Smart Contract
Security Audit
V1

PLSZEN Token Smart Contract

14/8/2022

https://saferico.com/

business@saferico.com
https://t.me/SFI_ANN

–

Table of Contents

Background

The purpose of the audit was to achieve the following:

- Ensure that the smart contract functions as intended.
- Identify potential security issues with the smart contract.

The information in this report should be used to understand the risk exposure of the smart contract, and as a guide to improve the security posture of the smart contract by remediating the issues that were identified.

Project Information

- **Platform**: Ethereum
- **Contract Address**: 0x5a24d7129b6f3fcad2220296df28911880ad22b0
- **Code Source:**

https://etherscan.io/address/0x5a24d7129b6f3fcad2220296df28911880ad22b0#code

Token Information

- Name: PZEN
- Total Supply: 880,000,000
- Holders: 324
- Total transactions: 1491

Contracts address deployed to test net (ETH)

PLSZEN Token smart contract on Eth test net by the auditor to test every function (ETH Test Net)

https://rinkeby.etherscan.io/address/0x4a681d6ed9fc42c56caa48172a8a39fe01f067cc

Executive Summary

According to our assessment, the customer`s solidity smart contract is **Secured**.

Well Secured	
Secured	✓
Poor Secured	
Insecure	

Automated checks are with remix IDE. All issues were performed by the team, which included the analysis of code functionality, manual audit found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the audit overview section. The general overview is presented in the Project Information section and all issues found are located in the audit overview section.

Team found 0 critical, 0 high, 0 medium, 3 low, 0 very low-level issues and 1 note in all solidity files of the contract

The files:

PZENDEPLOYERcontract.sol

File and Function Level Report

File in Scope:

Contract Name	SHA 256 hash	Contract Address
PZENDEPLOYERcontract.sol	**9e1f0fac21bf723551f8d3688a12ca063dbeae17269c82a13f0a690b705be61d**	0x5a24d7129b6f3fcad2220296df28911880ad22b0

- Contract: PZENDEPLOYERcontract
- Inherit: PZEN
- Observation: All passed including security check
- Test Report: passed
- Score: passed
- Conclusion: passed

Function	Test Result	Type / Return Type	Score
name	✓	Read / public	**Passed**
symbol	✓	Read / public	**Passed**
decimals	✓	Read / public	**Passed**
totalSupply	✓	Read / public	**Passed**
allowance	✓	Read / public	**Passed**
balanceOf	✓	Read / public	**Passed**
Owner	✓	Read / public	**Passed**
isExcludedFromFees	✓	Read / public	**Passed**
getUnlockTime	✓	Read / public	**Passed**
manger	✓	Read / public	**Passed**
isExcludedFromReward	✓	Read / public	**Passed**
reflectionFromToken	✓	Read / public	**Passed**

approve	✓	Write / public	**Passed**
TransferFrom	✓	Write / public	**Passed**
increaseAllowance	✓	Write / public	**Passed**
transfer	✓	Write / public	**Passed**
decreaseAllowance	✓	Write / public	**Passed**
withdrawLockedEth	✓	Write / public	**Passed**
lock	✓	Write / public	**Passed**
excludeFromFees	✓	Write / public	**Passed**
unLock	✓	Write / public	**Passed**
includeInReward	✓	Write / public	**Passed**
renounceOwnership	✓	Write / public	**Passed**
transferOwnership	✓	Write / public	**Passed**
burn	✓	Write / public	**Passed**
excludeFromReward	✓	Write / public	**Passed**
setPreseableEnabled	✓	Write / public	**Passed**
setRouterAddress	✓	Write / public	**Passed**
setSwapAndLiquifyEnabled	✓	Write / public	**Passed**
transferManagement	✓	Write / public	**Passed**

Issues Checking Status

No.	Issue Description	Checking Status
1	Compiler warnings.	**Passed**
2	Race conditions and Reentrancy. Cross-function race conditions.	**Passed**
3	Possible delays in data delivery.	**Passed**
4	Oracle calls.	**Passed**
5	Design Logic.	**Passed**
6	Timestamp dependence.	**Passed with notes**
7	Integer Overflow and Underflow.	**Passed**
8	DoS with Revert.	**Passed**
9	DoS with block gas limit.	**Passed with notes**
10	Methods execution permissions.	**Passed**
11	Economy model. If application logic is based on an incorrect economic model, the application would not function correctly and participants would incur financial losses. This type of issue is most often found in bonus rewards systems, Staking and Farming contracts, Vault and Vesting contracts, etc.	**Passed**
12	The impact of the exchange rate on the logic.	**Passed**
13	Private user data leaks.	**Passed**
14	Malicious Event log.	**Passed**
15	Scoping and Declarations.	**Passed**
16	Uninitialized storage pointers.	**Passed**
17	Arithmetic accuracy.	**Passed**

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can’t lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can’t have significant impact on execution
Note	Lowest-level vulnerabilities, code style violations and info statements can’t affect smart contract execution and can be ignored.

Audit Findings

Critical:

No Critical severity vulnerabilities were found.

High:

No High severity vulnerabilities were found.

Medium:

No Medium severity vulnerabilities were found.

Low:

#Pragam version not fixed

Description

It is a good practice to lock the solidity version for a live deployment (use 0.8.15 instead of ^0.8.4). contracts should be deployed with the same compiler version and flags that they have been tested the most with. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler which may have higher risks of undiscovered bugs. Contracts may also be deployed by others and the pragma indicates the compiler version intended by the original authors.

Remediation

Remove the ^ sign to lock the pragma version.

Status: Acknowledged

#Use of block.timestamp for comparisons

Description

The value of block.timestamp can be manipulated by the miner.
And conditions with strict equality is difficult to achieve - block.timestamp

Remediation

Avoid use of block.timestamp

Status: Acknowledged

#Owner privileges (In the period when the owner isn't renounced)

Description

The owner can lock and lock the smart contract.
The owner can enable or disable the trade.
The owner can include / exclude any address from Fees or Reward.

```
function setSwapAndLiquifyEnabled(bool enabled) external onlyManager {
        swapAndLiquifyEnabled = enabled;
        emit SwapAndLiquifyEnabledUpdated(swapAndLiquifyEnabled);
    }
    function setExcludedFromFee(address account, bool value) external onlyOwner {
_isExcludedFromFee[account] = value; }
function excludeFromReward(address account) external onlyOwner() {
        require(!_isExcludedFromRewards[account], "Account is not included");
        _exclude(account);
    }
function includeInReward(address account) external onlyOwner() {
        require(_isExcludedFromRewards[account], "Account is not excluded");
        for (uint256 i = 0; i < _excluded.length; i++) {
            if (_excluded[i] == account) {
                _excluded[i] = _excluded[_excluded.length - 1];
                _balances[account] = 0;
                _isExcludedFromRewards[account] = false;
                _excluded.pop();
                break;}}}
function lock(uint256 time) public virtual onlyOwner {
        _previousOwner = _owner;
        _owner = address(0);
        _lockTime = block.timestamp + time;
        emit OwnershipTransferred(_owner, address(0));
    }
    function unlock() public virtual {
        require(_previousOwner == msg.sender, "Only the previous owner can unlock
onwership");
        require(block.timestamp > _lockTime , "The contract is still locked");
        emit OwnershipTransferred(_owner, _previousOwner);
        _owner = _previousOwner;
    }
```

Remediation

Make these functions internal in next version or the team should announce the investors before change the fees and give them time if they want to use the old fees.

P.S: This issue is common to the majority of rewards smart contracts.

Status: Acknowledged.

Very Low:

No Very Low severity vulnerabilities were found.

Notes:

Constant calculations in the contract

Description

recalculated initialization will save 2847 units of gas in deployment

```
uint16 internal constant FEES_DIVISOR = 10**4;
    uint256 internal constant ZEROES = 10**DECIMALS;
    uint256 internal constant TOTAL_SUPPLY = 880 * 10**6 *10**9;
```

Recommendation

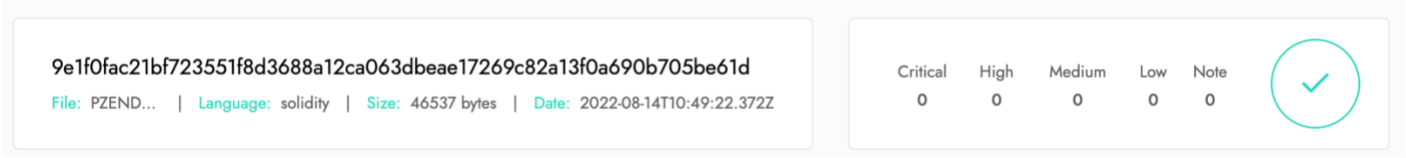
Replace the initialization as

```
uint16 internal constant FEES_DIVISOR = 100000;
    uint256 internal constant ZEROES = 10000000000;
    uint256 internal constant TOTAL_SUPPLY = 880000000000000000;
```

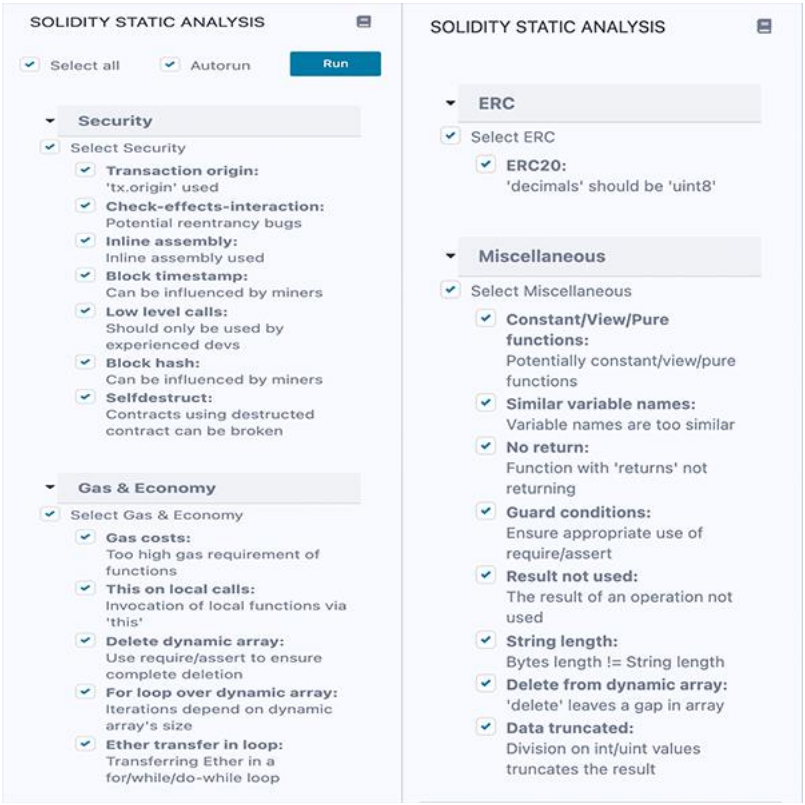
Status Acknowledged.

Automatic Testing

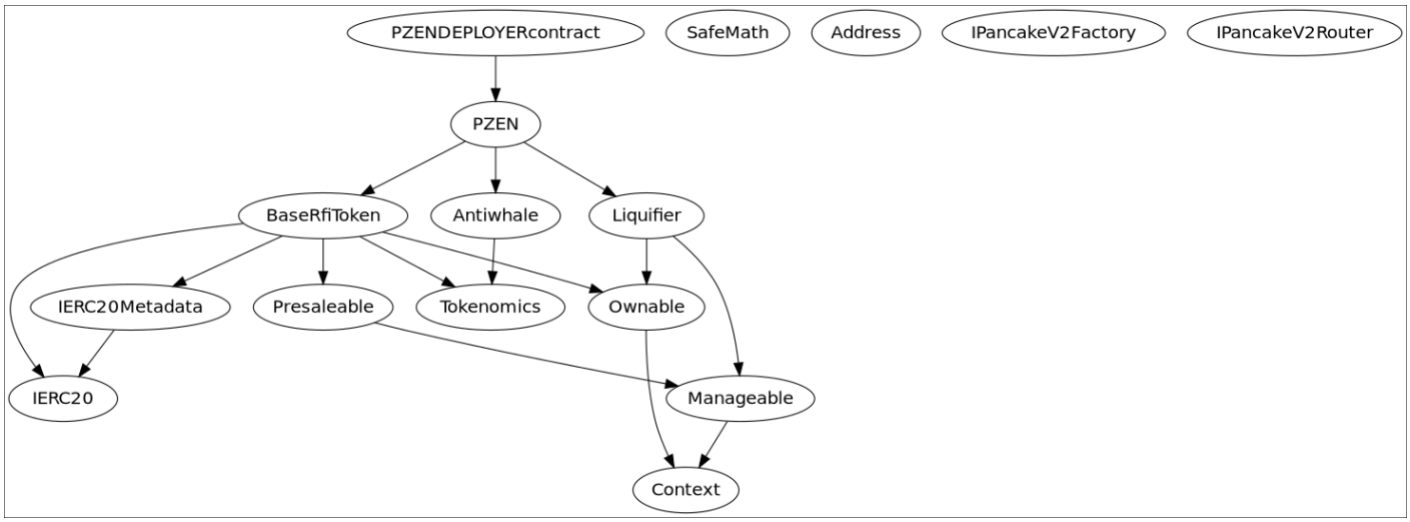
1- Check for security



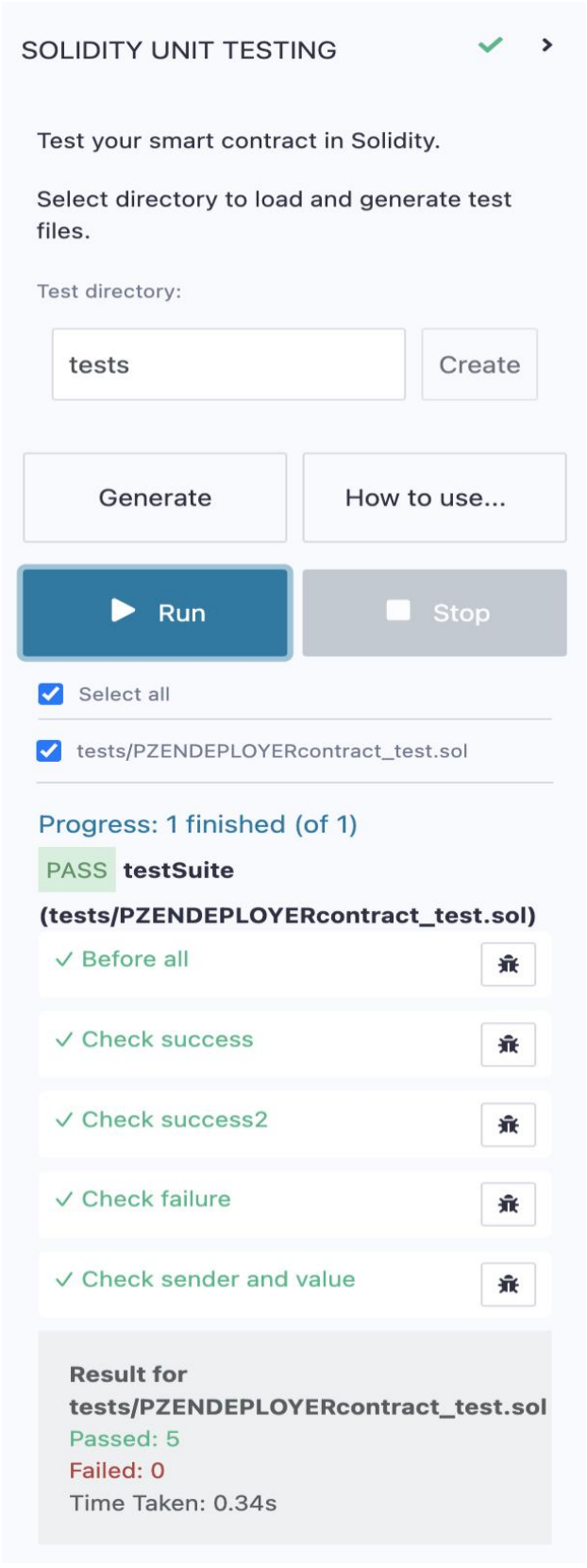
2- SOLIDITY STATIC ANALYSIS



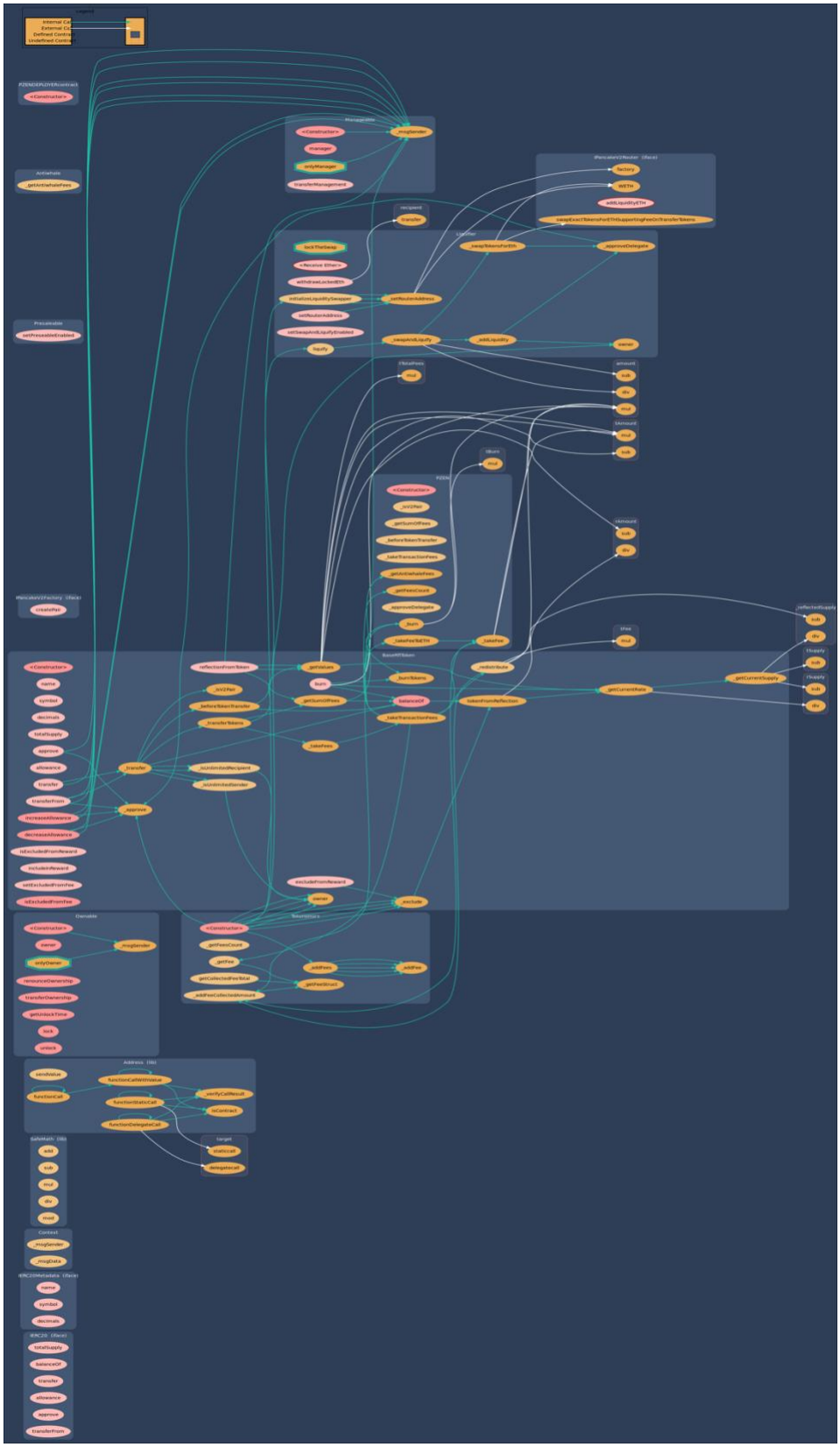
3- Inheritance graph



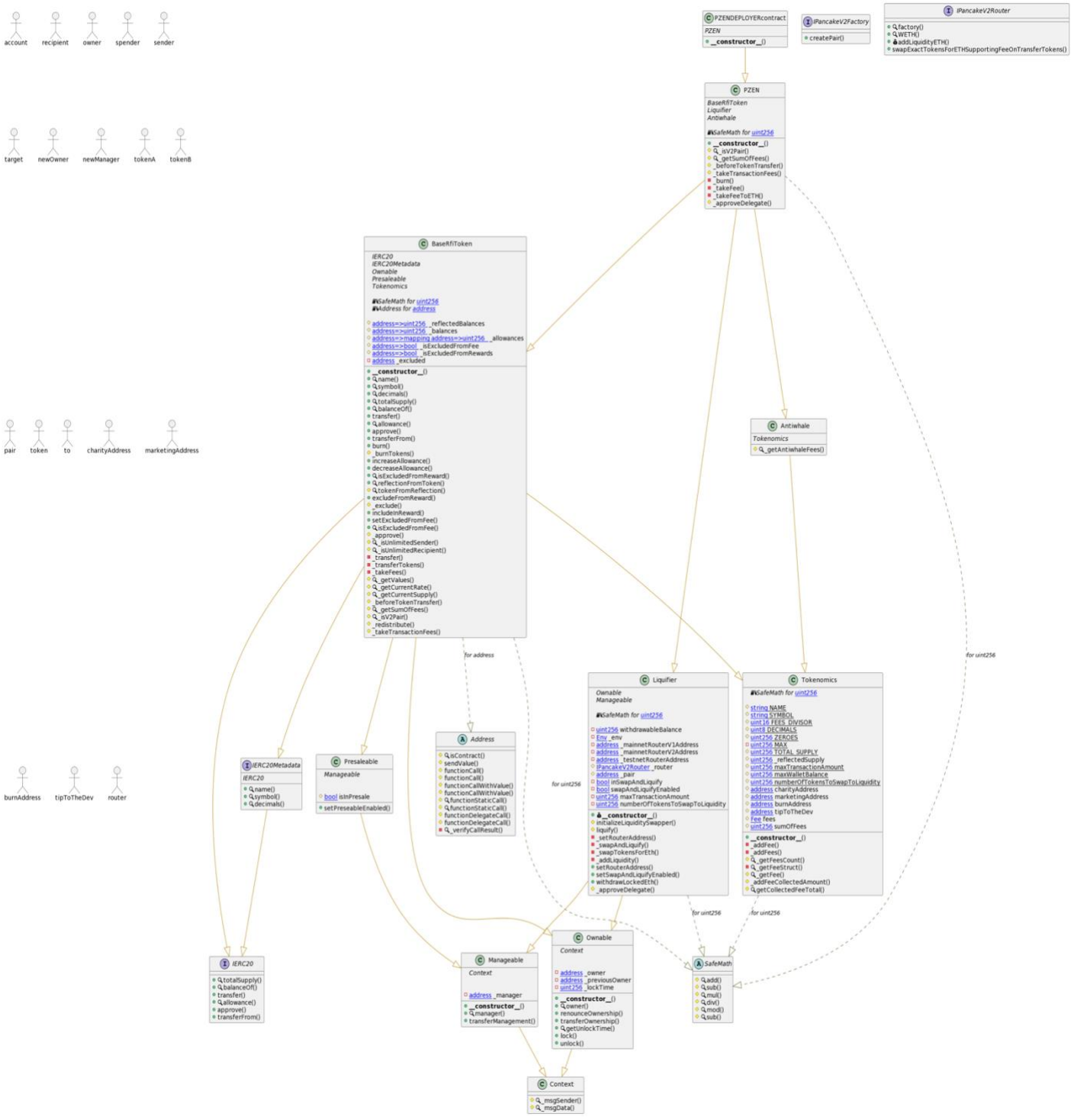
4- SOLIDITY UNIT TESTING



5- Call graph



Unified Modeling Language (UML)



Functions signature

```
Sighash   |   Function Signature
========================
16279055  =>  isContract(address)
39509351  =>  increaseAllowance(address,uint256)
18160ddd  =>  totalSupply()
70a08231  =>  balanceOf(address)
a9059cbb  =>  transfer(address,uint256)
dd62ed3e  =>  allowance(address,address)
095ea7b3  =>  approve(address,uint256)
23b872dd  =>  transferFrom(address,address,uint256)
06fdde03  =>  name()
95d89b41  =>  symbol()
313ce567  =>  decimals()
119df25f  =>  _msgSender()
8b49d47e  =>  _msgData()
771602f7  =>  add(uint256,uint256)
b67d77c5  =>  sub(uint256,uint256)
c8a4ac9c  =>  mul(uint256,uint256)
a391c15b  =>  div(uint256,uint256)
f43f523a  =>  mod(uint256,uint256)
e31bdc0a  =>  sub(uint256,uint256,string)
24a084df  =>  sendValue(address,uint256)
a0b5ffb0  =>  functionCall(address,bytes)
241b5886  =>  functionCall(address,bytes,string)
2a011594  =>  functionCallWithValue(address,bytes,uint256)
d525ab8a  =>  functionCallWithValue(address,bytes,uint256,string)
c21d36f3  =>  functionStaticCall(address,bytes)
dbc40fb9  =>  functionStaticCall(address,bytes,string)
ee33b7e2  =>  functionDelegateCall(address,bytes)
57387df0  =>  functionDelegateCall(address,bytes,string)
18c2c6a2  =>  _verifyCallResult(bool,bytes,string)
8da5cb5b  =>  owner()
715018a6  =>  renounceOwnership()
f2fde38b  =>  transferOwnership(address)
602bc62b  =>  getUnlockTime()
dd467064  =>  lock(uint256)
a69df4b5  =>  unlock()
481c6a75  =>  manager()
e4edf852  =>  transferManagement(address)
c9c65396  =>  createPair(address,address)
c45a0155  =>  factory()
ad5c4648  =>  WETH()
f305d719  =>  addLiquidityETH(address,uint256,uint256,uint256,address,uint256)
791ac947  =>
swapExactTokensForETHSupportingFeeOnTransferTokens(uint256,uint256,address[],addres
s,uint256)
346a695c  =>  _addFee(FeeType,uint256,address)
e694db42  =>  _addFees()
2ff46c73  =>  _getFeesCount()
44a297dc  =>  _getFeeStruct(uint256)
88a60fa8  =>  _getFee(uint256)
17d5a3fd  =>  _addFeeCollectedAmount(uint256,uint256)
8d11551e  =>  getCollectedFeeTotal(uint256)
74778cdc  =>  setPreseableEnabled(bool)
42966c68  =>  burn(uint256)
099bade9  =>  _burnTokens(address,uint256,uint256)
```

```
a457c2d7  =>  decreaseAllowance(address,uint256)
88f82020  =>  isExcludedFromReward(address)
4549b039  =>  reflectionFromToken(uint256,bool)
2d838119  =>  tokenFromReflection(uint256)
52390c02  =>  excludeFromReward(address)
d2480b0c  =>  _exclude(address)
3685d419  =>  includeInReward(address)
6612e66f  =>  setExcludedFromFee(address,bool)
5342acb4  =>  isExcludedFromFee(address)
104e81ff  =>  _approve(address,address,uint256)
81c4322b  =>  _isUnlimitedSender(address)
32049b2c  =>  _isUnlimitedRecipient(address)
30e0789e  =>  _transfer(address,address,uint256)
20d6115d  =>  _transferTokens(address,address,uint256,bool)
58cf1a0b  =>  _takeFees(uint256,uint256,uint256)
ddb595f3  =>  _getValues(uint256,uint256)
0ec1ce16  =>  _getCurrentRate()
97a9d560  =>  _getCurrentSupply()
d44bed71  =>  _beforeTokenTransfer(address,address,uint256,bool)
1202591c  =>  _getSumOfFees(address,uint256)
52854cce  =>  _isV2Pair(address)
cee8c7b0  =>  _redistribute(uint256,uint256,uint256,uint256)
f9d5a849  =>  _takeTransactionFees(uint256,uint256)
049498c0  =>  initializeLiquiditySwapper(Env,uint256,uint256)
dbc55a7e  =>  liquify(uint256,address)
1e4ed1de  =>  _setRouterAddress(address)
7bf54cb9  =>  _swapAndLiquify(uint256)
7f90ffe4  =>  _swapTokensForEth(uint256)
9e8af2af  =>  _addLiquidity(uint256,uint256)
41cb87fc  =>  setRouterAddress(address)
c49b9a80  =>  setSwapAndLiquifyEnabled(bool)
b7671a0d  =>  withdrawLockedEth(address)
238c12a9  =>  _approveDelegate(address,address,uint256)
577ca49c  =>  _getAntiwhaleFees(uint256,uint256)
d05e5e34  =>  _burn(uint256,uint256,uint256,uint256)
8a6bd23b  =>  _takeFee(uint256,uint256,uint256,address,uint256)
de2637aa  =>  _takeFeeToETH(uint256,uint256,uint256,address,uint256)
```

Automatic general report

Files Description Table

File Name	SHA-1 Hash
/Users/macbook/Desktop/smart contracts/PZENDEPLOYERcontract.sol	3d6bf771b69c7e5c563630623fa6ee6e979ac572

Contracts Description Table

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
└	totalSupply	External ❗		NO❗
└	balanceOf	External ❗		NO❗
└	transfer	External ❗	🛑	NO❗
└	allowance	External ❗		NO❗
└	approve	External ❗	🛑	NO❗
└	transferFrom	External ❗	🛑	NO❗
IERC20Metadata	Interface	IERC20		
└	name	External ❗		NO❗
└	symbol	External ❗		NO❗
└	decimals	External ❗		NO❗
Context	Implementation			
└	_msgSender	Internal 🔒		
└	_msgData	Internal 🔒		
SafeMath	Library			
└	add	Internal 🔒		
└	sub	Internal 🔒		
└	mul	Internal 🔒		
└	div	Internal 🔒		
└	mod	Internal 🔒		
└	sub	Internal 🔒		
Address	Library			
└	isContract	Internal 🔒		
└	sendValue	Internal 🔒	🛑	
└	functionCall	Internal 🔒	🛑	
└	functionCall	Internal 🔒	🛑	
└	functionCallWithValue	Internal 🔒	🛑	
└	functionCallWithValue	Internal 🔒	🛑	
└	functionStaticCall	Internal 🔒		
└	functionStaticCall	Internal 🔒		
└	functionDelegateCall	Internal 🔒	🛑	
└	functionDelegateCall	Internal 🔒	🛑	
└	_verifyCallResult	Private 🔐		
Ownable	Implementation	Context		

└	<Constructor>	Public ❗	🛑	NO❗
└	owner	Public ❗		NO❗
└	renounceOwnership	Public ❗	🛑	onlyOwner
└	transferOwnership	Public ❗	🛑	onlyOwner
└	getUnlockTime	Public ❗		NO❗
└	lock	Public ❗	🛑	onlyOwner
└	unlock	Public ❗	🛑	NO❗
Manageable	Implementation	Context		
└	<Constructor>	Public ❗	🛑	NO❗
└	manager	Public ❗		NO❗
└	transferManagement	External ❗	🛑	onlyManager
IPancakeV2Factory	Interface			
└	createPair	External ❗	🛑	NO❗
IPancakeV2Router	Interface			
└	factory	External ❗		NO❗
└	WETH	External ❗		NO❗
└	addLiquidityETH	External ❗	💵	NO❗
└	swapExactTokensForETHSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
Tokenomics	Implementation			
└	<Constructor>	Public ❗	🛑	NO❗
└	_addFee	Private 🔐	🛑	
└	_addFees	Private 🔐	🛑	
└	_getFeesCount	Internal 🔒		
└	_getFeeStruct	Private 🔐		
└	_getFee	Internal 🔒		
└	_addFeeCollectedAmount	Internal 🔒	🛑	
└	getCollectedFeeTotal	Internal 🔒		
Presaleable	Implementation	Manageable		
└	setPreseableEnabled	External ❗	🛑	onlyManager
BaseRfiToken	Implementation	IERC20, IERC20Metadata, Ownable, Presaleable, Tokenomics		
└	<Constructor>	Public ❗	🛑	NO❗
└	name	External ❗		NO❗
└	symbol	External ❗		NO❗
└	decimals	External ❗		NO❗
└	totalSupply	External ❗		NO❗
└	balanceOf	Public ❗		NO❗
└	transfer	External ❗	🛑	NO❗
└	allowance	External ❗		NO❗
└	approve	External ❗	🛑	NO❗
└	transferFrom	External ❗	🛑	NO❗
└	burn	External ❗	🛑	NO❗
└	_burnTokens	Internal 🔒	🛑	
└	increaseAllowance	Public ❗	🛑	NO❗
└	decreaseAllowance	Public ❗	🛑	NO❗
└	isExcludedFromReward	External ❗		NO❗
└	reflectionFromToken	External ❗		NO❗
└	tokenFromReflection	Internal 🔒		
└	excludeFromReward	External ❗	🛑	onlyOwner
└	_exclude	Internal 🔒	🛑	
└	includeInReward	External ❗	🛑	onlyOwner
└	setExcludedFromFee	External ❗	🛑	onlyOwner

└	isExcludedFromFee	Public ❗			NO❗
└	_approve	Internal 🔒	🛑	
└	_isUnlimitedSender	Internal 🔒		
└	_isUnlimitedRecipient	Internal 🔒		
└	_transfer	Private 🔐	🛑	
└	_transferTokens	Private 🔐	🛑	
└	_takeFees	Private 🔐	🛑	
└	_getValues	Internal 🔒		
└	_getCurrentRate	Internal 🔒		
└	_getCurrentSupply	Internal 🔒		
└	_beforeTokenTransfer	Internal 🔒	🛑	
└	_getSumOfFees	Internal 🔒		
└	_isV2Pair	Internal 🔒		
└	_redistribute	Internal 🔒	🛑	
└	_takeTransactionFees	Internal 🔒	🛑	
Liquifier	Implementation	Ownable, Manageable		
└	<Receive Ether>	External ❗	💵	NO❗
└	initializeLiquiditySwapper	Internal 🔒	🛑	
└	liquify	Internal 🔒	🛑	
└	_setRouterAddress	Private 🔐	🛑	
└	_swapAndLiquify	Private 🔐	🛑	lockTheSwap
└	_swapTokensForEth	Private 🔐	🛑	
└	_addLiquidity	Private 🔐	🛑	
└	setRouterAddress	External ❗	🛑	onlyManager
└	setSwapAndLiquifyEnabled	External ❗	🛑	onlyManager
└	withdrawLockedEth	External ❗	🛑	onlyManager
└	_approveDelegate	Internal 🔒	🛑	
Antiwhale	Implementation	Tokenomics		
└	_getAntiwhaleFees	Internal 🔒		
PZEN	Implementation	BaseRfiToken, Liquifier, Antiwhale		
└	<Constructor>	Public ❗	🛑	NO❗
└	_isV2Pair	Internal 🔒		
└	_getSumOfFees	Internal 🔒		
└	_beforeTokenTransfer	Internal 🔒	🛑	
└	_takeTransactionFees	Internal 🔒	🛑	
└	_burn	Private 🔐	🛑	
└	_takeFee	Private 🔐	🛑	
└	_takeFeeToETH	Private 🔐	🛑	
└	_approveDelegate	Internal 🔒	🛑	
PZENDEPLOYERcontract	Implementation	PZEN		
└	<Constructor>	Public ❗	🛑	PZEN

Legend

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

Conclusion

The contracts are written systematically. Team found no critical issues. So, it is good to go for production.

Since possible test cases can be unlimited and developer level documentation (code flow diagram with function level description) not provided, for such an extensive smart contract protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan Everything.

Security state of the reviewed contract is “Secured”.

✓ No mint function.
✓ No volatile code.
✓ No high severity issues were found.

Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice as of the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against the team on the basis of what it says or doesn't say, or how team produced it, and it is important for you to conduct your own independent investigations before making any decisions. team go into more detail on this in the below disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis, and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Saferico and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (Saferico s) owe no duty of care towards you or any other person, nor does Saferico make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and Saferico hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Saferico hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Saferico, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.