AegisX

Smart Contract Audits | KYC

PigVault

**PALLADIUM**

Security Assessment

# PigVault Token

March 24, 2023

# Table of Contents

**1 Assessment Summary**

**2 Technical Findings Summary**

**3 Project Overview**

3.1 Token Summary

3.2 Risk Analysis Summary

3.3 Main Contract Assessed

**4 Smart Contract Risk Checks**

4.1 Mint Check

4.2 Fees Check

4.3 Blacklist Check

4.4 MaxTx Check

4.5 Pause Trade Check

**5 Contract Ownership**

**6 Liquidity Ownership**

**7 KYC Check**

**8 Smart Contract Vulnerability Checks**

8.1 Smart Contract Vulnerability Details

8.2 Smart Contract Inheritance Details

8.3 Smart Contract Privileged Functions

**9 Assessment Results and Notes(Important)**

**10 Social Media Checks(Informational)**

**11 Technical Findings Details**

**12 Disclaimer**

# Assessment Summary

This report has been prepared for PigVault Token on the BNB Chain network. AegisX provides both client-centered and user-centered examination of the smart contracts and their current status when applicable. This report represents the security assessment made to find issues and vulnerabilities on the source code along with the current liquidity and token holder statistics of the protocol.

A comprehensive examination has been performed, utilizing Cross Referencing, Static Analysis, In-House Security Tools, and line-by-line Manual Review.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Inspecting liquidity and holders statistics to inform the current status to both users and client when applicable.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Verifying contract functions that allow trusted and/or untrusted actors to mint, lock, pause, and transfer assets.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders
- Thorough line-by-line manual review of the entire codebase by industry experts.

# Technical Findings Summary

## Classification of Risk

| Severity | Description |
|---|---|
| 🔴 Critical | Risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks. |
| 🟠 Major | Risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project. |
| 🟡 Medium | Risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform |
| 🟢 Minor | Risks can be any of the above but on a smaller scale. They generally do not compromise the overall integrity of the Project, but they may be less efficient than other solutions. |
| ℹ Informational | Errors are often recommended to improve the code's style or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code. |

## Findings

| Severity | Found | Pending | Resolved |
|---|---|---|---|
| 🔴 Critical | 2 | 0 | 2 |
| 🟠 Major | 1 | 0 | 1 |
| 🟡 Medium | 2 | 0 | 2 |
| 🟢 Minor | 2 | 0 | 2 |
| ℹ Informational | 3 | 3 | 0 |
| Total | 10 | 3 | 7 |

# Project Overview

## Contract Summary

| Parameter | Result |
|---|---|
| Address | 0x2d4B29b1E1D52B93a23F41e8febeA9576dC1EeCf |
| Name | PigVault |
| Token Tracker | PigVault (PGT) |
| Decimals | 18 |
| Supply | 100,000,000 |
| Platform | BNB Chain |
| compiler | v0.8.19+commit.7dd6d404 |
| Contract Name | PigVault |
| Optimization | 200 |
| LicenseType | MIT |
| Language | Solidity |
| Codebase | https://bscscan.com/token/0x2d4B29b1E1D52B93a23F41e8febeA9576dC1EeCf#code |
| Payment Tx | |

# Project Overview

## Risk Analysis Summary

| Parameter | Result |
|---|---|
| Buy Tax | 0% (25% Max) |
| Sale Tax | 0% (25% Max) |
| Is honeypot? | Clean |
| Can edit tax? | Yes |
| Is anti whale? | No |
| Is blacklisted? | No |
| Is whitelisted? | Yes |
| Holders | N/A |
| Security Score | 94 |
| Auditor Score | 80 |
| Confidence Level | Medium |

The following quick summary it's added to the project overview; however, there are more details about the audit and its results. Please read every detail.

## Main Contract Assessed
## Contract Name

| Name | Contract | Live |
|---|---|---|
| PigVault | 0x2d4B29b1E1D52B93a23F41e8febeA9576dC1EeCf | Yes |

## TestNet Contract Assessed
## Contract Name

| Name | Contract | Live |
|---|---|---|
| PigVault | 0x41DC8CcF21b059906A2b66184711CcdAe7562947 | Yes |

## Solidity Code Provided

| SolID | File Sha-1 | FileName |
|---|---|---|
| PigVault | 7855f45165878d6698c3354001af803c38902f15 | PigVault.sol |

## Mint Check

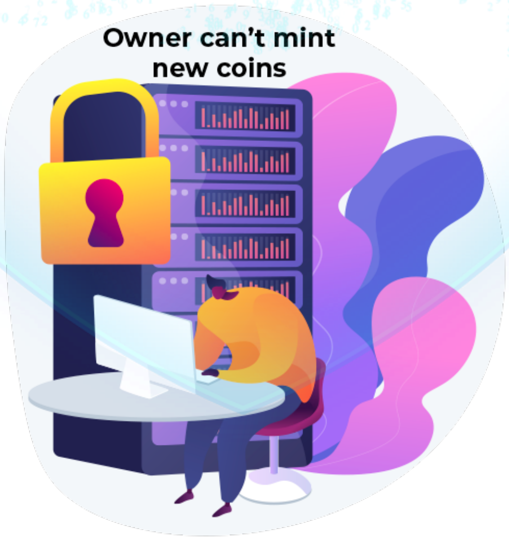
**The project owners of PigVault do not have a mint function in the contract, owners cannot mint tokens after initial deployment.**

**The Project has a Total Supply of 100,000,000 and cannot mint any more than the Max Supply.**

**Mint Notes:**

**Auditor Notes: A mint function only gets called in the constructor.**

**Project Owner Notes:**



## Fees Check

**The project owners of PigVault do not have the ability to set fees higher than 25%.**

**The team may have fees defined; however, they can't set those fees higher than 25% or may not be able to configure the same.**

**Tax Fee Notes:**

**Auditor Notes: The contract does have a tax and are capped at 25%.**

**Project Owner Notes:**

## Blacklist Check

**The project owners of PigVault do not have a blacklist function on their contract.**

**The project allow owners to transfer their tokens without any restrictions.**

**Token owner cannot blacklist the contract: Malicious or compromised owners can trap contracts relying on tokens with a blacklist.**

**Blacklist Notes:**

**Auditor Notes: The contract does not have a blacklist function.**

**Project Owner Notes:**

# MaxTx Check

## The Project Owners of PigVault cannot set max tx amount

## The Team allows any investors to swap, transfer or sell their total amount if needed.

**MaxTX Notes:**

**Auditor Notes: The contract does not have a max tx function.**

**Project Owner Notes:**

## Pause Trade Check

**The Project Owners of PigVault don't have the ability to stop or pause trading.**

**The Team has done a great job to avoid stop trading, and investors has the ability to trade at any given time without any problems**

**Pause Trade Notes:**

**Auditor Notes: The contract does not have a pause function.**

**Project Owner Notes:**

## Contract Ownership

The contract ownership of PigVault is not currently renounced. The ownership of the contract grants special powers to the protocol creators, making them the sole addresses that can call sensible ownable functions that may alter the state of the protocol.

The current owner is the address 0x8408d69bd878542c79d6fefa5b4dc88c2aca1e1c which can be viewed:
HERE

The owner wallet has the power to call the functions displayed on the privileged functions chart below, if the owner's wallet is compromised, they could exploit these privileges.

We recommend the team renounce ownership at the right time, if possible, or gradually migrate to a timelock with governing functionalities regarding transparency and safety considerations.

We recommend the team use a Multisignature Wallet if the contract is not going to be renounced; this will give the team more control over the contract.

## Liquidity Ownership

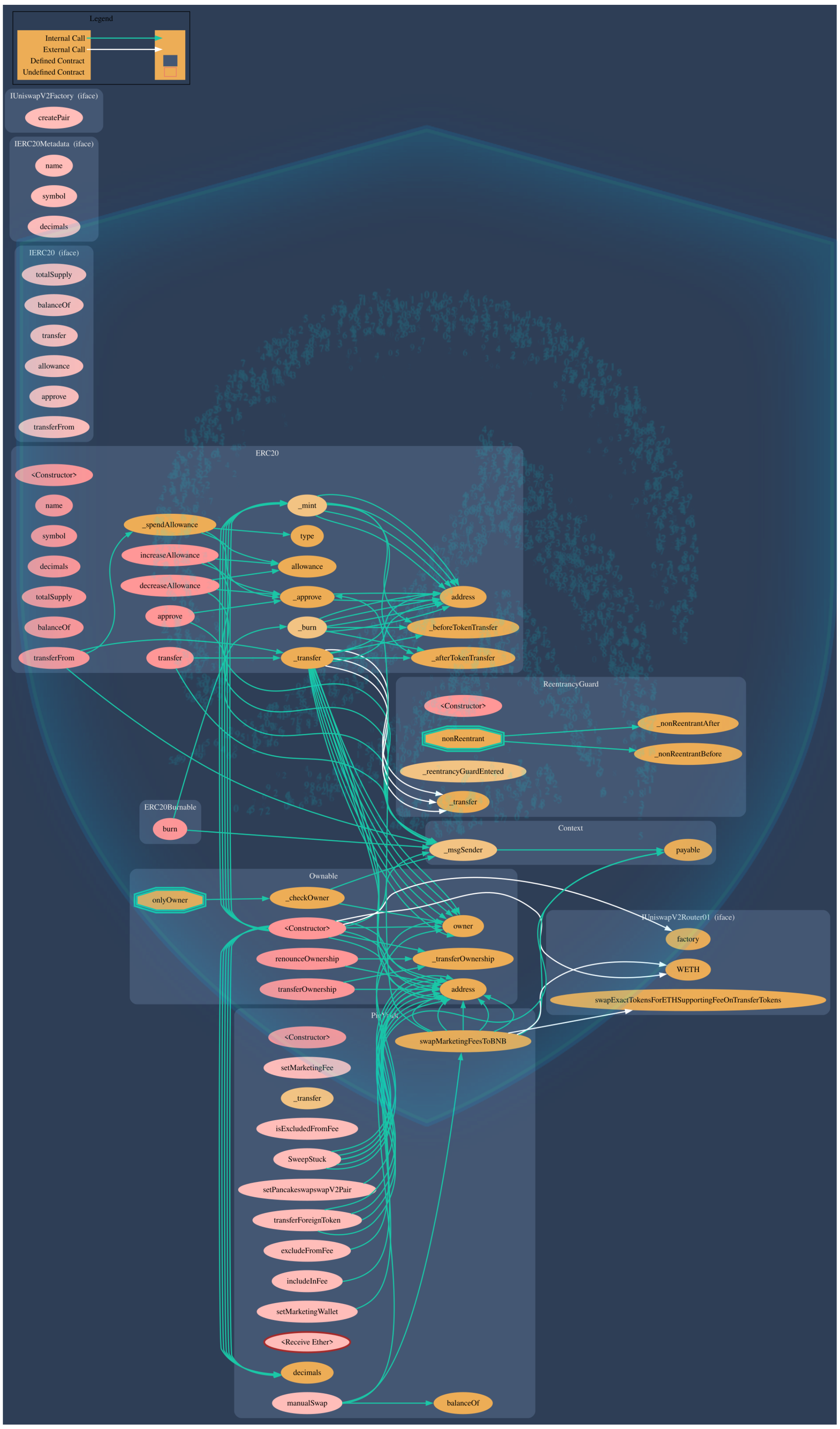
The token does not have liquidity at the moment of the audit, block 26744019

If liquidity is unlocked, then the token developers can do what is infamously known as 'rugpull'. Once investors start buying token from the exchange, the liquidity pool will accumulate more and more coins of established value (e.g., ETH or BNB or Tether). This is because investors are basically sending these tokens of value to the exchange, to get the new token. Developers can withdraw this liquidity from the exchange, cash in all the value and run off with it. Liquidity is locked by renouncing the ownership of liquidity pool (LP) tokens for a fixed time period, by sending them to a time-lock smart contract. Without ownership of LP tokens, developers cannot get liquidity pool funds back. This provides confidence to the investors that the token developers will not run away with the liquidity money. It is now a standard practice that all token developers follow, and this is what really differentiates a scam coin from a real one.

Read More

# Call Graph

The contract for PigVault has the following call graph structure.



# KYC Information

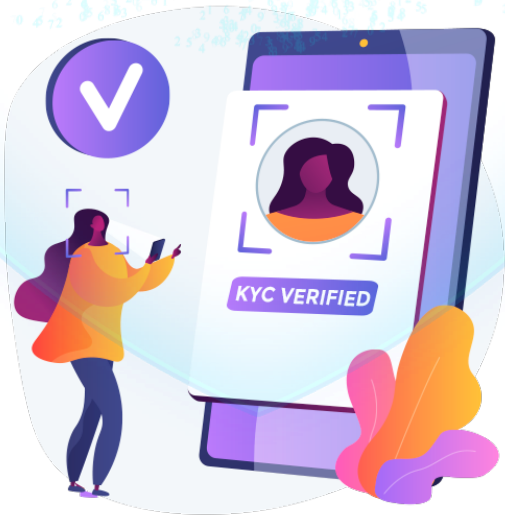
## The Project Owners of PigVault have provided KYC Documentation.

## KYC Certificate can be found on the Following: KYC Data

**KYC Information Notes:**

**Auditor Notes: KYC done by Pinksale**

**Project Owner Notes:**



## Smart Contract Vulnerability Checks

| ID | Severity | Name | File | locatio n |
|---|---|---|---|---|
| SWC-100 | Pass | Function Default Visibility | PigVault.sol | L: 0 C: 0 |
| SWC-101 | Pass | Integer Overflow and Underflow. | PigVault.sol | L: 0 C: 0 |
| SWC-102 | Pass | Outdated Compiler Version file. | PigVault.sol | L: 0 C: 0 |
| SWC-103 | Pass | A floating pragma is set. | PigVault.sol | L: 0 C: 0 |
| SWC-104 | Pass | Unchecked Call Return Value. | PigVault.sol | L: 0 C: 0 |
| SWC-105 | Pass | Unprotected Ether Withdrawal. | PigVault.sol | L: 0 C: 0 |
| SWC-106 | Pass | Unprotected SELFDESTRUCT Instruction | PigVault.sol | L: 0 C: 0 |
| SWC-107 | Pass | Read of persistent state following external call. | PigVault.sol | L: 0 C: 0 |
| SWC-108 | Pass | State variable visibility is not set.. | PigVault.sol | L: 0 C: 0 |
| SWC-109 | Pass | Uninitialized Storage Pointer. | PigVault.sol | L: 0 C: 0 |
| SWC-110 | Pass | Assert Violation. | PigVault.sol | L: 0 C: 0 |
| SWC-111 | Pass | Use of Deprecated Solidity Functions. | PigVault.sol | L: 0 C: 0 |
| SWC-112 | Pass | Delegate Call to Untrusted Callee. | PigVault.sol | L: 0 C: 0 |

| ID | Severity | Name | File | location |
|---|---|---|---|---|
| SWC-113 | Pass | Multiple calls are executed in the same transaction. | PigVault.sol | L: 0 C: 0 |
| SWC-114 | Pass | Transaction Order Dependence. | PigVault.sol | L: 0 C: 0 |
| SWC-115 | Pass | Authorization through tx.origin. | PigVault.sol | L: 0 C: 0 |
| SWC-116 | Pass | A control flow decision is made based on The block.timestamp environment variable. | PigVault.sol | L: 0 C: 0 |
| SWC-117 | Pass | Signature Malleability. | PigVault.sol | L: 0 C: 0 |
| SWC-118 | Pass | Incorrect Constructor Name. | PigVault.sol | L: 0 C: 0 |
| SWC-119 | Pass | Shadowing State Variables. | PigVault.sol | L: 0 C: 0 |
| SWC-120 | Pass | Potential use of block.number as source of randonmness. | PigVault.sol | L: 0 C: 0 |
| SWC-121 | Pass | Missing Protection against Signature Replay Attacks. | PigVault.sol | L: 0 C: 0 |
| SWC-122 | Pass | Lack of Proper Signature Verification. | PigVault.sol | L: 0 C: 0 |
| SWC-123 | Pass | Requirement Violation. | PigVault.sol | L: 0 C: 0 |
| SWC-124 | Pass | Write to Arbitrary Storage Location. | PigVault.sol | L: 0 C: 0 |
| SWC-125 | Pass | Incorrect Inheritance Order. | PigVault.sol | L: 0 C: 0 |
| SWC-126 | Pass | Insufficient Gas Griefing. | PigVault.sol | L: 0 C: 0 |

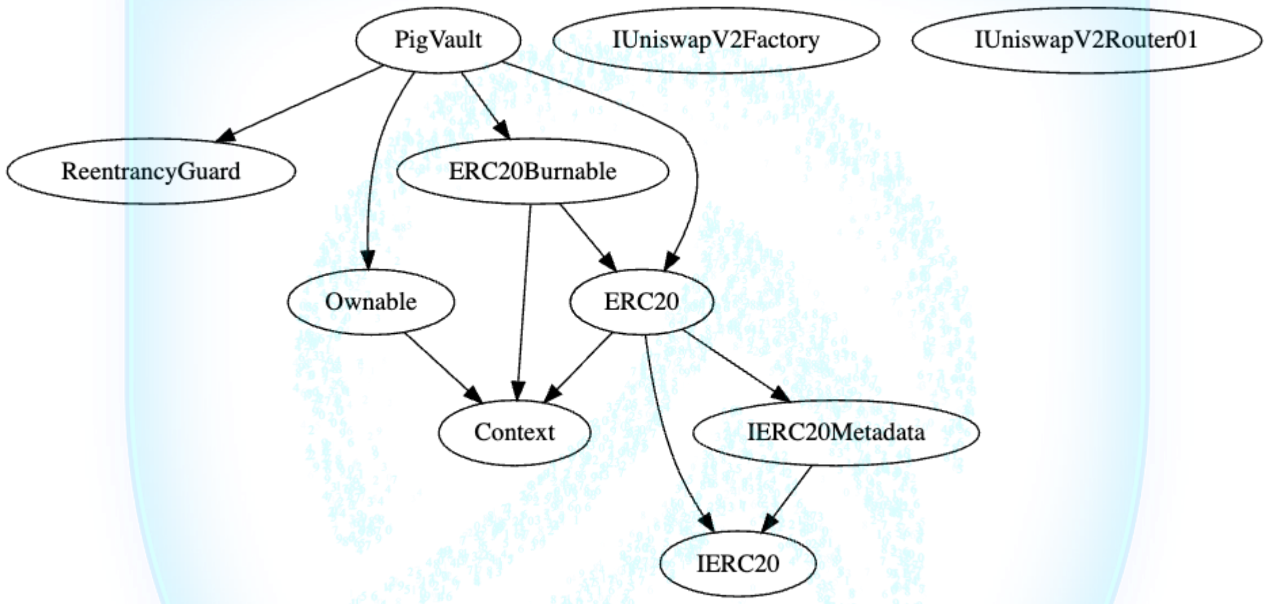
| ID | Severity | Name | File | locatio n |
|---|---|---|---|---|
| SWC-127 | Pass | Arbitrary Jump with Function Type Variable. | PigVault.sol | L: 0 C: 0 |
| SWC-128 | Pass | DoS With Block Gas Limit. | PigVault.sol | L: 0 C: 0 |
| SWC-129 | Pass | Typographical Error. | PigVault.sol | L: 0 C: 0 |
| SWC-130 | Pass | Right-To-Left-Override control character (U+202E). | PigVault.sol | L: 0 C: 0 |
| SWC-131 | Pass | Presence of unused variables. | PigVault.sol | L: 0 C: 0 |
| SWC-132 | Pass | Unexpected Ether balance. | PigVault.sol | L: 0 C: 0 |
| SWC-133 | Pass | Hash Collisions with Multiple Variable Length Arguments. | PigVault.sol | L: 0 C: 0 |
| SWC-134 | Pass | Message call with hardcoded gas amount. | PigVault.sol | L: 0 C: 0 |
| SWC-135 | Pass | Code With No Effects (Irrelevant/Dead Code). | PigVault.sol | L: 0 C: 0 |
| SWC-136 | Pass | Unencrypted Private Data On-Chain. | PigVault.sol | L: 0 C: 0 |

We scan the contract for additional security issues using MYTHX and industry-standard security scanning tools.

# Inheritance

## The contract for PigVault has the following inheritance structure.

## The Project has a Total Supply of 100,000,000



## Privileged Functions (onlyOwner)

| Function Name | Parameters | Visibility |
|---|---|---|
| renounceOwnership | | Public |
| transferOwnership | address newOwner | Public |
| setMarketingFee | uint256 _buyFee, uint256 _sellFee, uint256 _transferFee | External |
| excludeFromFee | address account | External |
| includeInFee | address account | External |
| setMarketingWallet | address _marketingWallet | External |
| setPancakeswapswapV2Pair | address _pancakeswapswapV2Pair | External |
| SweepStuck | | External |
| transferForeignToken | address _token, address _to | External |
| manualSwap | uint256 _amount | External |

## PGT-01 | Potential Sandwich Attacks.

| Category | Severity | Location | Status |
|---|---|---|---|
| Security | Informational | PigVault.sol: 804,13 | Pending |

### Description

A sandwich attack might happen when an attacker observes a transaction swapping tokens or adding liquidity without setting restrictions on slippage or minimum output amount. The attacker can manipulate the exchange rate by frontrunning (before the transaction being attacked) a transaction to purchase one of the assets and make profits by back running (after the transaction being attacked) a transaction to sell the asset. The following functions are called without setting restrictions on slippage or minimum output amount, so transactions triggering these functions are vulnerable to sandwich attacks, especially when the input amount is large:

| Function Name | Slippage |
|---|---|
| manualSwap | uint256 _amount |

### Remediation

We recommend setting reasonable minimum output amounts, instead of 0, based on token prices when calling the aforementioned functions.

### Project Action

1st review: the function manualSwap - an Oracle Implementation recommended, this may solve the problem of sandwich attack.

2nd review: the function swapMarketingFeesToBNB, which gets called by the function manualSwap, the amountOutMin value has been set to 0.001 ether. It may prevent sandwich attacks when the expected value in return is close to 0.001 BNB, but will not be of much help as the expected value becomes much larger than 0.001 BNB. However, as this function only gets called manually by the owner of the contract, the sandwich attack risk is only exposed to the owner and not to other investors, hence the severity has been marked as informational.

### References:

What Are Sandwich Attacks in DeFi — and How Can You Avoid Them?.

## PGT-02 | Function Visibility Optimization.

| Category | Severity | Location | Status |
|---|---|---|---|
| Gas Optimization | ● Minor | PigVault.sol: | ✓ Resolved |

### Description

The following functions are declared as public and are not invoked in any of the contracts contained within the projects scope:

| Function Name | Parameters | Visibility |
|---|---|---|

The functions that are never called internally within the contract should have external visibility

### Remediation

We advise that the function's visibility specifiers are set to external, and the array-based arguments change their data location from memory to calldata, optimizing the gas cost of the function.

### Project Action

1st review: please review these public functions and change those that can be changed to external.

2nd review: all functions on the main contract have been updated to external.

### References:

external vs public best practices.

## PGT-03 | Lack of Input Validation.

| Category | Severity | Location | Status |
|---|---|---|---|
| Volatile Code | ● Medium | PigVault.sol: | ✓ Resolved |

### Description

The given input is missing the check for the non-zero address and/or check for the value that is already set.

### Remediation

We advise the client to add the check for the passed-in values to prevent unexpected errors as below:

```
...
 require(receiver != address(0), "Receiver is the zero address");
 require(currentValue != NewValue, "Already set to the same value");
...
```

### Project Action

1st review: the function lock at line 628 lacks an input validation that reflects the comment at 637,9 'Contract is locked until 7 days.' The function _transfer lacks an input validation for an amount to be greater than 0. Lastly, the functions from lines 808 to 834 can use a validation whether the desired value to set is already set or not.

2nd review: the function _transfer still lacks an input validation requiring the amount to be greater than 0. Also, most of the functions from lines 831 to 901 can use a validation whether the desired value to set is already set or not. For the functions that deals with the transfer of the tokens, whether that is PGT or not, can use validations whether there is an outstanding balance to do a transfer or not.

3rd review: all of the functions have input validations in place.

## PGT-05 | Missing Event Emission.

| Category | Severity | Location | Status |
|---|---|---|---|
| Volatile Code | ● Minor | PigVault.sol: 808,5, 816,5 | ✓ Resolved |

### Description

Detected missing events for critical arithmetic parameters. There are functions that have no event emitted, so it is difficult to track off-chain changes.The linked code does not create an event for the transfer.

### Remediation

Emit an event for critical parameter changes. It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

### Project Action

1st review: the functions excludeFromFee and includeInFee lacks event emissions.

2nd review: all of the functions have event emissions.

## PGT-10 | Initial Token Distribution.

| Category | Severity | Location | Status |
|---|---|---|---|
| Centralization / Privilege | ● Minor | PigVault.sol: 690,9, 694,9 | Pending |

### Description

All of the PigVault tokens are sent to the contract deployer when deploying the contract. This could be a centralization risk as the deployer can distribute tokens without obtaining the consensus of the community.

### Remediation

We recommend the team to be transparent regarding the initial token distribution process, and the team shall make enough efforts to restrict the access of the private key.

### Project Action

1st&2nd review: the total supply goes to the contract deployer, and it was found to be in the deployer at the time of this audit.

3rd review: the total supply gets minted at the deployment of the contract and gets dispersed to five different addresses declared constant. It isn't made clear who holds the ownership of those 5 different addresses and how those tokens will be treated once the token gets listed for trade. As per the declaration on the contract, the owner and team are to get 30% of the supply, and 15% of the supply is to be injected into the liquidity. Investors are advised to verify how many tokens are in the possession of which party and trade with caution.

## PGT-11 | Contract Owner Lock Function.

| Category | Severity | Location | Status |
|---|---|---|---|
| Questionable Function | ● Medium | PigVault.sol: 586,1, 628,5 | ✓ Resolved |

### Description

The contract Ownable is a non-standard customized contract built off of a standard openzeppelin's Ownable contract. Added functions include a function named lock. Only the owner may call this function and for a set period the ownership gets set to address(0). 1st concern is that it lacks an input validation of how long this lock can last, while the function unlock has a comment in one of its validations that the contract is locked for 7 days. 2nd concern is that it can very well be deceiving that the contract has been renounced while it was simply 'locked' and the ownership can be reclaimed at any time as long as the unlock conditions have been met.

### Remediation

1st review: set a proper input validation for the function lock, and declare in comments on the contract the purpose this function serves.

### Project Action

2nd review: functions lock and unlock have been removed and the Ownable contract has been replaced with the standard openzeppelin's Ownable contract.

## PGT-12 | Centralization Risks In The onlyOwner Role(s)

| Category | Severity | Location | Status |
|---|---|---|---|
| Centralization / Privilege | Informational | PigVault.sol: 614,9 | Pending |

### Description

In the contract PigVault, the role onlyOwner has authority over the functions that lead to centralization risks.
Any compromise to the onlyOwner account(s) may allow the hacker to take advantage of this authority.

### Remediation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage.
We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked.
In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

### Project Action

_owner has a right to exclude addresses from fees.

## PGT-13 | Extra Gas Cost For User

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | 🔴 Critical | PigVault.sol: 773,13 | ✓ Resolved |

### Description

The user may trigger a tax distribution during the transfer process, which will cost a lot of gas and it is unfair to let a single user bear it.

### Remediation

We advise the client to make the owner responsible for the gas costs of the tax distribution.

### Project Action

2nd review: there are two pending issues, one critical and one medium. (1) The critical issue is that the way the function swapMarketingFeesToBNB is in place in the function _transfer logic causes any token transfers that incur fee collection to fail. (2) The medium issue is by having the function swapMarketingFeesToBNB triggered at every transfer of the tokens, it adds an extra gas burden on every investor that attempts to buy/sell/transfer the token.

3rd review: the two pending issues from the previous review have been removed as the function swapMarketingFeesToBNB is no longer being called at every transfer of the tokens.

## PGT-14 | Unnecessary Use Of SafeMath

| Category | Severity | Location | Status |
|---|---|---|---|
| Logical Issue | 🔴 Critical | PigVault.sol: 473,1, 693,5 | ✓ Resolved |

### Description

The SafeMath library is used unnecessarily. With Solidity compiler versions 0.8.0 or newer, arithmetic operations will automatically revert in case of integer overflow or underflow.
An implementation of SafeMath library is found. SafeMath library is used for uint256 type in PigVault contract.

### Remediation

We advise removing the usage of SafeMath library and using the built-in arithmetic operations provided by the Solidity programming language

### Project Action

1st review: utilizing Safemath is unnecessary and bad practice for compiler versions 0.8+. The contract does not require any complicated mathematical operation that requires a separate contract such as Safemath.

2nd review: Safemath has been removed.

## PGT-15 | Dysfunctional Exclude From Fee.

| Category | Severity | Location | Status |
|---|---|---|---|
| Dysfunctional logic | ● Major | PigVault.sol: 751,9 | ✓ Resolved |

### Description

Exclude from fee only occurs when both the sender and recipient are excluded from fee. Question to the dev: Was the excluding from fee logic found at the location intended to work only when both sender and recipient are excluded from fee?

### Remediation

1st review: if the intended way of operation of this logic is to exclude from fee when either of the sender and the recipient is excluded from fee, review the logic written, rewrite it, and perform testings on testnet to verify the fee does not get taken when just either of sender or recipient is excluded from fee.

### Project Action

2nd review: the logic has been updated.

# Social Media Checks

| Social Media | URL | Result |
|---|---|---|
| Website | https://www.pigvault.com/ | Pass |
| Telegram | https://t.me/pigvaults | Pass |
| Twitter | https://twitter.com/pigvaultbsc | Pass |
| OtherSocial | https://pigvault.gitbook.io/docs/ | Pass |

We recommend to have 3 or more social media sources including a completed working websites.

**Social Media Information Notes:**

**Auditor Notes: undefined**

**Project Owner Notes:**

# Assessment Results

## Score Results

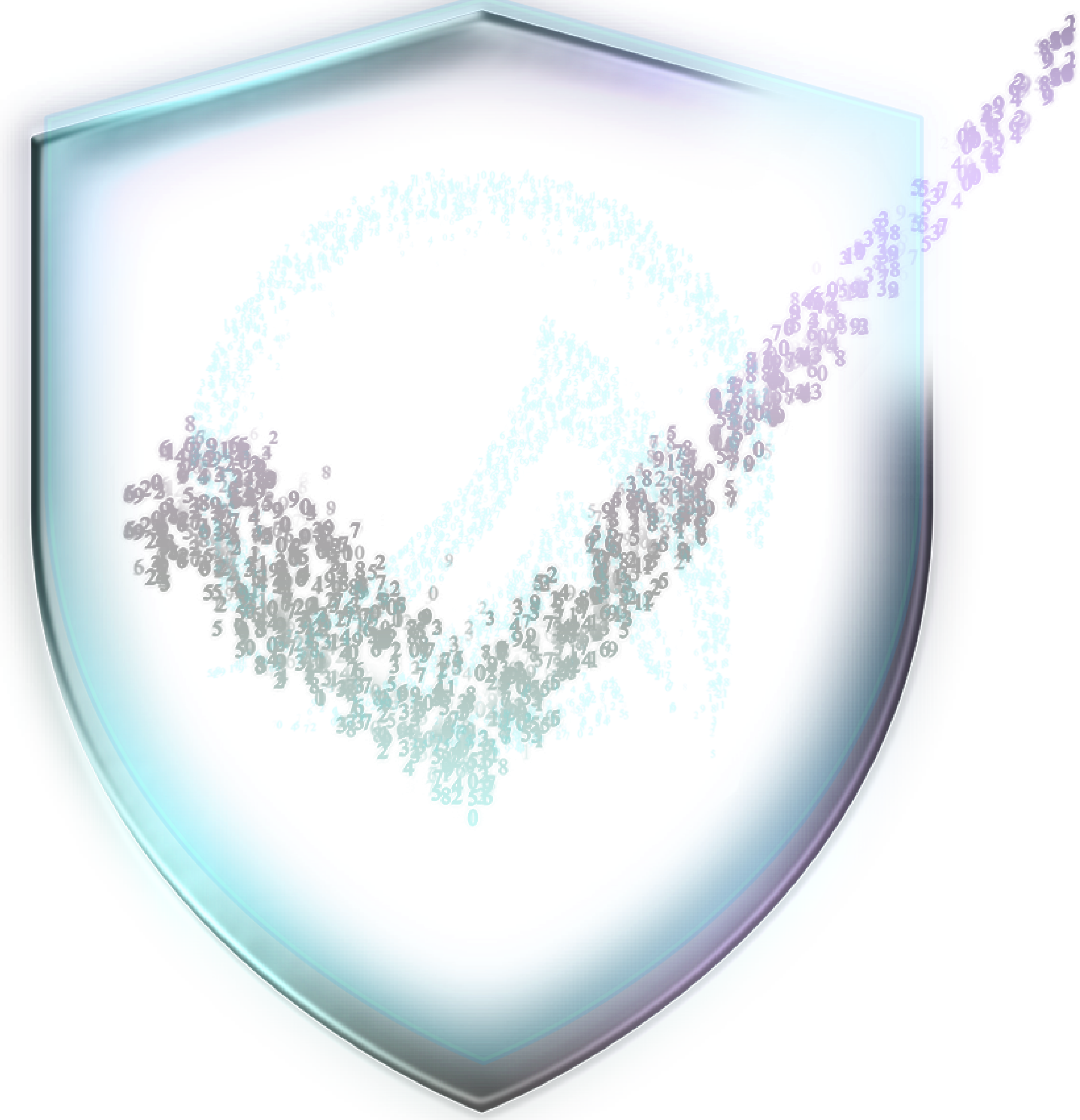
| Review | Score |
|---|---|
| Overall Score | 96/100 |
| Auditor Score | 80/100 |

| Review by Section | Score |
|---|---|
| Manual Scan Score | 18/18 |
| SWC Scan Score | 37/37 |
| Advance Check Score | 41/45 |

The maximum score is 100, however to attain that value the project must pass the reviews and provide all the data needed for the assessment. Minimum score to pass is 80 points. If a project fails to attain 80 and/or has unresolved critical and/or major and/or medium finding(s) in the Palladium tier assessments, an automatic failure is given. Read our notes and final assessment below.

## Assessment Results

## Auditor Score = 80
## Audit Passed



PASSED

AegisX

## Important Notes from the Auditor:

- 3rd(NEW): the total supply gets minted at the deployment of the contract and gets dispersed to five different addresses declared constant in the constructor. It isn't made clear who holds the ownership of those 5 different addresses and how those tokens will be treated once the token gets listed for trade. As per the declaration on the contract, the owner and team are to get 30% of the supply, and 15% of the supply is to be injected into the liquidity. Investors are advised to verify how many tokens are in the possession of which party and trade with caution.
- 
- 3rd(NEW): the two pending issues from the previous review have been removed as the function swapMarketingFeesToBNB is no longer being called at every transfer of the tokens. Regarding the potential sandwich attack vulnerability, the amountOutMin value in the function on line 804, has been set to 0.001 ether. It may prevent sandwich attacks when the expected value in return is close to 0.001 BNB, but will not be of much help as the expected value becomes much larger than 0.001

BNB. However, as this function only gets called manually by the owner of the contract, the sandwich attack risk is only exposed to the owner and not to other investors, hence the severity has been marked as informational.

- 2nd: the function swapMarketingFeesToBNB has two pending issues, one critical and one medium. (1) The critical issue is that the way the function swapMarketingFeesToBNB is in place in the function _transfer logic causes any token transfers that incur fee collection to fail. (2) The medium issue is by having the function swapMarketingFeesToBNB triggered at every transfer of the tokens, it adds an extra gas burden on every investor that attempts to buy/sell/transfer the token.

-

- 3rd(NEW): the issue remains standing.

- 2nd: NOTE: the functions SweepStuck utilizes payable(address).transfer format to transfer BNB. This is NOT a recommended way to transfer ETH anymore by solidity standard.  Advice to research what the best practice is when 'sending ether', and update the code accordingly. ie. Uniswap library's TransferHelper.sol

-

- 2nd: Ownable has been updated.
- 1st: the contract Ownable is a non-standard customized contract built off of a standard openzeppelin's Ownable contract. Added functions include a function named lock. Only the owner may call this function and for a set period the ownership gets set to address(0). 1st concern is that it lacks an input validation of how long this lock can last, while the function unlock has a comment in one of its validations that the contract is locked for 7 days. 2nd concern is that it can very well be deceiving that the contract has been renounced while it was simply 'locked' and the ownership can be reclaimed at any time as long as the unlock conditions have been met.
- 
- 2nd: Safemath has been removed.
- 1st: utilizing Safemath is unnecessary and bad practice for compiler versions 0.8+. The contract does not require any complicated mathematical operation that requires a separate contract such as Safemath.
- 
- 2nd: the logic has been updated.
- 1st: exclude from the fee only occurs when both the

sender and recipient are excluded from the fee. Question to the dev: Was the excluding from fee logic found at the location intended to work only when both sender and recipient are excluded from the fee? If the intended way of operation of this logic is to exclude from fee when either of the sender and the recipient is excluded from fee, review the logic written, rewrite it, and perform testings on testnet to verify the fee does not get taken when just either of sender or recipient is excluded from fee.

-

- 1st: the total supply goes to the contract deployer, and it was found to be in the deployer at the time of this audit.

-

- NOTE: The function setPancakeswapswapV2Pair found at line 834, has the potential to nullify the fee-taking logic at 774,13 and 778,13 when an incorrect pair address gets set.

# Appendix

## Finding Categories

### Centralization / Privilege

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

### Gas Optimization

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

### Logical Issue

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

### Control Flow

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

### Volatile Code

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

### Coding Style

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

### Inconsistency

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

### Coding Best Practices

ERC 20 Conding Standards are a set of rules that each developer should follow to ensure the code meet a set of creterias and is readable by all the developers.

## Disclaimer

AegisX has conducted an independent security assessment to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the reviewed code for the scope of this assessment. This report does not constitute agreement, acceptance, or advocation for the Project, and users relying on this report should not consider this as having any merit for financial advice in any shape, form, or nature. The contracts audited do not account for any economic developments that the Project in question may pursue, and the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude, and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are entirely free of exploits, bugs, vulnerabilities or deprecation of technologies.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence, regardless of the findings presented. Information is provided 'as is, and AegisX is under no covenant to audited completeness, accuracy, or solidity of the contracts. In no event will AegisX or its partners, employees, agents, or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions or actions with regards to the information provided in this audit report.

The assessment services provided by AegisX are subject to dependencies and are under continuing development. You agree that your access or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies with high levels of technical risk and uncertainty. The assessment reports could include false positives, negatives, and unpredictable results. The services may access, and depend upon, multiple layers of third parties.