Cyberscope

Audit Report
INBUVE

April 2025

Network	ARBITRUM
Address	0xe9b8490329aac8a511f5bd9f4cdcb1b2a1964c19
Audited by	© cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	TSI	Tokens Sufficiency Insurance	Unresolved
●	OCTD	Transfers Contract's Tokens	Unresolved
●	UTH	Undifferentiated Token Handling	Unresolved
●	RVA	Reusable Vested Amount	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	MC	Missing Check	Unresolved
●	MTEE	Missing Transfer Event Emission	Unresolved
●	NWES	Nonconformity with ERC-20 Standard	Unresolved
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	RF	Redundant Functionality	Unresolved
●	ZD	Zero Division	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

Review

Contract Name	IBVToken
Compiler Version	v0.8.26+commit.8a97fa7a
Optimization	200 runs
Explorer	https://arbiscan.io/address/0xe9b8490329aac8a511f5bd9f4cdcb1b2a1964c19
Address	0xe9b8490329aac8a511f5bd9f4cdcb1b2a1964c19
Network	ARBITRUM
Symbol	IBV
Decimals	18
Total Supply	1.000.000.000
Badge Eligibility	Must Fix Criticals

Audit Updates

Initial Audit	29 Apr 2025

Source Files

Filename	SHA256
IBVToken.sol	c9acf947522eb9bd23ee98b3f816745445081d4660e7bd03bbba6b1cd9395bd7

Overview

IBVToken Contract

The `IBVToken` contract is an implementation of an ERC20 token with added functionality for staking, vesting, and tax management. It integrates several advanced features, including flexible vesting schedules, staking plans with varying durations and reward rates, and a tax distribution mechanism. This contract enables the creation and management of token vesting schedules for beneficiaries, facilitates staking for rewards, and supports tax deductions on transfers, while also allowing specific addresses to be exempt from these taxes.

Staking Functionality

The `IBVToken` contract provides a staking system, allowing users to stake tokens into different plans with varying durations and reward rates. Stakers can choose from pre-defined staking plans, each offering different annualized reward rates. The contract ensures that users cannot stake below a minimum threshold (`MIN_STAKE_AMOUNT`), and the staked tokens are locked until the end of the staking period. Once the staking period ends, users can unstake their tokens along with accrued rewards. This incentivizes long-term token holding and participation.

Vesting Schedule Functionality

The contract allows the owner to create and manage vesting schedules for beneficiaries. Vesting schedules can be set up with custom durations, intervals, and amounts, ensuring gradual token release. The contract enforces that tokens are not released until they have fully vested, providing a transparent and secure method for distributing tokens over time.

Tax Transfer Mechanism

The `IBVToken` contract includes a tax transfer mechanism, allowing the contract owner to set up tax rates and specify addresses that should receive a portion of the transfer amount. The tax rate is configurable, and the total tax sum cannot exceed 3%. This feature ensures that a percentage of each transfer can be redistributed to designated addresses, such as a treasury or specific participants.

Token Burning Functionality

The `IBVToken` contract allows for the burning of tokens, reducing the total supply. This function is useful for projects aiming to manage inflation or decrease the circulating supply over time. Tokens can be burned by the owner or users themselves, and the total supply decreases accordingly.

Pausable Contract

The contract includes a pause functionality, which allows the owner to stop all critical functions such as staking, transfers, and vesting in the event of an emergency. This provides an additional layer of security and control over the token's operations.

Roles

IBVToken Contract

Owner

Owner can interact with the following functions:

- `function addStakingPlan(uint256 duration, uint256 rewardRate)`
- `function addVestingSchedule(address beneficiary, uint256 totalAmount, uint256 startTime, uint256 duration, uint256 interval)`
- `function pause()`
- `function unpause()`
- `function addAddressIgnoreTax(address _addressIgnoreTax)`
- `function removeAddressIgnoreTax(address _addressIgnoreTax)`
- `function setTaxTransfer(uint256[] memory _percentsTaxTransfer, address[] memory _addressesTaxTransfer)`

Users

The following functions can be interacted with by regular users:

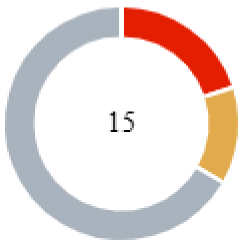
- BEP Functions
- `function stake(uint256 amount, uint256 planId)`
- `function unstake(uint256 positionId)`
- `function releaseVestedTokens()`
- `function getActiveStakingPositions(address user)`

Retrieval Functions

The following functions can be used to retrieve relevant information:

- BEP Functions
- `function releasableAmount(address beneficiary)`
- `function getPercentsTaxTransfer()`
- `function getAddressesTaxTransfer()`
- `function getOwner()`
- `function calculateRewards(address user, uint256 positionId)`
- `function getActiveStakingPositions(address user)`

Findings Breakdown



●	Critical	3
●	Medium	2
●	Minor / Informative	10

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	3	0	0	0
● Medium	2	0	0	0
● Minor / Informative	10	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	IBVToken.sol#L153,189,300,310
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users excluding the owner. The owner may take advantage of it by setting `_paused` to `true`. As a result, the contract may operate as a honeypot.

```
function pause() public onlyOwner {
    _paused = true;
}
modifier whenNotPaused() {
    require(!_paused, "Contract is paused");
    _;
}
function transfer(address recipient, uint256 amount) public
override whenNotPaused returns (bool)
function transferFrom(address sender, address recipient,
uint256 amount) public override whenNotPaused returns (bool)
```

Recommendation

The team should carefully manage the private keys of the owner’s account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

TSI - Tokens Sufficiency Insurance

Criticality	Critical
Location	IBVToken.sol#L218,266
Status	Unresolved

Description

The contract does not ensure that users will receive their tokens. As mentioned in details in sections `RVA` and `UTH` vesting schedules are created based on the current balance of the contract. However, this check does not account for amounts that are already vested. Additionally the function does not account for the tokens staked. This means that a vesting schedule will be created but when the tokens should be released, the contract may not hold enough to cover the entire releasable amount. This will result in an error and users will not be able to receive their tokens.

```
function addVestingSchedule( address beneficiary, uint256
totalAmount, uint256 startTime, uint256 duration, uint256
interval) public onlyOwner {
    //...
    require(totalAmount <= _balances[address(this)],
"Insufficient contract balance");
    //...
}
function releaseVestedTokens() public {
    VestingSchedule storage schedule =
vestingSchedules[_msgSender()];
    require(schedule.totalAmount > 0, "No vesting schedule");
    uint256 unreleased = releasableAmount(_msgSender());
    require(unreleased > 0, "No tokens to release");
    schedule.releasedAmount += unreleased;
    _transfer(address(this), _msgSender(), unreleased);
}
```

Additionally, the users can stake their tokens in the contract to earn rewards. After the duration of the chosen plan they should be able to use the `unstake` function to retrieve the tokens they staked plus the reward. Due to the contracts current configuration there is no ensurance that the contract will be able to cover the amount staked moreover the reward. This is because there is no distinction between staked, vested and reward tokens. The unstake function uses the `_transfer` method to transfer the tokens to the user which will result in an error if the scenario mentioned above is true. This means that users will not be able to get their tokens.

```
function unstake(uint256 positionId) public whenNotPaused {
    require(positionId < stakingPositions[_msgSender()].length,
"Invalid position");
    StakingPosition storage position =
stakingPositions[_msgSender()][positionId];
    require(position.isActive, "Position already unstaked");
    StakingPlan memory plan = stakingPlans[position.planId];
    require(block.timestamp >= position.startTime +
plan.duration, "Staking not ended");
    uint256 reward = calculateRewards(_msgSender(),
positionId);
    uint256 totalAmount = position.amount + reward;
    position.isActive = false;
    _transfer(address(this), _msgSender(), totalAmount);
    emit Unstaked(_msgSender(), position.amount, reward,
positionId);
}
```

Recommendation

It is recommended to consider implementing a more decentralized, automated and secure approach for handling the contract tokens. One possible solution is to create states that keep track of the vested and staked tokens separately while also saving a portion of the tokens for distribution and rewards. The contract should additionally ensure that users will receive the tokens they staked without errors.

OCTD - Transfers Contract's Tokens

Criticality	Critical
Location	IBVToken.sol#L218
Status	Unresolved

Description

The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by creating a vesting schedule for all the tokens staked and vested and release time equal to the present time. Then by calling the `releaseVestedTokens` function they can claim all the tokens. Additionally, the owner is able to use `addStakingPlan` to add a staking plan with excessive reward rates and small staking duration. This could allow them to claim the contract's tokens through the `unstake` method.

```
function releaseVestedTokens() public {
    VestingSchedule storage schedule =
vestingSchedules[_msgSender()];
    require(schedule.totalAmount > 0, "No vesting schedule");

    uint256 unreleased = releasableAmount(_msgSender());
    require(unreleased > 0, "No tokens to release");

    schedule.releasedAmount += unreleased;
    _transfer(address(this), _msgSender(), unreleased);
}
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

UTH - Undifferentiated Token Handling

Criticality	Medium
Location	IBVToken.sol#L197,266,284
Status	Unresolved

Description

The contract does not separate the balances of tokens staked, vested and tokens used for rewards. This results in staked tokens being vested and vested tokens being used as rewards. Additionally, due to the lack of token differentiation the same tokens can be vested multiple times as mentioned in the `RVA` section.

```
function addVestingSchedule( address beneficiary, uint256
totalAmount, uint256 startTime, uint256 duration, uint256
interval) public onlyOwner {
    //...
    require(totalAmount <= _balances[address(this)],
"Insufficient contract balance");
    //...
}

function unstake(uint256 positionId) public whenNotPaused {
    //...
    uint256 reward = calculateRewards(_msgSender(),
positionId);
    uint256 totalAmount = position.amount + reward;
    _transfer(address(this), _msgSender(), totalAmount);
    //...
}

function calculateRewards(address user, uint256 positionId)
public view returns (uint256) {
    require(positionId < stakingPositions[user].length,
"Invalid position");

    StakingPosition memory position =
stakingPositions[user][positionId];
    if (!position.isActive) return 0;

    StakingPlan memory plan = stakingPlans[position.planId];
    uint256 stakingDuration = block.timestamp -
position.startTime;

    if (stakingDuration > plan.duration) {
        stakingDuration = plan.duration;
    }

    return (position.amount * plan.rewardRate *
stakingDuration) / (SECONDS_PER_YEAR * 100);
}
```

Recommendation

It is recommended that the tokens used for vesting, staking, and rewards are stored in state separately. This will ensure that there are always enough tokens to cover all obligations, prevent accounting errors, and make the contract's logic easier to audit and maintain.

RVA - Reusable Vested Amount

Criticality	Medium
Location	IBVToken.sol#L205
Status	Unresolved

Description

`addVestingSchedule` does not account for the already vested tokens. Instead it just checks the contract's entire balance. This can result in multiple vesting schedules created, vesting the same tokens. If the tokens in the contract are not sufficient the vested tokens will not be able to be claimed.

```
    function addVestingSchedule( address beneficiary, uint256
totalAmount, uint256 startTime, uint256 duration, uint256
interval) public onlyOwner {
        //...
        require(totalAmount <= _balances[address(this)],
"Insufficient contract balance");
        //...
    }
```

Recommendation

The contract should check for the already vested amount and only allow the vesting of not already vested tokens. This could be achieved by keeping track of the amounts vested and separating them from the rest of the contract's balance.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	IBVToken.sol#L79,96,101,189,193,197,439
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

```
function setTaxTransfer(uint256[] memory _percentsTaxTransfer,
address[] memory _addressesTaxTransfer) public onlyOwner
function addAddressIgnoreTax(address _addressIgnoreTax) public
onlyOwner
function removeAddressIgnoreTax(address _addressIgnoreTax)
public onlyOwner
```

```
function pause() public onlyOwner
function unpause() public onlyOwner
function addVestingSchedule( address beneficiary, uint256
totalAmount, uint256 startTime, uint256 duration, uint256
interval) public onlyOwner
function addStakingPlan(uint256 duration, uint256 rewardRate)
public onlyOwner
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

MC - Missing Check

Criticality	Minor / Informative
Location	IBVToken.sol#L79,91,96,184,197
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

`setTaxTransfer` should check if each entry on `_percentsTaxTransfer` is zero and on `_addressesTaxTransfer` is the zero address.

```
function setTaxTransfer(uint256[] memory _percentsTaxTransfer,
address[] memory _addressesTaxTransfer) public onlyOwner
```

Both `addAddressIgnoreTax` and `removeAddressIgnoreTax` should check if the parameter added is the zero address.

```
function addAddressIgnoreTax(address _addressIgnoreTax) public
onlyOwner
function removeAddressIgnoreTax(address _addressIgnoreTax)
public onlyOwner
```

`_addStakingPlan` should check if `duration` and `rewardRate` are non-zero values.

```
function _addStakingPlan(uint256 duration, uint256 rewardRate)
private
```

`addVestingSchedule` should check if `beneficiary` is a non-zero address. `startTime` should be at least equal to the current timestamp. `duration` should be larger than `interval` and both of them should have non-zero, reasonable values.

```
function addVestingSchedule( address beneficiary, uint256
totalAmount, uint256 startTime, uint256 duration, uint256
interval) public onlyOwner
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

MTEE - Missing Transfer Event Emission

Criticality	Minor / Informative
Location	IBVToken.sol#L162,163
Status	Unresolved

Description

The contract does not emit an event when portions of the main amount are transferred during the transfer process. This lack of event emission results in decreased transparency and traceability regarding the flow of tokens, and hinders the ability of decentralized applications (dApps), such as blockchain explorers, to accurately track and analyze these transactions.

```
_balances[address(this)] = vestingAmount;
_balances[_msgSender()] = _totalSupply - vestingAmount;
```

Recommendation

It is advisable to incorporate the emission of detailed event logs following each asset transfer. These logs should encapsulate key transaction details, including the identities of the sender and receiver, and the quantity of assets transferred. Implementing this practice will enhance the reliability and transparency of transaction tracking systems, ensuring accurate data availability for ecosystem participants.

NWES - Nonconformity with ERC-20 Standard

Criticality	Minor / Informative
Location	IBVToken.sol#L335
Status	Unresolved

Description

The contract does not fully conform to the ERC20 Standard. Specifically, according to the standard, transfers of 0 values must be treated as normal transfers and fire the Transfer event. However, the contract implements a conditional check that prohibits transfers of 0 values. This discrepancy between the contract's implementation and the ERC20 standard may lead to inconsistencies and incompatibilities with other contracts.

```
function _transfer(address sender, address recipient, uint256
amount) private {
        //...
        require(amount > 0, "Transfer amount must be greater
than zero");
        //...
}
```

Recommendation

The incorrect implementation of the ERC20 standard could potentially lead to problems when interacting with the contract, as other contracts or applications that expect the ERC20 interface may not behave as expected. The team is advised to review and revise the implementation of the transfer mechanism to ensure full compliance with the ERC20 standard. https://eips.ethereum.org/EIPS/eip-20.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	IBVToken.sol#L253
Status	Unresolved

Description

The `_transfer()` function is used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

```
function stake(uint256 amount, uint256 planId) public
whenNotPaused {
    _transfer(_msgSender(), address(this), amount);
    uint256 positionId = stakingPositions[_msgSender()].length;
    stakingPositions[_msgSender()].push(StakingPosition({
        amount: amount,
        startTime: block.timestamp,
        planId: planId,
        isActive: true
    }));
    //...
}
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

```
 Actual Transferred Amount = Balance After Transfer - Balance
Before Transfer
```

RF - Redundant Functionality

Criticality	Minor / Informative
Location	IBVToken.sol#L400
Status	Unresolved

Description

The contract inherits the `Ownalble` contract that enables basic access control to basic functions by an address defined as the `owner`. `Ownable` has the `owner` function that returns the current owner. However the contract implements the `getOwner` function that has same functionality, therefore it is redundant.

```
function getOwner() external view returns (address) {
    return owner();
}
```

Recommendation

It is recommended to remove redundant functions to increase code readability and optimization.

ZD - Zero Division

Criticality	Minor / Informative
Location	IBVToken.sol#L241,242
Status	Unresolved

Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert.

```
uint256 elapsedIntervals = elapsedTime / schedule.interval;
uint256 totalIntervals = schedule.duration / schedule.interval;
```

Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero, or should not allow the execution of the corresponding statements.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	IBVToken.sol#L71,79,91,96,115,116,117
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
uint256[] memory _percentsTax
uint256[] memory _percentsTaxTransfer
address[] memory _addressesTaxTransfer
address _addressIgnoreTax
uint8 private constant _decimals = 18
string private constant _symbol = "IBV"
string private constant _name = "INBUVE"
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/stable/style-guide.html#naming-conventions.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	IBVToken.sol#L241,243
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 elapsedIntervals = elapsedTime / schedule.interval
uint256 vestedAmount = (schedule.totalAmount *
elapsedIntervals) / totalIntervals
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	IBVToken.sol#L2
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.19;
```

Recommendation

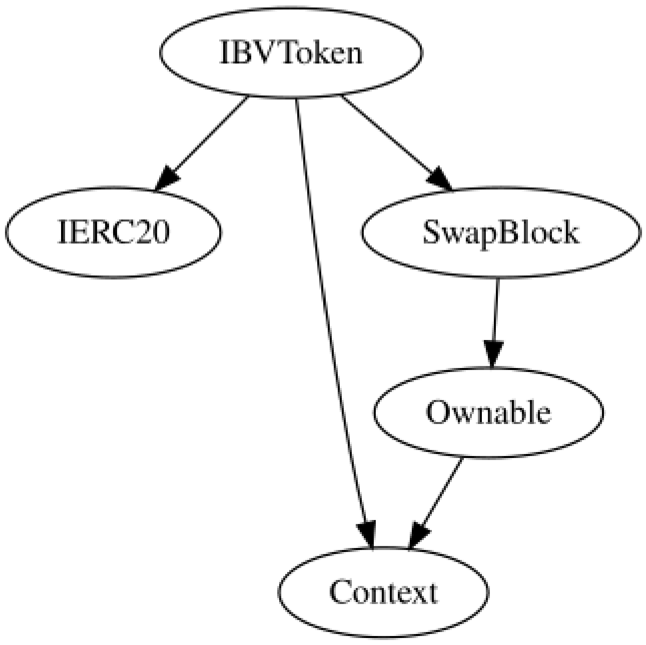
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Functions Analysis

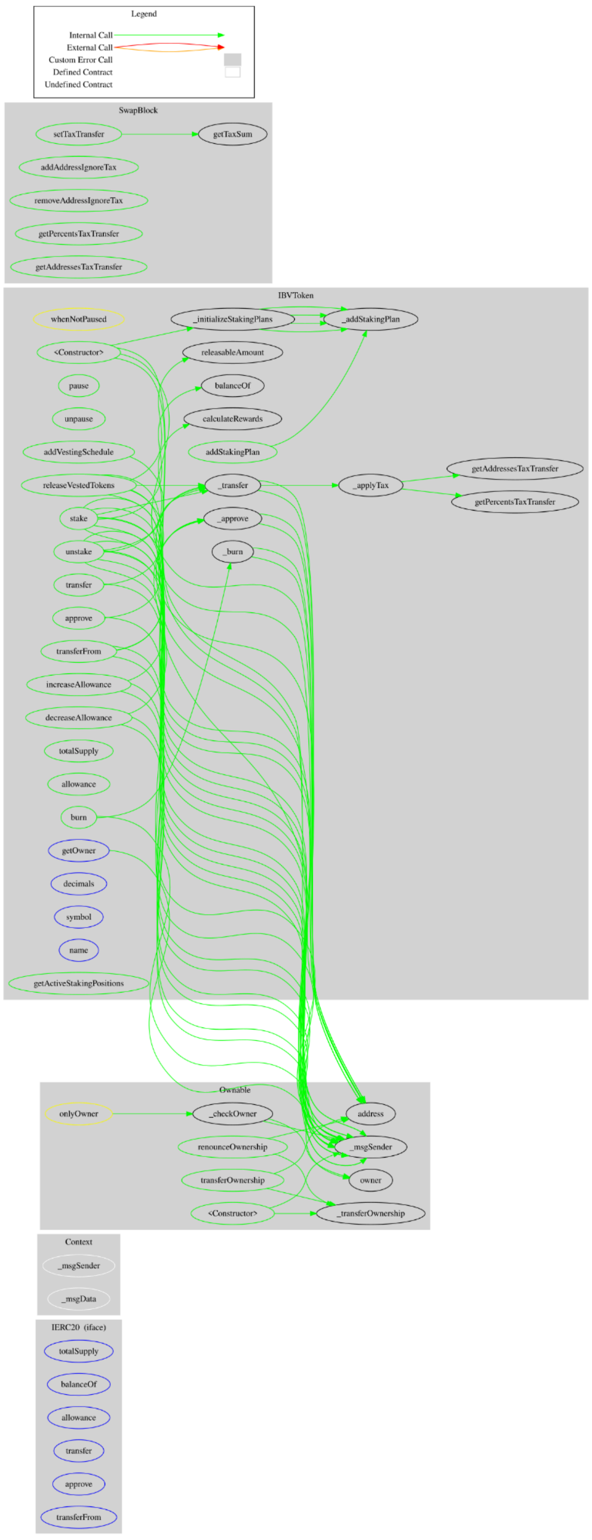
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SwapBlock	Implementation	Ownable		
	getTaxSum	Internal		
	setTaxTransfer	Public	✓	onlyOwner
	addAddressIgnoreTax	Public	✓	onlyOwner
	removeAddressIgnoreTax	Public	✓	onlyOwner
	getPercentsTaxTransfer	Public		-
	getAddressesTaxTransfer	Public		-
IBVToken	Implementation	Context, IERC20, SwapBlock		
		Public	✓	-
	_initializeStakingPlans	Private	✓	onlyOwner
	_addStakingPlan	Private	✓	
	pause	Public	✓	onlyOwner
	unpause	Public	✓	onlyOwner
	addVestingSchedule	Public	✓	onlyOwner
	releaseVestedTokens	Public	✓	-
	releasableAmount	Public		-
	stake	Public	✓	whenNotPaused
	unstake	Public	✓	whenNotPaused

	calculateRewards	Public		-
	transfer	Public	✓	whenNotPaused
	approve	Public	✓	whenNotPaused
	transferFrom	Public	✓	whenNotPaused
	increaseAllowance	Public	✓	whenNotPaused
	decreaseAllowance	Public	✓	whenNotPaused
	_transfer	Private	✓	
	_approve	Private	✓	
	_applyTax	Private	✓	
	totalSupply	Public		-
	balanceOf	Public		-
	allowance	Public		-
	burn	Public	✓	whenNotPaused
	_burn	Private	✓	
	getOwner	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getActiveStakingPositions	Public		-
	addStakingPlan	Public	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
SwapBlock
setTaxTransfer
getTaxSum
addAddressIgnoreTax
removeAddressIgnoreTax
getPercentsTaxTransfer
getAddressesTaxTransfer
IBVToken
whenNotPaused
_initializeStakingPlans
_addStakingPlan
<Constructor>
releasableAmount
pause
balanceOf
unpause
calculateRewards
addVestingSchedule
addStakingPlan
releaseVestedTokens
_transfer
_applyTax
getAddressesTaxTransfer
getPercentsTaxTransfer
stake
_approve
unstake
_burn
transfer
approve
transferFrom
increaseAllowance
decreaseAllowance
totalSupply
allowance
burn
getOwner
decimals
symbol
name
getActiveStakingPositions
Ownable
onlyOwner
_checkOwner
address
renounceOwnership
_msgSender
transferOwnership
owner
<Constructor>
_transferOwnership
Context
_msgSender
_msgData
IERC20 (iface)
totalSupply
balanceOf
allowance
transfer
approve
transferFrom

Summary

INBUVE contract implements a token, staking and vesting mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 3% fees.

Disclaimer

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

cyberscope.io