value to false and deny all the users from deposits_BUSD.

Code:

Listing 28: PistonRace.sol

```
304 usersRealDeposits[_addr].deposits += _total_amount;
305 if(STORE_BUSD_VALUE){
306     usersRealDeposits[_addr].deposits_BUSD += pistonTokenPriceFeed.
        ↪ getPrice(_total_amount.div(1 ether)); // new cash in BUSD
307 }
```

Risk Level:

Likelihood – 2
Impact - 3

Recommendation:

It is recommended to remove the setter associated to this variable to avoid this centralization risk.

Status - Acknowledged

The Piston team has acknowledged the risk.

D.4 Ref_Depth Should Be Lower Than 255 [LOW]

Description:

The i variable is an unit8, therefore it can take values from 0 to 255. Thus, the ref_depth should never go higher than 255, otherwise the transaction will always revert

Code:

Listing 29: PistonRace.sol

```
436 for(uint8 i = 0; i < ref_depth; i++) {
437     if(_upline == address(0)) break;

439     users[_upline].total_structure++;

441     _upline = users[_upline].upline;
442 }
```

Risk Level:

Likelihood – 2
Impact - 3

Recommendation:

Add a require statement in the setter of ref_depth to make sure the new value is less than 255.

Status - Acknowledged

The Piston team has acknowledged the risk.

D.5 Missing Value Verification [LOW]

Description:

Certain functions lack a safety check in the values, the values of the arguments should be verified to allow only the ones that go with the contract's logic. The maxBuyAmount, maxWalletBalance and maxSellAmount variables should be different from zero, otherwise users will not be able to transfer the tokens.

Code:

Listing 30: PistonRace.sol

```
184 function updateTaxes(uint256 _depositTax, uint256 _claimTax) public
        ↪ onlyOwner {
185     DepositTax = _depositTax;
186     ClaimTax = _claimTax;
187 }
```

Listing 31: PistonRace.sol

```
194 function updatePayoutRate(uint256 _newPayoutRate) public onlyOwner {
195     payoutRate = _newPayoutRate;
196 }
```

Listing 32: PistonRace.sol

```
213 function updateInitialDeposit(uint256 _newInitialDeposit) public
        ↪ onlyOwner {
214     minimumInitial = _newInitialDeposit * 1e18;
215 }
```

Risk Level:

Likelihood – 2
Impact - 3

Recommendation:

It's recommended to verify the values provided in the arguments. The concerns can be resolved by utilizing a require statement.

Status - Acknowledged

The Piston team has acknowledged the risk.

D.6 Renounce Ownership [LOW]

Description:

Typically, the contract's owner is the account that deploys the contract. As a result, the owner can perform certain privileged activities. The renounceOwnership function is used in smart contracts to renounce ownership. However, if the contract's ownership has never been transferred before renouncing it, it will never have an Owner, which may result in a denial of service.

Code:

Listing 33: PistonRace.sol

```
6 contract PistonRace is OwnableUpgradeable {
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

It is advised that the Owner cannot call renounceOwnership without first transferring ownership to a different address. Additionally, if a multi-signature wallet is utilized, executing the renounceOwnership method will require two or more users to sign the transaction. Alternatively, the Renounce Ownership functionality can be disabled by overriding it.

Status - Acknowledged

The Piston team has acknowledged the risk.

D.7 Floating Pragma [LOW]

Description:

The contract makes use of the floating-point pragma 0.8.9. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version, that may introduce issues in the contract system.

Code:

Listing 34: PistonRace.sol

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.9;
```

Risk Level:

Likelihood – 2
Impact - 2

Recommendation:

Consider locking the pragma version. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status - Acknowledged

The Piston team has acknowledged the risk.

4 Best Practices

BP.1 Unnecessary variable initialization

Description:

When a variable is declared in solidity, it gets initialized with its type's default value. Thus, there is no need to initialize a variable with the default value. Also, the swapEnabled and tradingEnabled are initialized as false, then changed to true right after.

Code:

Listing 35: PistonToken.sol

```
51 //settings
52   swapEnabled = false;
53   tradingEnabled = false;
54   swapEnabled = true;
55   tradingEnabled = true;
56   maxBuyAmount = 20000 * (10**18);
57   maxWalletBalance = 20000 * (10**18);
58   maxSellAmount = 3000 * (10**18);
59   swapTokensAtAmount = 250 * (10**18);
60   totalFees = 10;
61   extraSellFee = 0;
62   _mint(owner(), 1000000 * (10**18));
63  }
```

BP.2 Remove swapEnabled From ThesetTradingEnabled()

Description:

The setTradingEnabled() function is supposed to modify the tradingEnabled, but the code also modifies the swapEnabled variable. Therefore, it is recommended to remove the swapEnabled from the function, as it has already a separate setter.

BP.3 Implement Transfer Conditions In _beforeTokenTransfer():

Description:

It is recommended to implement all the transfer verification in before _beforeTokenTransfer() that exists in the ERC20 standard in order to separate logic and make the transfer() function code cleaner

BP.4 Deploy Contract Using Script

Description:

It is recommended to implement a script that deploys the contracts and sets their addresses in the right order to avoid any deployment errors.

BP.5 Use Ownable from openzeppelin

Description:

It is recommended to use the Ownable contract from Openzeppelin and the onlyOwner modifier as a best practice.

BP.6 Declare The Ref_Balances Array In One Line

Description:

The ref_balances array can be initialized with these specific values in one line instead of pushing each element at once.

Code:

Listing 36: PistonRace.sol

```
165     ref_balances.push(100 ether); // 1 $100 worth of PSTN
166     ref_balances.push(300 ether); // 2 $300 worth of PSTN
167     ref_balances.push(500 ether); // 3 $500 worth of PSTN
168     ref_balances.push(700 ether); // 4 $700 worth of PSTN
169     ref_balances.push(900 ether); // 5 $900 worth of PSTN
170     ref_balances.push(1100 ether); // 6 $1100 worth of PSTN
171     ref_balances.push(1300 ether); // 7 $1300 worth of PSTN
172     ref_balances.push(1500 ether); // 8 $1500 worth of PSTN
173     ref_balances.push(1700 ether); // 9 $1700 worth of PSTN
174     ref_balances.push(1900 ether); // 10 $1900 worth of PSTN
175     ref_balances.push(2100 ether); // 11 $2100 worth of PSTN
176     ref_balances.push(2300 ether); // 12 $2300 worth of PSTN
177     ref_balances.push(2500 ether); // 13 $2500 worth of PSTN
178     ref_balances.push(2700 ether); // 14 $2700 worth of PSTN
179     ref_balances.push(2900 ether); // 15 $2900 worth of PSTN
180 }
```

BP.7 Remove The Test Functions

Description:

The ref_balances array can be initialized with these specific values in one line instead of pushing each element at once.

Code:

Listing 37: PistonRace.sol

```
198 function TESTAccumulatedDiv(address _addr, uint256 _value) public/*
        ↪ onlyOwner */{
199     users[_addr].accumulatedDiv = _value;
200 }
201 function TEST_UPDATE_EJECT_DAYS() public onlyOwner {
202     userDepositEjectDays = 15 minutes;
203 }
```

BP.8 Move Interfaces To Separate Files

Description:

It is recommended to move interfaces to separate files and import them in order to optimize the size of the contract.

5 Static Analysis (Slither)

Description:

ShellBoxes expanded the coverage of the specific contract areas using automated testing methodologies. Slither, a Solidity static analysis framework, was one of the tools used. Slither was run on all-scoped contracts in both text and binary formats. This tool can be used to test mathematical relationships between Solidity instances statically and variables that allow for the detection of errors or inconsistent usage of the contracts' APIs throughout the entire codebase.

Results:

```
/// PistonPriceFeed.sol
PistonPriceFeed.setMarketPair(address).value (PistonPriceFeed.sol#16)
    ↪ lacks a zero-check on :
                - marketPairAddress = value (PistonPriceFeed.sol#18)
PistonPriceFeed.setOwner(address).value (PistonPriceFeed.sol#21) lacks a
    ↪  zero-check on :
                - owner = value (PistonPriceFeed.sol#23)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #missing-zero-address-validation

Different versions of Solidity are used:
        - Version used: ['0.8.9', '^0.8.0', '^0.8.9']
        - 0.8.9 (PistonPriceFeed.sol#2)
        - ^0.8.9 (libs/IUniswapV2Pair.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.
            ↪ sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20
            ↪ .sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/
            ↪ extensions/IERC20Metadata.sol#4)
```

```
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol
            ↪ #4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #different-pragma-directives-are-used

Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.
    ↪ sol#21-23) is never used and should be removed
ERC20._burn(address,uint256) (node_modules/@openzeppelin/contracts/token
    ↪ /ERC20/ERC20.sol#280-295) is never used and should be removed
ERC20._mint(address,uint256) (node_modules/@openzeppelin/contracts/token
    ↪ /ERC20/ERC20.sol#257-267) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dead-code

Pragma version0.8.9 (PistonPriceFeed.sol#2) necessitates a version too
    ↪ recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version^0.8.9 (libs/IUniswapV2Pair.sol#3) necessitates a version
    ↪ too recent to be trusted. Consider deploying with
    ↪ 0.6.12/0.7.6/0.8.7
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/
    ↪ ERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/
    ↪ IERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/
    ↪ extensions/IERC20Metadata.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context
    ↪ .sol#4) allows old versions
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #incorrect-versions-of-solidity
PistonPriceFeed.sol analyzed (6 contracts with 75 detectors), 13 result(
    ↪ s) found

//PistonPriceFeed.sol
```

```
PistonPriceFeed.setMarketPair(address).value (PistonPriceFeed.sol#16)
    ↪ lacks a zero-check on :
                - marketPairAddress = value (PistonPriceFeed.sol#18)
PistonPriceFeed.setOwner(address).value (PistonPriceFeed.sol#21) lacks a
    ↪  zero-check on :
                - owner = value (PistonPriceFeed.sol#23)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #missing-zero-address-validation

Different versions of Solidity are used:
        - Version used: ['0.8.9', '^0.8.0', '^0.8.9']
        - 0.8.9 (PistonPriceFeed.sol#2)
        - ^0.8.9 (libs/IUniswapV2Pair.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.
            ↪ sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20
            ↪ .sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/
            ↪ extensions/IERC20Metadata.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol
            ↪ #4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #different-pragma-directives-are-used

Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.
    ↪ sol#21-23) is never used and should be removed
ERC20._burn(address,uint256) (node_modules/@openzeppelin/contracts/token
    ↪ /ERC20/ERC20.sol#280-295) is never used and should be removed
ERC20._mint(address,uint256) (node_modules/@openzeppelin/contracts/token
    ↪ /ERC20/ERC20.sol#257-267) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dead-code
```

```
Pragma version0.8.9 (PistonPriceFeed.sol#2) necessitates a version too
   ↪ recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version^0.8.9 (libs/IUniswapV2Pair.sol#3) necessitates a version
   ↪ too recent to be trusted. Consider deploying with
   ↪ 0.6.12/0.7.6/0.8.7
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/
   ↪ ERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/
   ↪ IERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/
   ↪ extensions/IERC20Metadata.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context
   ↪ .sol#4) allows old versions
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #incorrect-versions-of-solidity
PistonPriceFeed.sol analyzed (6 contracts with 75 detectors), 13 result(
   ↪ s) found

//PistonRace.sol

Compilation warnings/errors on PistonRace.sol:
Warning: Contract code size exceeds 24576 bytes (a limit introduced in
   ↪ Spurious Dragon). This contract may not be deployable on mainnet.
   ↪  Consider enabling the optimizer (with a low "runs" value!),
   ↪ turning off revert strings, or using libraries.
 --> PistonRace.sol:6:1:
  |
6 | contract PistonRace is OwnableUpgradeable {
  | ^ (Relevant source part starts here and spans across multiple lines)
    ↪ .
```

```
PistonRace.total_bnb (PistonRace.sol#106) is never initialized. It is
    ↪ used in:
        - PistonRace.contractInfo() (PistonRace.sol#858-860)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #uninitialized-state-variables

PistonRace.unstakeBoost() (PistonRace.sol#352-397) performs a
    ↪ multiplication on the result of a division:
        -current_amount_BUSD = pistonPrice.mul(usersBoosts[_addr].
            ↪ stakedBoost_PSTN.div(1000000000000000000)) (PistonRace.sol
            ↪ #366)
PistonRace.eject() (PistonRace.sol#671-736) performs a multiplication on
    ↪  the result of a division:
        -current_amount_BUSD = pistonPrice.mul(user.userDepositsForEject[
            ↪ i].amount_PSTN.div(1000000000000000000)) (PistonRace.sol
            ↪ #685)
PistonRace.eject() (PistonRace.sol#671-736) performs a multiplication on
    ↪  the result of a division:
        -pistonPrice.mul(user.userDepositsForEject[i].amount_PSTN.div
            ↪ (1000000000000000000)) <= user.userDepositsForEject[i].
            ↪ amount_BUSD (PistonRace.sol#693)
PistonRace.eject() (PistonRace.sol#671-736) performs a multiplication on
    ↪  the result of a division:
        -ejectTaxAmount = amountAvailableForEject.div(100).mul(EjectTax)
            ↪ (PistonRace.sol#718)
PistonRace.sustainabilityFeeV2(address,uint256) (PistonRace.sol#803-807)
    ↪  performs a multiplication on the result of a division:
        -_bracket = users[_addr].payouts.add(_pendingDiv).div(
            ↪ deposit_bracket_size) (PistonRace.sol#804)
        -_bracket * 5 (PistonRace.sol#806)
PistonRace.payoutOf(address) (PistonRace.sol#810-837) performs a
    ↪ multiplication on the result of a division:
        -share = users[_addr].deposits.mul(payoutRate * 1e18).div(100e18)
            ↪ .div(86400) (PistonRace.sol#819)
```

```
        -payout = share * block.timestamp.safeSub(users[_addr].
            ↪ deposit_time) (PistonRace.sol#821)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #divide-before-multiply

Reentrancy in PistonRace.airdrop(address,uint256) (PistonRace.sol
    ↪ #865-905):
        External calls:
        - require(bool,string)(pistonToken.transferFrom(_addr,address(
            ↪ this),_amount),PISTON to contract transfer failed; check
            ↪ balance and allowance.) (PistonRace.sol#873-880)
        State variables written after the call(s):
        - users[_to].accumulatedDiv = gross_payout (PistonRace.sol#887)
        - users[_to].deposits += _realizedAmount (PistonRace.sol#890)
        - users[_to].deposit_time = block.timestamp (PistonRace.sol#891)
Reentrancy in PistonRace.deposit(address,uint256) (PistonRace.sol
    ↪ #266-329):
        External calls:
        - require(bool,string)(pistonToken.transferFrom(_addr,address(
            ↪ this),_amount),PISTON token transfer failed) (PistonRace.
            ↪ sol#294-301)
        State variables written after the call(s):
        - _deposit(_addr,_total_amount) (PistonRace.sol#311)
                - users[_addr].deposits += _amount (PistonRace.sol#452)
                - users[_addr].deposit_time = block.timestamp (PistonRace.
                    ↪ sol#453)
        - _refPayout(_addr,realizedDeposit + taxedDivs,ref_bonus) (
            ↪ PistonRace.sol#313)
                - users[_addr].ref_claim_pos = ref_depth (PistonRace.sol
                    ↪ #472)
                - users[_up].accumulatedDiv = gross_payout (PistonRace.sol
                    ↪ #483)
                - users[_up].deposits += _bonus (PistonRace.sol#484)
```

```
            - users[_up].deposit_time = block.timestamp (PistonRace.
              ↪ sol#485)
            - users[_up].match_bonus += _bonus (PistonRace.sol#489)
            - users[_addr].ref_claim_pos = ref_depth (PistonRace.sol
              ↪ #497)
            - users[_addr].ref_claim_pos += 1 (PistonRace.sol#504)
            - users[_addr].ref_claim_pos += 1 (PistonRace.sol#513)
            - users[_addr].ref_claim_pos = 0 (PistonRace.sol#517)
        - users[_addr].userDepositsForEject.push(UserDepositsForEject(
          ↪ _total_amount,pistonTokenPriceFeed.getPrice(_total_amount.
          ↪ div(1000000000000000000)),block.timestamp,false)) (
          ↪ PistonRace.sol#317-324)
        - usersRealDeposits[_addr].deposits += _total_amount (PistonRace.
          ↪ sol#304)
        - usersRealDeposits[_addr].deposits_BUSD += pistonTokenPriceFeed.
          ↪ getPrice(_total_amount.div(1000000000000000000)) (
          ↪ PistonRace.sol#306)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities-1

PistonRace.updateCompoundTax(uint256) (PistonRace.sol#221-224) contains
    ↪ a tautology or contradiction:
        - require(bool)(_newCompoundTax >= 0 && _newCompoundTax <= 20) (
          ↪ PistonRace.sol#222)
PistonRace.updateExitTax(uint256) (PistonRace.sol#226-229) contains a
    ↪ tautology or contradiction:
        - require(bool)(_newExitTax >= 0 && _newExitTax <= 20) (
          ↪ PistonRace.sol#227)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #tautology-or-contradiction

PistonRace.deposit(address,uint256).taxedDivs (PistonRace.sol#284) is a
    ↪ local variable never initialized
```

```
PistonRace.eject().amountDeposits_PSTN (PistonRace.sol#674) is a local
    ↪ variable never initialized
PistonRace.eject().amountAvailableForEject (PistonRace.sol#673) is a
    ↪ local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #uninitialized-local-variables

PistonRace.unstakeBoost() (PistonRace.sol#352-397) ignores return value
    ↪ by tokenMint.mint(address(this),differenceToMint) (PistonRace.sol
    ↪ #386)
PistonRace._claim_out(address) (PistonRace.sol#593-614) ignores return
    ↪ value by tokenMint.mint(address(this),differenceToMint) (
    ↪ PistonRace.sol#602)
PistonRace.eject() (PistonRace.sol#671-736) ignores return value by
    ↪ tokenMint.mint(address(this),differenceToMint) (PistonRace.sol
    ↪ #730)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #unused-return

PistonRace.updateTaxes(uint256,uint256) (PistonRace.sol#184-187) should
    ↪ emit an event for:
        - DepositTax = _depositTax (PistonRace.sol#185)
        - ClaimTax = _claimTax (PistonRace.sol#186)
PistonRace.updatePayoutRate(uint256) (PistonRace.sol#194-196) should
    ↪ emit an event for:
        - payoutRate = _newPayoutRate (PistonRace.sol#195)
PistonRace.updateRefDepth(uint256) (PistonRace.sol#205-207) should emit
    ↪ an event for:
        - ref_depth = _newRefDepth (PistonRace.sol#206)
PistonRace.updateRefBonus(uint256) (PistonRace.sol#209-211) should emit
    ↪ an event for:
        - ref_bonus = _newRefBonus (PistonRace.sol#210)
PistonRace.updateInitialDeposit(uint256) (PistonRace.sol#213-215) should
    ↪  emit an event for:
```

```
        - minimumInitial = _newInitialDeposit * 1e18 (PistonRace.sol#214)
PistonRace.updateMinimumAmount(uint256) (PistonRace.sol#217-219) should
    ↪ emit an event for:
        - minimumAmount = _newminimumAmount * 1e18 (PistonRace.sol#218)
PistonRace.updateCompoundTax(uint256) (PistonRace.sol#221-224) should
    ↪ emit an event for:
        - CompoundTax = _newCompoundTax (PistonRace.sol#223)
PistonRace.updateExitTax(uint256) (PistonRace.sol#226-229) should emit
    ↪ an event for:
        - ExitTax = _newExitTax (PistonRace.sol#228)
PistonRace.updateDepositBracketSize(uint256) (PistonRace.sol#231-233)
    ↪ should emit an event for:
        - deposit_bracket_size = _newBracketSize * 1000000000000000000 (
            ↪ PistonRace.sol#232)
PistonRace.updateMaxPayoutCap(uint256) (PistonRace.sol#235-237) should
    ↪ emit an event for:
        - max_payout_cap = _newPayoutCap * 1000000000000000000 (
            ↪ PistonRace.sol#236)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #missing-events-arithmetic

PistonRace.getNextUpline(address,uint256,uint256) (PistonRace.sol
    ↪ #522-546) has external calls inside a loop: _max_roll_ok = users[
    ↪ _addr].deposits.add(_bonus) < this.maxRollOf(usersRealDeposits[
    ↪ _addr].deposits) (PistonRace.sol#537)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ /#calls-inside-a-loop

Reentrancy in PistonRace._claim_out(address) (PistonRace.sol#593-614):
        External calls:
        - tokenMint.mint(address(this),differenceToMint) (PistonRace.sol
            ↪ #602)
        State variables written after the call(s):
```

```
        - usersWithdrawn[_addr].withdrawn = realizedPayout (PistonRace.
            ↪ sol#605)
        - usersWithdrawn[_addr].withdrawn_BUSD = pistonTokenPriceFeed.
            ↪ getPrice(realizedPayout.div(1000000000000000000)) (
            ↪ PistonRace.sol#606)
    Reentrancy in PistonRace._claim_out(address) (PistonRace.sol#593-614):
        External calls:
        - tokenMint.mint(address(this),differenceToMint) (PistonRace.sol
            ↪ #602)
        - require(bool)(pistonToken.transfer(address(msg.sender),
            ↪ realizedPayout)) (PistonRace.sol#609)
        State variables written after the call(s):
        - total_txs ++ (PistonRace.sol#612)
    Reentrancy in PistonRace.airdrop(address,uint256) (PistonRace.sol
        ↪ #865-905):
        External calls:
        - require(bool,string)(pistonToken.transferFrom(_addr,address(
            ↪ this),_amount),PISTON to contract transfer failed; check
            ↪ balance and allowance.) (PistonRace.sol#873-880)
        State variables written after the call(s):
        - airdrops[_addr].airdrops += _realizedAmount (PistonRace.sol
            ↪ #894)
        - airdrops[_addr].last_airdrop = block.timestamp (PistonRace.sol
            ↪ #895)
        - airdrops[_to].airdrops_received += _realizedAmount (PistonRace.
            ↪ sol#896)
        - total_airdrops += _realizedAmount (PistonRace.sol#899)
        - total_txs += 1 (PistonRace.sol#900)
    Reentrancy in PistonRace.deposit(address,uint256) (PistonRace.sol
        ↪ #266-329):
        External calls:
        - require(bool,string)(pistonToken.transferFrom(_addr,address(
            ↪ this),_amount),PISTON token transfer failed) (PistonRace.
            ↪ sol#294-301)
```

```
        State variables written after the call(s):
        - _deposit(_addr,_total_amount) (PistonRace.sol#311)
                - total_deposited += _amount (PistonRace.sol#454)
        - total_txs ++ (PistonRace.sol#327)
Reentrancy in PistonRace.stakeBoost(uint256) (PistonRace.sol#332-350):
        External calls:
        - require(bool,string)(pistonToken.transferFrom(_addr,address(
            ↪ this),_amount),PISTON to contract transfer failed; check
            ↪ balance and allowance for staking.) (PistonRace.sol
            ↪ #337-344)
        State variables written after the call(s):
        - usersBoosts[_addr].stakedBoost_PSTN += _amount (PistonRace.sol
            ↪ #345)
        - usersBoosts[_addr].last_action_time = block.timestamp (
            ↪ PistonRace.sol#346)
        - usersBoosts[_addr].stakedBoost_BUSD += pistonTokenPriceFeed.
            ↪ getPrice(_amount.div(1000000000000000000)) (PistonRace.sol
            ↪ #348)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities-2

Reentrancy in PistonRace._claim_out(address) (PistonRace.sol#593-614):
        External calls:
        - tokenMint.mint(address(this),differenceToMint) (PistonRace.sol
            ↪ #602)
        - require(bool)(pistonToken.transfer(address(msg.sender),
            ↪ realizedPayout)) (PistonRace.sol#609)
        Event emitted after the call(s):
        - Leaderboard(_addr,users[_addr].referrals,users[_addr].deposits,
            ↪ users[_addr].payouts,users[_addr].total_structure) (
            ↪ PistonRace.sol#611)
Reentrancy in PistonRace.airdrop(address,uint256) (PistonRace.sol
    ↪ #865-905):
        External calls:
```

```
        - require(bool,string)(pistonToken.transferFrom(_addr,address(
            ↪ this),_amount),PISTON to contract transfer failed; check
            ↪ balance and allowance.) (PistonRace.sol#873-880)
        Event emitted after the call(s):
        - NewAirdrop(_addr,_to,_realizedAmount,block.timestamp) (
            ↪ PistonRace.sol#903)
        - NewDeposit(_to,_realizedAmount) (PistonRace.sol#904)
Reentrancy in PistonRace.deposit(address,uint256) (PistonRace.sol
    ↪ #266-329):
        External calls:
        - require(bool,string)(pistonToken.transferFrom(_addr,address(
            ↪ this),_amount),PISTON token transfer failed) (PistonRace.
            ↪ sol#294-301)
        Event emitted after the call(s):
        - Leaderboard(_addr,users[_addr].referrals,users[_addr].deposits,
            ↪ users[_addr].payouts,users[_addr].total_structure) (
            ↪ PistonRace.sol#326)
        - MatchPayout(_up,_addr,_bonus) (PistonRace.sol#493)
                - _refPayout(_addr,realizedDeposit + taxedDivs,ref_bonus)
                    ↪ (PistonRace.sol#313)
        - NewDeposit(_addr,_amount) (PistonRace.sol#457)
                - _deposit(_addr,_total_amount) (PistonRace.sol#311)
        - NewDeposit(_up,_bonus) (PistonRace.sol#492)
                - _refPayout(_addr,realizedDeposit + taxedDivs,ref_bonus)
                    ↪ (PistonRace.sol#313)
Reentrancy in PistonRace.eject() (PistonRace.sol#671-736):
        External calls:
        - tokenMint.mint(address(this),differenceToMint) (PistonRace.sol
            ↪ #730)
        - require(bool)(pistonToken.transfer(address(msg.sender),
            ↪ amountAvailableForEject)) (PistonRace.sol#733)
        Event emitted after the call(s):
        - Ejected(msg.sender,amountAvailableForEject,block.timestamp) (
            ↪ PistonRace.sol#735)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities-3

PistonRace.eject() (PistonRace.sol#671-736) uses timestamp for
    ↪ comparisons
        Dangerous comparisons:
        - require(bool,string)(user.userDepositsForEject[0].depositTime >
            ↪  block.timestamp.sub(userDepositEjectDays),eject period is
            ↪  over) (PistonRace.sol#680)
PistonRace.payoutOf(address) (PistonRace.sol#810-837) uses timestamp for
    ↪  comparisons
        Dangerous comparisons:
        - users[_addr].payouts + payout > max_payout (PistonRace.sol#826)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #block-timestamp

AddressUpgradeable.verifyCallResult(bool,bytes,string) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol
    ↪ #174-194) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/
            ↪ utils/AddressUpgradeable.sol#186-189)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #assembly-usage

PistonRace.eject() (PistonRace.sol#671-736) compares to a boolean
    ↪ constant:
        -user.userDepositsForEject[i].ejected == false (PistonRace.sol
            ↪ #683)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #boolean-equality

Different versions of Solidity are used:
        - Version used: ['^0.8.0', '^0.8.1', '^0.8.2', '^0.8.9']
        - ^0.8.9 (PistonRace.sol#2)
```

```
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access
          ↪ /OwnableUpgradeable.sol#4)
        - ^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/
          ↪ utils/Initializable.sol#4)
        - ^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/
          ↪ AddressUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/
          ↪ ContextUpgradeable.sol#4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #different-pragma-directives-are-used

AddressUpgradeable.functionCall(address,bytes) (node_modules/
   ↪ @openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol
   ↪ #85-87) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes,string) (node_modules/
   ↪ @openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol
   ↪ #95-101) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256) (
   ↪ node_modules/@openzeppelin/contracts-upgradeable/utils/
   ↪ AddressUpgradeable.sol#114-120) is never used and should be
   ↪ removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (
   ↪ node_modules/@openzeppelin/contracts-upgradeable/utils/
   ↪ AddressUpgradeable.sol#128-139) is never used and should be
   ↪ removed
AddressUpgradeable.functionStaticCall(address,bytes) (node_modules/
   ↪ @openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol
   ↪ #147-149) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes,string) (
   ↪ node_modules/@openzeppelin/contracts-upgradeable/utils/
   ↪ AddressUpgradeable.sol#157-166) is never used and should be
   ↪ removed
AddressUpgradeable.sendValue(address,uint256) (node_modules/
   ↪ @openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol
```

```
    ↪ #60-65) is never used and should be removed
AddressUpgradeable.verifyCallResult(bool,bytes,string) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol
    ↪ #174-194) is never used and should be removed
ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/utils/ContextUpgradeable.sol#18-19) is
    ↪ never used and should be removed
ContextUpgradeable.__Context_init_unchained() (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol
    ↪ #21-22) is never used and should be removed
ContextUpgradeable._msgData() (node_modules/@openzeppelin/contracts-
    ↪ upgradeable/utils/ContextUpgradeable.sol#27-29) is never used and
    ↪  should be removed
Initializable._disableInitializers() (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/proxy/utils/Initializable.sol#131-137) is
    ↪ never used and should be removed
SafeMath.max(uint256,uint256) (PistonRace.sol#964-966) is never used and
    ↪  should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dead-code

Pragma version^0.8.9 (PistonRace.sol#2) necessitates a version too
    ↪ recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ access/OwnableUpgradeable.sol#4) allows old versions
Pragma version^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ proxy/utils/Initializable.sol#4) allows old versions
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ utils/AddressUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ utils/ContextUpgradeable.sol#4) allows old versions
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #incorrect-versions-of-solidity
```

```
OwnableUpgradeable.__gap (node_modules/@openzeppelin/contracts-
    ↪ upgradeable/access/OwnableUpgradeable.sol#94) is never used in
    ↪ PistonRace (PistonRace.sol#6-909)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #unused-state-variable

PistonRace.total_bnb (PistonRace.sol#106) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #state-variables-that-could-be-declared-constant
PistonRace.sol analyzed (9 contracts with 75 detectors), 63 result(s)
    ↪ found

///PistonTokenController.sol

PistonTokenController.swapAndLiquify() (PistonTokenController.sol#66-96)
    ↪  ignores return value by IERC20Upgradeable(BUSD).transfer(
    ↪ _ecosystemWalletAddress,IERC20Upgradeable(BUSD).balanceOf(address
    ↪ (this))) (PistonTokenController.sol#86)
PistonTokenController.swapAndLiquify() (PistonTokenController.sol#66-96)
    ↪  ignores return value by pistonToken.transfer(deadWallet,forBurn)
    ↪  (PistonTokenController.sol#89)
PistonTokenController.swapAndLiquify() (PistonTokenController.sol#66-96)
    ↪  ignores return value by pistonToken.transfer(
    ↪ _raceContractAddress,pistonToken.balanceOf(address(this))) (
    ↪ PistonTokenController.sol#92)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #unchecked-transfer

PistonTokenController.swapTokensForBUSD(uint256) (PistonTokenController.
    ↪ sol#98-113) ignores return value by pistonToken.approve(address(
    ↪ uniswapV2Router),tokenAmount) (PistonTokenController.sol#104)
PistonTokenController.addLiquidity(uint256,uint256) (
    ↪ PistonTokenController.sol#115-131) ignores return value by
```

```
    ↪ pistonToken.approve(address(uniswapV2Router),tokenAmount) (
    ↪ PistonTokenController.sol#117)
PistonTokenController.addLiquidity(uint256,uint256) (
    ↪ PistonTokenController.sol#115-131) ignores return value by
    ↪ IERC20Upgradeable(BUSD).approve(address(uniswapV2Router),
    ↪ busdAmount) (PistonTokenController.sol#118)
PistonTokenController.addLiquidity(uint256,uint256) (
    ↪ PistonTokenController.sol#115-131) ignores return value by
    ↪ uniswapV2Router.addLiquidity(address(pistonToken),BUSD,
    ↪ tokenAmount,busdAmount,0,0,owner(),block.timestamp) (
    ↪ PistonTokenController.sol#121-130)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #unused-return

PistonTokenController.setContracts(address,address).
    ↪ ecosystemWalletAddress (PistonTokenController.sol#61) lacks a
    ↪ zero-check on :
              - _ecosystemWalletAddress = ecosystemWalletAddress (
                 ↪ PistonTokenController.sol#62)
PistonTokenController.setContracts(address,address).raceContractAddress
    ↪ (PistonTokenController.sol#61) lacks a zero-check on :
              - _raceContractAddress = raceContractAddress (
                 ↪ PistonTokenController.sol#63)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #missing-zero-address-validation

Reentrancy in PistonTokenController.initialize(address) (
    ↪ PistonTokenController.sol#34-55):
        External calls:
        - uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).
           ↪ createPair(address(pistonToken),BUSD) (
           ↪ PistonTokenController.sol#44)
        State variables written after the call(s):
        - burnPercent = 10 (PistonTokenController.sol#52)
```

```
        - liquidityPercent = 20 (PistonTokenController.sol#51)
        - marketingDevPercent = 30 (PistonTokenController.sol#53)
        - racePercent = 40 (PistonTokenController.sol#54)
        - uniswapV2Router = _uniswapV2Router (PistonTokenController.sol
          ↪ #48)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities-2

AddressUpgradeable.verifyCallResult(bool,bytes,string) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol
    ↪ #174-194) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/
          ↪ utils/AddressUpgradeable.sol#186-189)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #assembly-usage

Different versions of Solidity are used:
        - Version used: ['^0.8.0', '^0.8.1', '^0.8.2', '^0.8.9']
        - ^0.8.9 (PistonTokenController.sol#2)
        - ^0.8.9 (libs/IUniswapV2Factory.sol#3)
        - ^0.8.9 (libs/IUniswapV2Pair.sol#3)
        - ^0.8.9 (libs/IUniswapV2Router.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access
          ↪ /OwnableUpgradeable.sol#4)
        - ^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/
          ↪ utils/Initializable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/
          ↪ ERC20/ERC20Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/
          ↪ ERC20/IERC20Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/
          ↪ ERC20/extensions/IERC20MetadataUpgradeable.sol#4)
        - ^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/
          ↪ AddressUpgradeable.sol#4)
```

```
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/
            ↪ ContextUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/
            ↪ math/SafeMathUpgradeable.sol#4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #different-pragma-directives-are-used

AddressUpgradeable.functionCall(address,bytes) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol
    ↪ #85-87) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes,string) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol
    ↪ #95-101) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256) (
    ↪ node_modules/@openzeppelin/contracts-upgradeable/utils/
    ↪ AddressUpgradeable.sol#114-120) is never used and should be
    ↪ removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (
    ↪ node_modules/@openzeppelin/contracts-upgradeable/utils/
    ↪ AddressUpgradeable.sol#128-139) is never used and should be
    ↪ removed
AddressUpgradeable.functionStaticCall(address,bytes) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol
    ↪ #147-149) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes,string) (
    ↪ node_modules/@openzeppelin/contracts-upgradeable/utils/
    ↪ AddressUpgradeable.sol#157-166) is never used and should be
    ↪ removed
AddressUpgradeable.sendValue(address,uint256) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol
    ↪ #60-65) is never used and should be removed
AddressUpgradeable.verifyCallResult(bool,bytes,string) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol
    ↪ #174-194) is never used and should be removed
```

```
ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/utils/ContextUpgradeable.sol#18-19) is
    ↪ never used and should be removed
ContextUpgradeable.__Context_init_unchained() (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol
    ↪ #21-22) is never used and should be removed
ContextUpgradeable._msgData() (node_modules/@openzeppelin/contracts-
    ↪ upgradeable/utils/ContextUpgradeable.sol#27-29) is never used and
    ↪  should be removed
ERC20Upgradeable.__ERC20_init(string,string) (node_modules/@openzeppelin
    ↪ /contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#55-57) is
    ↪  never used and should be removed
ERC20Upgradeable.__ERC20_init_unchained(string,string) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.
    ↪ sol#59-62) is never used and should be removed
ERC20Upgradeable._burn(address,uint256) (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#285-300)
    ↪ is never used and should be removed
ERC20Upgradeable._mint(address,uint256) (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#262-272)
    ↪ is never used and should be removed
Initializable._disableInitializers() (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/proxy/utils/Initializable.sol#131-137) is
    ↪ never used and should be removed
SafeMathUpgradeable.div(uint256,uint256,string) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/math/
    ↪ SafeMathUpgradeable.sol#191-200) is never used and should be
    ↪ removed
SafeMathUpgradeable.mod(uint256,uint256) (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/utils/math/SafeMathUpgradeable.sol#151-153)
    ↪  is never used and should be removed
SafeMathUpgradeable.mod(uint256,uint256,string) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/math/
    ↪ SafeMathUpgradeable.sol#217-226) is never used and should be
```

```
    ↪ removed
SafeMathUpgradeable.sub(uint256,uint256,string) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/math/
    ↪ SafeMathUpgradeable.sol#168-177) is never used and should be
    ↪ removed
SafeMathUpgradeable.tryAdd(uint256,uint256) (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/utils/math/SafeMathUpgradeable.sol#22-28)
    ↪ is never used and should be removed
SafeMathUpgradeable.tryDiv(uint256,uint256) (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/utils/math/SafeMathUpgradeable.sol#64-69)
    ↪ is never used and should be removed
SafeMathUpgradeable.tryMod(uint256,uint256) (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/utils/math/SafeMathUpgradeable.sol#76-81)
    ↪ is never used and should be removed
SafeMathUpgradeable.tryMul(uint256,uint256) (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/utils/math/SafeMathUpgradeable.sol#47-57)
    ↪ is never used and should be removed
SafeMathUpgradeable.trySub(uint256,uint256) (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/utils/math/SafeMathUpgradeable.sol#35-40)
    ↪ is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dead-code

Pragma version^0.8.9 (PistonTokenController.sol#2) necessitates a
    ↪ version too recent to be trusted. Consider deploying with
    ↪ 0.6.12/0.7.6/0.8.7
Pragma version^0.8.9 (libs/IUniswapV2Factory.sol#3) necessitates a
    ↪ version too recent to be trusted. Consider deploying with
    ↪ 0.6.12/0.7.6/0.8.7
Pragma version^0.8.9 (libs/IUniswapV2Pair.sol#3) necessitates a version
    ↪ too recent to be trusted. Consider deploying with
    ↪ 0.6.12/0.7.6/0.8.7
Pragma version^0.8.9 (libs/IUniswapV2Router.sol#3) necessitates a
    ↪ version too recent to be trusted. Consider deploying with
```

```
    ↪ 0.6.12/0.7.6/0.8.7
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ access/OwnableUpgradeable.sol#4) allows old versions
Pragma version^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ proxy/utils/Initializable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ token/ERC20/ERC20Upgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ token/ERC20/IERC20Upgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ token/ERC20/extensions/IERC20MetadataUpgradeable.sol#4) allows
    ↪ old versions
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ utils/AddressUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ utils/ContextUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ utils/math/SafeMathUpgradeable.sol#4) allows old versions
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #incorrect-versions-of-solidity

Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256
    ↪ ,uint256,uint256,address,uint256).amountADesired (libs/
    ↪ IUniswapV2Router.sol#12) is too similar to IUniswapV2Router01.
    ↪ addLiquidity(address,address,uint256,uint256,uint256,uint256,
    ↪ address,uint256).amountBDesired (libs/IUniswapV2Router.sol#13)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #variable-names-are-too-similar

PistonTokenController.slitherConstructorConstantVariables() (
    ↪ PistonTokenController.sol#11-152) uses literals with too many
    ↪ digits:
```

```
        - deadWallet = 0x000000000000000000000000000000000000dEaD (
            ↪ PistonTokenController.sol#18)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #too-many-digits

OwnableUpgradeable.__gap (node_modules/@openzeppelin/contracts-
    ↪ upgradeable/access/OwnableUpgradeable.sol#94) is never used in
    ↪ PistonTokenController (PistonTokenController.sol#11-152)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #unused-state-variable
PistonTokenController.sol analyzed (14 contracts with 75 detectors), 53
    ↪ result(s) found


////PistonToken.sol


Contract locking ether found:
        Contract PistonToken (PistonToken.sol#9-213) has payable
            ↪ functions:
         - PistonToken.receive() (PistonToken.sol#74-75)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #contracts-that-lock-ether

PistonToken.mintMaster (PistonToken.sol#28) is written in both
        mintMaster = owner() (PistonToken.sol#45)
        mintMaster = owner() (PistonToken.sol#49)
PistonToken.swapEnabled (PistonToken.sol#12) is written in both
        swapEnabled = false (PistonToken.sol#52)
        swapEnabled = true (PistonToken.sol#54)
PistonToken.tradingEnabled (PistonToken.sol#13) is written in both
        tradingEnabled = false (PistonToken.sol#53)
        tradingEnabled = true (PistonToken.sol#55)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #write-after-write

PistonToken.setFees(uint256,uint256) (PistonToken.sol#96-102) should
    ↪ emit an event for:
        - totalFees = _totalFees (PistonToken.sol#99)
        - extraSellFee = _extraSellFee (PistonToken.sol#100)
PistonToken.setMaxBuyAmount(uint256) (PistonToken.sol#128-130) should
    ↪ emit an event for:
        - maxBuyAmount = amount * 10 ** 18 (PistonToken.sol#129)
PistonToken.setMaxWalletBalance(uint256) (PistonToken.sol#132-134)
    ↪ should emit an event for:
        - maxWalletBalance = amount * 10 ** 18 (PistonToken.sol#133)
PistonToken.setMaxSellAmount(uint256) (PistonToken.sol#136-138) should
    ↪ emit an event for:
        - maxSellAmount = amount * 10 ** 18 (PistonToken.sol#137)
PistonToken.setSwapTokensAtAmount(uint256) (PistonToken.sol#140-142)
    ↪ should emit an event for:
        - swapTokensAtAmount = amount * 10 ** 18 (PistonToken.sol#141)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #missing-events-arithmetic

PistonToken.setUniswapV2PairAndController(address,address).
    ↪ _uniswapV2Pair (PistonToken.sol#90) lacks a zero-check on :
            - uniswapV2Pair = address(_uniswapV2Pair) (PistonToken.sol
                ↪ #91)
PistonToken.setUniswapV2PairAndController(address,address)._controller (
    ↪ PistonToken.sol#90) lacks a zero-check on :
            - controller = address(_controller) (PistonToken.sol#92)
PistonToken.setMintMasterAddress(address)._value (PistonToken.sol#148)
    ↪ lacks a zero-check on :
            - mintMaster = _value (PistonToken.sol#150)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #missing-zero-address-validation
```

```
AddressUpgradeable.verifyCallResult(bool,bytes,string) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol
    ↪ #174-194) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/
            ↪ utils/AddressUpgradeable.sol#186-189)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #assembly-usage

Different versions of Solidity are used:
        - Version used: ['^0.8.0', '^0.8.1', '^0.8.2', '^0.8.9']
        - ^0.8.9 (PistonToken.sol#2)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access
            ↪ /OwnableUpgradeable.sol#4)
        - ^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/
            ↪ utils/Initializable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/
            ↪ ERC20/ERC20Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/
            ↪ ERC20/IERC20Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/
            ↪ ERC20/extensions/IERC20MetadataUpgradeable.sol#4)
        - ^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/
            ↪ AddressUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/
            ↪ ContextUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/
            ↪ math/SafeMathUpgradeable.sol#4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #different-pragma-directives-are-used

AddressUpgradeable.functionCall(address,bytes) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol
    ↪ #85-87) is never used and should be removed
```

```
AddressUpgradeable.functionCall(address,bytes,string) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol
    ↪ #95-101) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256) (
    ↪ node_modules/@openzeppelin/contracts-upgradeable/utils/
    ↪ AddressUpgradeable.sol#114-120) is never used and should be
    ↪ removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (
    ↪ node_modules/@openzeppelin/contracts-upgradeable/utils/
    ↪ AddressUpgradeable.sol#128-139) is never used and should be
    ↪ removed
AddressUpgradeable.functionStaticCall(address,bytes) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol
    ↪ #147-149) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes,string) (
    ↪ node_modules/@openzeppelin/contracts-upgradeable/utils/
    ↪ AddressUpgradeable.sol#157-166) is never used and should be
    ↪ removed
AddressUpgradeable.sendValue(address,uint256) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol
    ↪ #60-65) is never used and should be removed
AddressUpgradeable.verifyCallResult(bool,bytes,string) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol
    ↪ #174-194) is never used and should be removed
ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/utils/ContextUpgradeable.sol#18-19) is
    ↪ never used and should be removed
ContextUpgradeable.__Context_init_unchained() (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol
    ↪ #21-22) is never used and should be removed
ContextUpgradeable._msgData() (node_modules/@openzeppelin/contracts-
    ↪ upgradeable/utils/ContextUpgradeable.sol#27-29) is never used and
    ↪  should be removed
```

```
ERC20Upgradeable._burn(address,uint256) (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#285-300)
    ↪ is never used and should be removed
Initializable._disableInitializers() (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/proxy/utils/Initializable.sol#131-137) is
    ↪ never used and should be removed
SafeMathUpgradeable.div(uint256,uint256,string) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/math/
    ↪ SafeMathUpgradeable.sol#191-200) is never used and should be
    ↪ removed
SafeMathUpgradeable.mod(uint256,uint256) (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/utils/math/SafeMathUpgradeable.sol#151-153)
    ↪  is never used and should be removed
SafeMathUpgradeable.mod(uint256,uint256,string) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/math/
    ↪ SafeMathUpgradeable.sol#217-226) is never used and should be
    ↪ removed
SafeMathUpgradeable.sub(uint256,uint256,string) (node_modules/
    ↪ @openzeppelin/contracts-upgradeable/utils/math/
    ↪ SafeMathUpgradeable.sol#168-177) is never used and should be
    ↪ removed
SafeMathUpgradeable.tryAdd(uint256,uint256) (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/utils/math/SafeMathUpgradeable.sol#22-28)
    ↪ is never used and should be removed
SafeMathUpgradeable.tryDiv(uint256,uint256) (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/utils/math/SafeMathUpgradeable.sol#64-69)
    ↪ is never used and should be removed
SafeMathUpgradeable.tryMod(uint256,uint256) (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/utils/math/SafeMathUpgradeable.sol#76-81)
    ↪ is never used and should be removed
SafeMathUpgradeable.tryMul(uint256,uint256) (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/utils/math/SafeMathUpgradeable.sol#47-57)
    ↪ is never used and should be removed
```

```
SafeMathUpgradeable.trySub(uint256,uint256) (node_modules/@openzeppelin/
    ↪ contracts-upgradeable/utils/math/SafeMathUpgradeable.sol#35-40)
    ↪ is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dead-code

Pragma version^0.8.9 (PistonToken.sol#2) necessitates a version too
    ↪ recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ access/OwnableUpgradeable.sol#4) allows old versions
Pragma version^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ proxy/utils/Initializable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ token/ERC20/ERC20Upgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ token/ERC20/IERC20Upgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ token/ERC20/extensions/IERC20MetadataUpgradeable.sol#4) allows
    ↪ old versions
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ utils/AddressUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ utils/ContextUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/
    ↪ utils/math/SafeMathUpgradeable.sol#4) allows old versions
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #incorrect-versions-of-solidity

PistonToken.initialize() (PistonToken.sol#40-67) uses literals with too
    ↪ many digits:
        - _mint(owner(),1000000 * (10 ** 18)) (PistonToken.sol#66)
PistonToken.slitherConstructorConstantVariables() (PistonToken.sol
    ↪ #9-213) uses literals with too many digits:
```

```
        - deadWallet = 0x000000000000000000000000000000000000dEaD (
            ↪ PistonToken.sol#15)
    Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
        ↪ #too-many-digits

    OwnableUpgradeable.__gap (node_modules/@openzeppelin/contracts-
        ↪ upgradeable/access/OwnableUpgradeable.sol#94) is never used in
        ↪ PistonToken (PistonToken.sol#9-213)
    Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
        ↪ #unused-state-variable
    PistonToken.sol analyzed (9 contracts with 75 detectors), 49 result(s)
        ↪ found
```

Conclusion:

Most of the vulnerabilities found by the analysis have already been addressed by the smart contract code review.

6 Conclusion

In this audit, we examined the design and implementation of Piston contract and discovered several issues of varying severity. Piston team addressed 2 issues raised in the initial report and implemented the necessary fixes, while classifying the rest as a risk with low-probability of occurrence. Shellboxes' auditors advised Piston Team to maintain a high level of vigilance and to keep those findings in mind in order to avoid any future complications.

For a Contract Audit, contact us at contact@shellboxes.com