Cyberscope

Audit Report

IMAL

July 2023

Network	BSC
Address	0xff43312e1a2d09033963ca594e722f3d29dc2f5b
Audited by	© cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	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
●	RC	Redundant Code	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	DEE	Duplicate Event Emission	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L08	Tautology or Contradiction	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Explorer	https://bscscan.com/address/0xff43312e1a2d09033963ca594e722f3d29dc2f5b

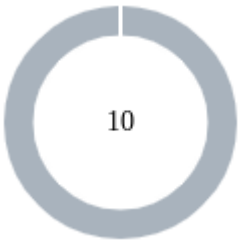
Audit Updates

Initial Audit	20 Jul 2023 https://github.com/cyberscope-io/audits/blob/main/imal/v1/audit.pdf
Corrected Phase 2	24 Jul 2023

Source Files

Filename	SHA256
Imal.sol	5dcb7bf72045800eb355b42096a57a59dcb45e50ea70edad73d7d2da33592ba4

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	10

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

RC - Redundant Code

Criticality	Minor / Informative
Location	Imal.sol#L324
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

Specifically, the variable `bpEnabled` is initialized to `false` and is never modified elsewhere in the contract. This means that the condition `bpEnabled && !BPDisabledForever` will always evaluate to `false`, rendering the code inside the if statement redundant. As a result, the call to `BP.protect(sender, recipient, amount)` will never be executed.

```
    bool public bpEnabled;

        if (bpEnabled && !BPDisabledForever) {
            BP.protect(sender, recipient, amount);
         }
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. If the `bpEnabled` variable is not intended to be modified in the future, the entire if statement can be safely removed to optimize the contract's bytecode size and execution efficiency. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	Imal.sol#L287,304
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
    function setBotProtectionDisableForever() external onlyOwner {
        require(BPDisabledForever == false);
        BPDisabledForever = true;
    }

    function excludeFromFee(address address_, bool isExcluded) external
onlyOwner {
        isExcludedFromFee[address_] = isExcluded;
    }
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	Imal.sol#L299,304
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract update the state of excluded addresses even if their current state is the same as the the one passed as an argument. As a result, the contract performs redundant storage writes.

```
    function setBeneficiaryAddress(address beneficiaryAddress_) external
        onlyOwner {
        beneficiaryAddress = beneficiaryAddress_;
        emit SetBeneficiaryAddress(beneficiaryAddress);
    }

    function excludeFromFee(address address_, bool isExcluded) external
onlyOwner {
        isExcludedFromFee[address_] = isExcluded;
    }
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

DEE - Duplicate Event Emission

Criticality	Minor / Informative
Location	Imal.sol#L348
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract could emits the two events with the same data. As a result event duplication is created.

```
emit TransferFee(sender, beneficiaryAddress, feeAmount);
emit Transfer(sender, beneficiaryAddress, feeAmount);
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Imal.sol#L212,213
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
BPContract public BP
bool public bpEnabled
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Imal.sol#L212,214
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.

```
BPContract public BP
bool public BPDisabledForever = false
```

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/v0.8.17/style-guide.html#naming-convention.

L08 - Tautology or Contradiction

Criticality	Minor / Informative
Location	Imal.sol#L290
Status	Unresolved

Description

A tautology is a logical statement that is always true, regardless of the values of its variables. A contradiction is a logical statement that is always false, regardless of the values of its variables.

Using tautologies or contradictions can lead to unintended behavior and can make the code harder to understand and maintain. It is generally considered good practice to avoid tautologies and contradictions in the code.

```
require(feePercentage_ >= 0, "Imal: transaction fee percentage equals
0")
```

Recommendation

The team is advised to carefully consider the logical conditions is using in the code and ensure that it is well-defined and make sense in the context of the smart contract.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	Imal.sol#L284
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
require(BPDisabledForever == false)
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Imal.sol#L221,296
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
beneficiaryAddress = beneficiaryAddress_
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	Imal.sol#L11,93,118,186
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 ^0.8.0;
```

Recommendation

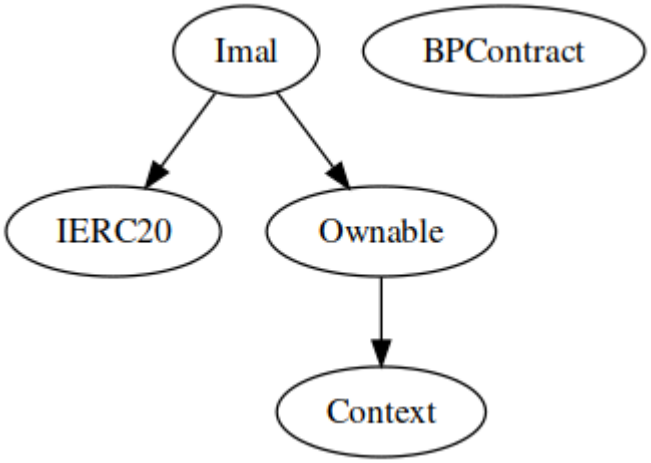
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest 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

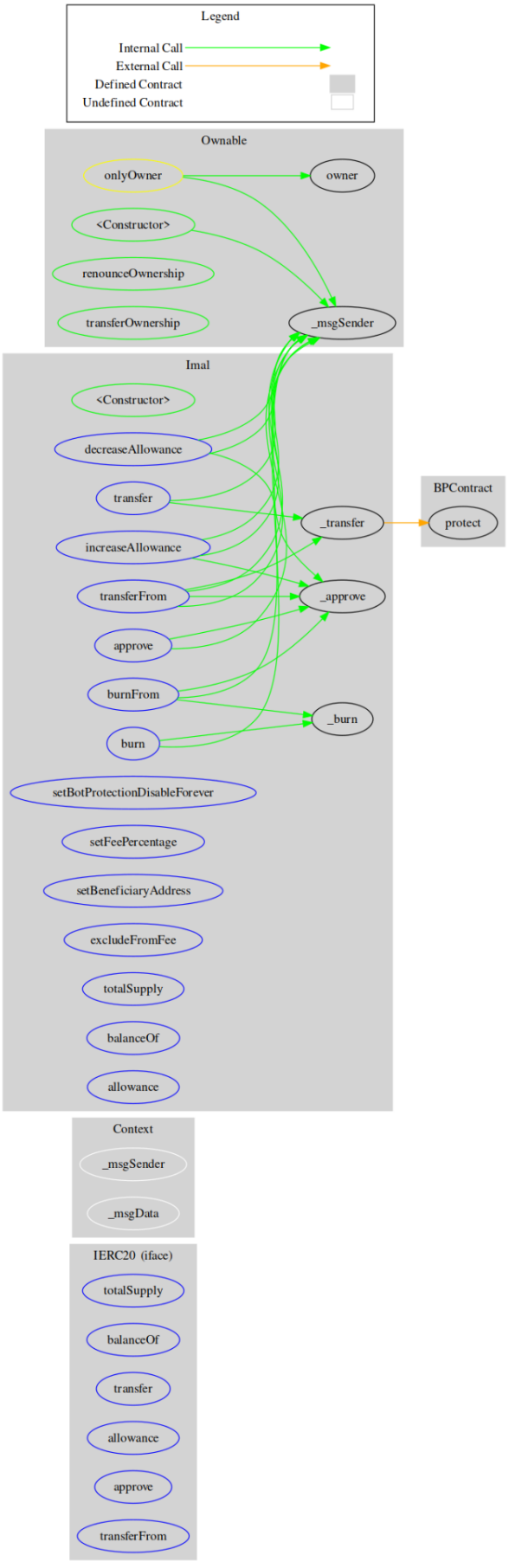
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner

BPContract	Implementation			
	protect	External	✓	-
Imal	Implementation	IERC20, Ownable		
		Public	✓	-
	burn	External	✓	-
	burnFrom	External	✓	-
	transfer	External	✓	-
	approve	External	✓	-
	transferFrom	External	✓	-
	increaseAllowance	External	✓	-
	decreaseAllowance	External	✓	-
	setBotProtectionDisableForever	External	✓	onlyOwner
	setFeePercentage	External	✓	onlyOwner
	setBeneficiaryAddress	External	✓	onlyOwner
	excludeFromFee	External	✓	onlyOwner
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	_transfer	Private	✓	
	_burn	Private	✓	
	_approve	Private	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Ownable
onlyOwner
owner
<Constructor>
renounceOwnership
transferOwnership
_msgSender
Imal
<Constructor>
decreaseAllowance
transfer
increaseAllowance
transferFrom
approve
burnFrom
burn
setBotProtectionDisableForever
setFeePercentage
setBeneficiaryAddress
excludeFromFee
totalSupply
balanceOf
allowance
_transfer
_approve
_burn
BPContract
protect
Context
_msgSender
_msgData
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom

Summary

IMAL contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. IMAL is an interesting project that has a friendly and growing community. There are some functions that can be abused by the owner. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract will eliminate all the contract threats. There is also a limit of max 10% fees.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/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 and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

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

https://www.cyberscope.io