SAFUAUDIT
SMART CONTRACT AUDITING

INTERNET 3.0 AUTOSTAKING PROJECT

SMART CONTRACT AUDIT

March 24, 2022

INTRODUCTION

Client	Internet 3.0 Autostaking Project (IAS)
Language	Solidity
Contract Address	0x0268965dA4E5b73e64E9973548613Fd95a7B3872
Decimals	18
Supply	5,000,000,000
Platform	Binance Smart Chain
Compiler	v0.7.4+commit.3f05b770
Optimization	Yes, with 200 runs
Website	https://internet3.space

TABLE OF CONTENTS

01 INTRODUCTION

02 ABSTRACT

03 VULNERABILITIES TEST

04 MANUAL ANALYSIS

05 CONCLUSIONS

Approach

Audit Details

Our comprehensive audit report provides a full overview of the audited system's architecture, smart contract codebase, and details on any vulnerabilities found within the system.

Audit Goals

The audit goal is to ensure that the project is built to protect investors and users, preventing potentially catastrophic vulnerabilities after launch, that lead to scams and rugpulls.

,

Code Quality

Our analysis includes both automatic tests and manual code analysis for the following aspects:

- Exploits
- Back-doors
- Vulnerability
- Accuracy
- Readability

Tools

- Remix IDE
- MythX, Mythril
- SWC Registry
- Open Zeppelin Code Analyzer
- Solidity Code Complier

RISK CLASSIFICATION

CRITICAL

Issues on this level are critical to the smart contract's performance/functionality and should be fixed before moving to a live environment.

MEDIUM

Issues on this level could potentially bring problems and should eventually be fixed.

MINOR

Issues on this level are minor details and warning that can remain unfixed but would be better fixed at some point in the future

INFORMATIONAL

Information level is to offer suggestions for improvement of efficacity or security for features with a risk free factor.

ABSTRACT

Fees	Ownership	Mint function
Buy Fees: 13% Sell Fees: 24% *at audit time	Owned	Rebase Token

Owner can set fees	Max Tx amount	Pause
Owner can't set fees	Owner can't set max Tx amount	Owner can't pause trading

Vulnerabilities Test

SWC ID	Description	
SWC-100	Function Default Visibility	Passed
SWC-101	Integer Overflow and Underflow	Passed
SWC-102	Outdated Compiler Version	Passed
SWC-103	Floating Pragma	Minor
SWC-104	Unchecked Call Return Value	Passed
SWC-105	Unprotected Ether Withdrawal	Passed
SWC-106	Unprotected SELF-DESTRUCT Instruction	Passed
SWC-107	Re-entrancy	Passed
SWC-108	State Variable Default Visibility	Minor
SWC-109	Uninitialized Storage Pointer	Passed
SWC-110	Assert Violation	Passed
SWC-111	Use of Deprecated Solidity Functions	Passed
SWC-112	Delegate Call to Untrusted Callee	Passed
SWC-113	DoS with Failed Call	Passed
SWC-114	Transaction Order Dependence	Passed
SWC-115	Authorization through tx.origin	Passed

SWC-116	Block values as a proxy for time	**Passed**
SWC-117	Signature Malleability	**Passed**
SWC-118	Incorrect Constructor Name	**Passed**
SWC-119	Shadowing State Variables	**Passed**
SWC-120	Weak Sources of Randomness from Chain Attributes	**Passed**
SWC-121	Missing Protection against Signature Replay Attacks	**Passed**
SWC-122	Lack of Proper Signature Verification	**Passed**
SWC-123	Requirement Violation	**Passed**
SWC-124	Write to Arbitrary Storage Location	**Passed**
SWC-125	Incorrect Inheritance Order	**Passed**
SWC-126	Insufficient Gas Griefing	**Passed**
SWC-127	Arbitrary Jump with Function Type Variable	**Passed**
SWC-128	DoS With Block Gas Limit	**Passed**
SWC-129	Typographical Error	**Passed**
SWC-130	Right-To-Left-Override control character (U+202E)	**Passed**
SWC-131	Presence of unused variables	**Passed**
SWC-132	Unexpected Ether balance	**Passed**
SWC-133	Hash Collisions With Multiple Variable Length Arguments	**Passed**
SWC-134	Message call with the hardcoded gas amount	**Passed**
SWC-135	Code With No Effects (Irrelevant/Dead Code)	**Passed**
SWC-136	Unencrypted Private Data On-Chain	**Passed**

MANUAL ANALYSIS

The contract is verified to check if functions do and work as they should and malicious code is not inserted.

	Tested	Result
Transfer	Yes	**Passed**
Total Supply	Yes	**Passed**
Buy Back	Yes	**N/A**
Burn	Yes	**N/A**
Mint	Yes	**N/A**
Rebase	Yes	**Medium**
Pause	Yes	**N/A**
Blacklist	Yes	**N/A**
Lock	Yes	**N/A**
Max Transaction	Yes	**N/A**
Transfer Ownership	Yes	**Passed**
Renounce Ownership	Yes	**Passed**

CONTRACT INSPECTION

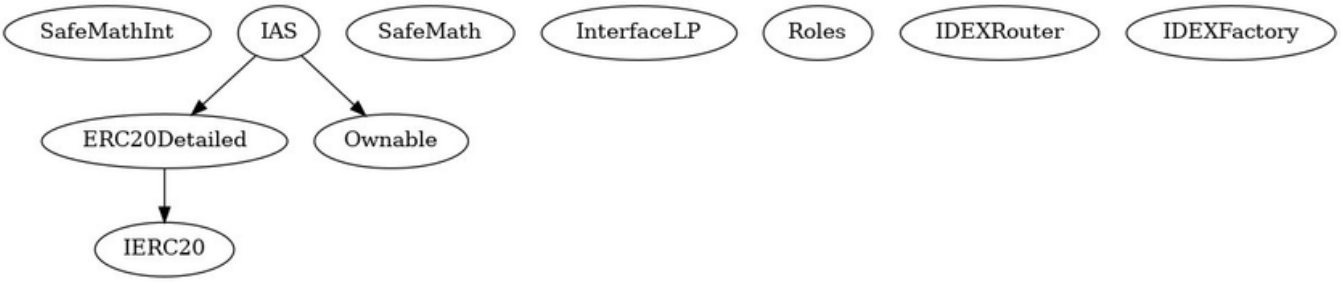
SafeMathInt	Library			
└	mul	Internal 🔒		
└	div	Internal 🔒		
└	sub	Internal 🔒		
└	add	Internal 🔒		
└	abs	Internal 🔒		
IERC20	Interface			
└	totalSupply	External ❗		NO❗
└	balanceOf	External ❗		NO❗
└	allowance	External ❗		NO❗
└	transfer	External ❗	🛑	NO❗
└	approve	External ❗	🛑	NO❗
└	transferFrom	External ❗	🛑	NO❗
SafeMath	Library			
└	add	Internal 🔒		
└	sub	Internal 🔒		
└	sub	Internal 🔒		
└	mul	Internal 🔒		
└	div	Internal 🔒		
└	div	Internal 🔒		
└	mod	Internal 🔒		
InterfaceLP	Interface			
└	sync	External ❗	🛑	NO❗
Roles	Library			
└	add	Internal 🔒	🛑	
└	remove	Internal 🔒	🛑	
└	has	Internal 🔒		
ERC20Detailed	Implementation	IERC20		
└	<Constructor>	Public ❗	🛑	NO❗
└	name	Public ❗		NO❗
└	symbol	Public ❗		NO❗
└	decimals	Public ❗		NO❗

IDEXRouter	Interface			
└	factory	External ❗		|NO❗
└	WETH	External ❗		|NO❗
└	addLiquidity	External ❗	🛑	|NO❗
└	addLiquidityETH	External ❗	💸	|NO❗
└	swapExactTokensForTokensSupportingFeeOnTransferTokens	External ❗	🛑	|NO❗
└	swapExactETHForTokensSupportingFeeOnTransferTokens	External ❗	💸	|NO❗
└	swapExactTokensForETHSupportingFeeOnTransferTokens	External ❗	🛑	|NO❗
IDEXFactory	Interface			
└	createPair	External ❗	🛑	|NO❗
Ownable	Implementation			
└	<Constructor>	Public ❗	🛑	|NO❗
└	owner	Public ❗		|NO❗
└	renounceOwnership	Public ❗	🛑	onlyOwner
└	transferOwnership	Public ❗	🛑	onlyOwner
└	_transferOwnership	Internal 🔒	🛑		
IAS	Implementation	ERC20Detailed, Ownable		
└	<Constructor>	Public ❗	🛑	ERC20Detailed
└	<Receive Ether>	External ❗	💸	|NO❗
└	totalSupply	External ❗		|NO❗
└	allowance	External ❗		|NO❗
└	balanceOf	Public ❗		|NO❗
└	checkFeeExempt	External ❗		|NO❗
└	checkSwapThreshold	External ❗		|NO❗
└	shouldRebase	Internal 🔒			
└	shouldTakeFee	Internal 🔒			
└	shouldSwapBack	Public ❗		|NO❗
└	getCirculatingSupply	Public ❗		|NO❗
└	getLiquidityBacking	Public ❗		|NO❗
└	isOverLiquified	Public ❗		|NO❗
└	manualSync	Public ❗	🛑	|NO❗
└	transfer	External ❗	🛑	validRecipient
└	_basicTransfer	Internal 🔒	🛑		
└	_transferFrom	Internal 🔒	🛑		
└	transferFrom	External ❗	🛑	validRecipient
└	_swapAndLiquify	Private 🔐	🛑		

└	_addLiquidity	Private 🔐	🛑	
└	_addLiquidityBusd	Private 🔐	🛑	
└	_swapTokensForBNB	Private 🔐	🛑	
└	_swapTokensForBusd	Private 🔐	🛑	
└	swapBack	Internal 🔒	🛑	swapping
└	takeFee	Internal 🔒	🛑	
└	decreaseAllowance	External ❗	🛑	NO❗
└	increaseAllowance	External ❗	🛑	NO❗
└	approve	External ❗	🛑	NO❗
└	_rebase	Private 🔐	🛑	
└	coreRebase	Private 🔐	🛑	
└	manualRebase	External ❗	🛑	onlyOwner
└	setAutomatedMarketMakerPair	Public ❗	🛑	onlyOwner
└	setFeeExempt	External ❗	🛑	onlyOwner
└	setTargetLiquidity	External ❗	🛑	onlyOwner
└	setSwapBackSettings	External ❗	🛑	onlyOwner
└	setFeeReceivers	External ❗	🛑	onlyOwner
└	clearStuckBalance	External ❗	🛑	onlyOwner
└	rescueToken	External ❗	🛑	onlyOwner
└	setAutoRebase	External ❗	🛑	onlyOwner
└	setRebaseFrequency	External ❗	🛑	onlyOwner
└	setRewardYield	External ❗	🛑	onlyOwner
└	setFeesOnNormalTransfers	External ❗	🛑	onlyOwner
└	setIsLiquidityInBnb	External ❗	🛑	onlyOwner
└	setNextRebase	External ❗	🛑	onlyOwner

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable
🔐	Private function
🔒	Internal function
NO❗	Function has no modifier

INHERITANCE TREE



Inheritance is a feature of the object-oriented programming language. It is a way of extending the functionality of a program, used to separate the code, reduces the dependency, and increases the re-usability of the existing code. Solidity supports inheritance between smart contracts, where multiple contracts can be inherited into a single contract.

Important Snippets

Owner can trigger Manual Rebase

```
function manualRebase() external onlyOwner{
    require(!inSwap, "Try again");
    require(nextRebase <= block.timestamp, "Not in time");

    uint256 circulatingSupply = getCirculatingSupply();
    int256 supplyDelta = int256(circulatingSupply.mul(rewardYield).div(rewardYieldDenominator));

    coreRebase(supplyDelta);
    manualSync();
}
```

Owner can set rebase frequency

```
function setRebaseFrequency(uint256 _rebaseFrequency) external onlyOwner {
    require(_rebaseFrequency <= MAX_REBASE_FREQUENCY, "Too high");
    rebaseFrequency = _rebaseFrequency;
}
function setNextRebase(uint256 _nextRebase) external onlyOwner {
    nextRebase = _nextRebase;
}
```

Owner can set reward Yield

```
function setRewardYield(uint256 _rewardYield, uint256 _rewardYieldDenominator) external onlyOwner {
    rewardYield = _rewardYield;
    rewardYieldDenominator = _rewardYieldDenominator;
}
```

GOOD PRACTICES

- The owner cannot set max Tx amount
- The owner cannot stop or pause the smart contract
- The owner cannot set fees
- The smart contract utilizes "SafeMath" to prevent overflows

```
library SafeMath {
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    function sub(
        uint256 a,
        uint256 b,
        string memory errorMessage
    ) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }

    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    function div(
        uint256 a,
        uint256 b,
        string memory errorMessage
    ) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        uint256 c = a / b;

        return c;
    }

    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0);
        return a % b;
    }
}
```

DISCLAIMER

SafuAudit.com is not a financial institution and the information provided on this website does not constitute investment advice, financial advice, trading advice or any other sort of advice. You should not treat any of the website's content as such. Investing in crypto assets carries a high level of risk and does not hold guarantees for not sustaining financial loss due to their volatility.

Accuracy of Information

SafuAudit will strive to ensure accuracy of information listed on this website although it will not hold any responsibility for any missing or wrong information. SafuAudit provides all information as is. You understand that you are using any and all information available here at your own risk. Any use or reliance on our content and services is solely at your own risk and discretion.

The purpose of the audit is to analyse the on-chain smart contract source code, and to provide basic overview of the project.

While we have used all the information available to us for this straightforward investigation, you should not rely on this report only — we recommend proceeding with several independent audits Be aware that smart contracts deployed on a blockchain aren't secured enough against external vulnerability, or a hack. Be aware that active smart contract owner privileges constitute an elevated impact to smart contract's safety and security. Therefore, SafuAudit does not guarantee the explicit security of the audited smart contract. 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.

AUDIT RESULTS

CRITICAL

No critical severity issues have been found.

MEDIUM

- Rebase functions (automatic and manual) can be adjusted according to some parameters that have arbitrary limits set.

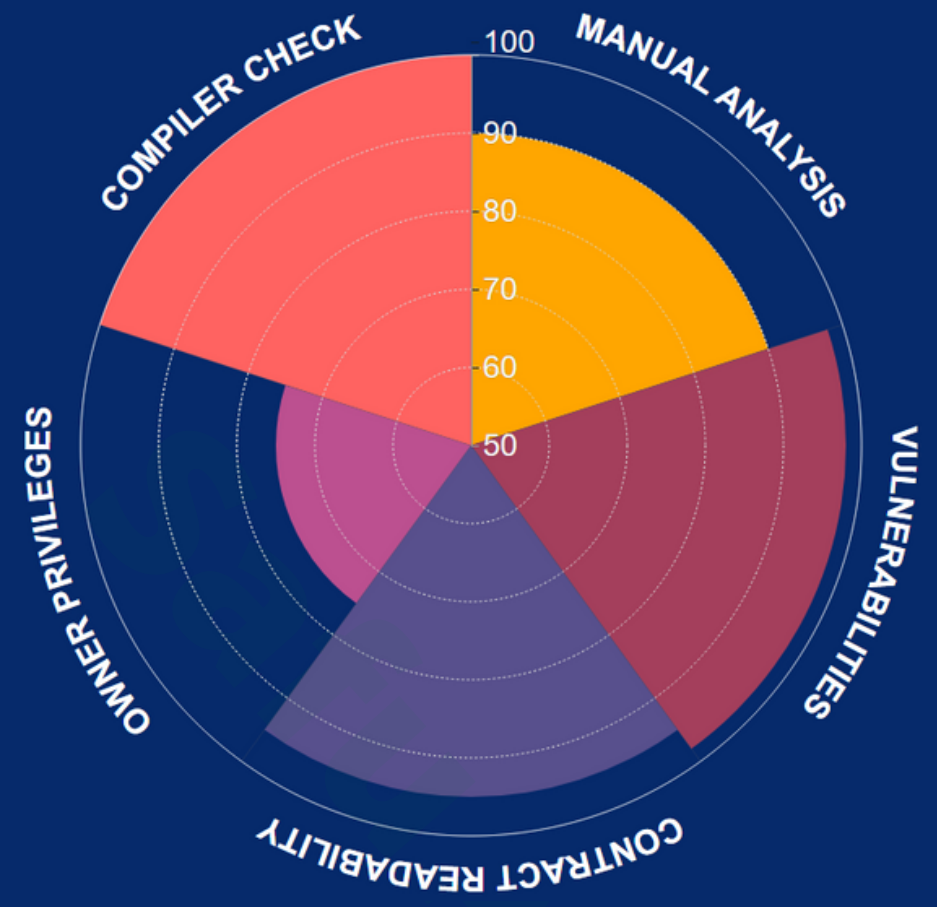
MINOR

- A floating pragma is set. It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.
- State variable visibility is not set. It is best practice to set the visibility of state variables explicitly. The default visibility for "_isFeeExempt, DEAD, ZERO, targetLiquidity, targetLiquidityDenominator, inSwap" is internal.

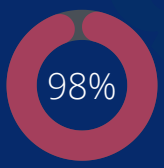
INFORMATIONAL

The standard audit model does not offer suggestions and consulting for improvements of efficacity.

SCORE

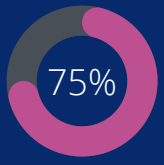


Manual Analysis



Vulnerabilities

Contract Readability



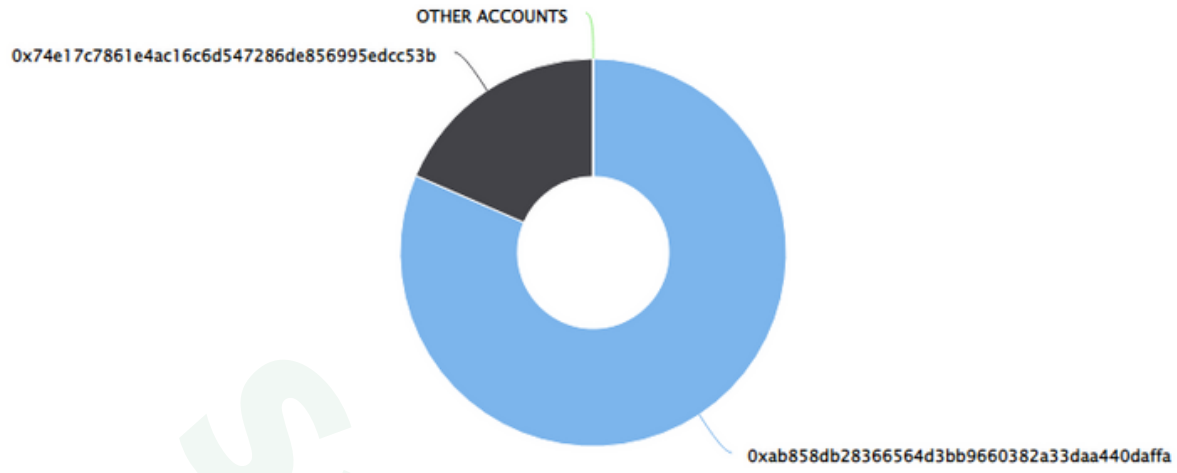
Owner Privileges

Compiler Check

Final Score:92.6

SUMMARY

Top 10 holders



Rank	Address	Quantity (Token)	Percentage
1	0xab858db28366564d3bb9660382a33daa440daffa	4,075,000,000	81.5000%
2	0x74e17c7861e4ac16c6d547286de856995edcc53b	925,000,000	18.5000%

Conclusion

Project Internet 3.0 Autostaking Project does not contain any severe issues. It utilizes Rebase function that increases or decreases total supply and can be adjusted according to some parameters that have arbitrary limits set.

SafuAudit has tested the security based on manual and automated tests. Please note that we don't offer any warranties for business model.

SafuAudit.com