Cyberscope

Audit Report

MagnumBit

April 2024

Network BSC_TESTNET

Address 0xc54474dbac77184655B9b4763b9329A346298217

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	Unresolved
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	IFLS	Ineffective Fee Limit Setting	Unresolved
Minor / Informative	DDP	Decimal Division Precision	Unresolved
Minor / Informative	EIS	Excessively Integer Size	Unresolved
Minor / Informative	IDI	Immutable Declaration Improvement	Unresolved
Minor / Informative	MEM	Missing Error Messages	Unresolved
Minor / Informative	MEE	Missing Events Emission	Unresolved
Minor / Informative	PAMAR	Pair Address Max Amount Restriction	Unresolved
Minor / Informative	RFM	Redundant Fee Mechanism	Unresolved
Minor / Informative	RFU	Redundant Functionalities Usage	Unresolved
Minor / Informative	RSW	Redundant Storage Writes	Unresolved
Minor / Informative	RC	Repetitive Calculations	Unresolved
Minor / Informative	L02	State Variables could be Declared Constant	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L07	Missing Events Arithmetic	Unresolved

●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	MetaForge
Compiler Version	v0.8.24+commit.e11b9ed9
Optimization	200 runs
Explorer	https://testnet.bscscan.com/address/0xc54474dbac77184655b9b4763b9329a346298217
Address	0xc54474dbac77184655b9b4763b9329a346298217
Network	BSC_TESTNET
Symbol	MFORGE
Decimals	18
Total Supply	109,999,997.865
Badge Eligibility	Must Fix Criticals

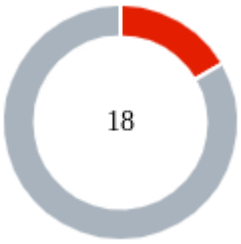
Audit Updates

Initial Audit	27 Apr 2024

Source Files

Filename	SHA256
contracts/Mforge.sol	19047b6755ef3dc4fbd5cd9ba214dd3e17c903dd4ff1919c5faf2aaf04e33d65

Findings Breakdown



Critical	3
Medium	0
Minor / Informative	15

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	3	0	0	0
Medium	0	0	0	0
Minor / Informative	15	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	contracts/Mforge.sol#L291
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 the `maxTransactionAmount` or `maxWallet` to zero. As a result, the contract may operate as a honeypot.

Additionally, if any of the `adminWallet` , `boostLiquidityWallet` , `lpProviderWallet` , `rewardsPoolWallet` addresses is set to zero, the contract will prevent the successful execution of the transfers.

```
if (isAMM[from] && !isExcludedFromWalletLimits[to]) {
    require(
        amount <= maxTransactionAmount,
        "!maxTransactionAmount."
    );
    require(amount + balanceOf(to) <= maxWallet, "!maxWallet");
} else if (isAMM[to] && !isExcludedFromWalletLimits[from]) {
    require(
        amount <= maxTransactionAmount,
        "!maxTransactionAmount."
    );
} else if (!isExcludedFromWalletLimits[to]) {
    require(amount + balanceOf(to) <= maxWallet, "!maxWallet");
}
...

super._transfer(from, adminWallet, adminFee);
...
super._transfer(from, boostLiquidityWallet, boostLpFee);
super._transfer(from, lpProviderWallet, lpProviderFee);

super._transfer(from, adminWallet, adminFee);
...
super._transfer(from, rewardsPoolWallet, rewardPoolFee);
super._transfer(from, lpProviderWallet, lpProviderFee);
```

Additionally, the contract owner has the authority to stop transactions, as described in detail in sections `PAMAR`. As a result, the contract might operate as a honeypot.

Recommendation

The contract could embody a check for not allowing setting the `maxTransactionAmount` and `maxWallet` less than a reasonable amount. A suggested implementation could check that the minimum amount should be more than a fixed percentage of the total supply. Additionally, the contract should prevent the set of the addresses to the zero address. 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.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	contracts/Mforge.sol#L117,131
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the `setBuyFees` or `setSellFees` function with a high percentage value.

```
    function setBuyFees(
        uint256 marketingFee,
        uint256 liquidityFee
    ) external onlyOwner {
        buyMarketingFee = marketingFee;
        buyLiquidityFee = liquidityFee;

        buyTotalFees = buyMarketingFee + buyLiquidityFee;

        if (maxBuyFeeSet) {
            require(buyTotalFees <= maxBuyFee);
        }
    }

    function setSellFees(
        uint256 marketingFee,
        uint256 liquidityFee
    ) external onlyOwner {
        sellMarketingFee = marketingFee;
        sellLiquidityFee = liquidityFee;

        sellTotalFees = sellMarketingFee + sellLiquidityFee;

        if (maxSellFeeSet) {
            require(sellTotalFees <= maxSellFee);
        }
    }
```

Recommendation

The contract could embody a check for the maximum acceptable value. 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.

IFLS - Ineffective Fee Limit Setting

Criticality	Critical
Location	contracts/Mforge.sol#L53,203,209
Status	Unresolved

Description

The contract includes the `enableMaxSellFeeLimit` and `enableMaxBuyFeeLimit` functionss intended to set maximum fee limits for buying and selling. However, both the `maxSellFee` and `maxBuyFee` are initialized to zero and the conditions within these functions require the new limit to be less than the existing `maxSellFee` or `maxBuyFee`. Given that these initial limits are set to zero, any non-zero limit passed to these functions will not satisfy the condition, rendering the functions incapable of ever setting a positive fee limit. This logical flaw prevents the effective use of these fee management features, defeating their purpose and potentially affecting the contract's functionality concerning fee limits.

```
    uint256 maxSellFee;
     uint256 maxBuyFee;

     function enableMaxSellFeeLimit(uint256 limit) external onlyOwner {
         require(limit <= feeDenominator && limit < maxSellFee);
         maxSellFee = limit;
         maxSellFeeSet = true;
     }

     function enableMaxBuyFeeLimit(uint256 limit) external onlyOwner {
         require(limit <= feeDenominator && limit < maxBuyFee);
         maxBuyFee = limit;
         maxBuyFeeSet = true;
     }
```

Recommendation

It is recommended to reconsider the conditions under which these functions can be called. Specifically, the contract should embody stable, non-zero maximum fee limits that can be adjusted only within defined bounds. One approach could be to initialize `maxSellFee` and `maxBuyFee` to reasonable default values that align with expected operational parameters. Further, it would be prudent to revise the conditions to allow updates that are less than or equal to the current max fees rather than strictly less, enabling the flexibility needed for practical fee management while maintaining control over fee ceiling settings.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	contracts/Mforge.sol#L244,264
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
uint256 adminFee = (fees * 500) / feeDenominator;
uint256 burnFee = (fees * 200) / feeDenominator;
uint256 boostLpFee = (fees * 100) / feeDenominator;
uint256 lpProviderFee = (fees * 200) / feeDenominator;
...
uint256 adminFee = (fees * 600 ) / feeDenominator;
uint256 burnFee = (fees * 100 ) / feeDenominator;
uint256 rewardPoolFee = (fees * 200 ) / feeDenominator;
uint256 lpProviderFee = (fees * 100 ) / feeDenominator;
```

Recommendation

The team is advised to take into consideration the rounding results that are produced from the solidity calculations. The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

EIS - Excessively Integer Size

Criticality	Minor / Informative
Location	contracts/Mforge.sol#L39
Status	Unresolved

Description

The contract is using a bigger unsigned integer data type that the maximum size that is required. By using an unsigned integer data type larger than necessary, the smart contract consumes more storage space and requires additional computational resources for calculations and operations involving these variables. This can result in higher transaction costs, longer execution times, and potential scalability bottlenecks.

```
uint256 public feeDenominator = 1000;
```

Recommendation

To address the inefficiency associated with using an oversized unsigned integer data type, it is recommended to accurately determine the required size based on the range of values the variable needs to represent.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/Mforge.sol#L106
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
swapThreshold
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

MEM - Missing Error Messages

Criticality	Minor / Informative
Location	contracts/Mforge.sol#L127,141,154,159,166,204,210,288,314,324
Status	Unresolved

Description

The contract is using missing error messages. These are no error messages to accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(buyTotalFees <= maxBuyFee)
require(sellTotalFees <= maxSellFee)
require(limitsInEffect)
require(LPTokenReceiver != newReceiver)
require(marketingReceiver != newReceiver)
require(limit <= feeDenominator && limit < maxSellFee)
require(limit <= feeDenominator && limit < maxBuyFee)
require(blockTransferCount[tx.origin][block.number] == 0)
require(tokenAddress != address(this))
require(success)
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/Mforge.sol#L177,181,188
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 setAMM(address ammAddress, bool isAMM_) external
onlyOwner {
        isAMM[ammAddress] = isAMM_;
    }

    function setWalletExcludedFromLimits(
        address wallet,
        bool isExcluded
    ) external onlyOwner {
        isExcludedFromWalletLimits[wallet] = isExcluded;
    }

    function setWalletExcludedFromFees(
        address wallet,
        bool isExcluded
    ) external onlyOwner {
        isExcludedFromFee[wallet] = 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.

PAMAR - Pair Address Max Amount Restriction

Criticality	Minor / Informative
Location	contracts/Mforge.sol#L302
Status	Unresolved

Description

The contract is configured to enforce a maximum token accumulation limit through checks. This mechanism aims to prevent excessive token concentration by reverting transactions that overcome the specified cap. However, this functionality encounters issues when transactions default to the pair address during sales. If the pair address is not listed in the exceptions, then the sale transactions are inadvertently stopped, effectively disrupting operations and making the contract susceptible to unintended behaviors akin to a honeypot.

```
else if (!isExcludedFromWalletLimits[to]) {
  require(amount + balanceOf(to) <= maxWallet, "!maxWallet");
```

Recommendation

It is advised to modify the contract to ensure uninterrupted operations by either permitting the pair address to exceed the established token accumulation limit or by safeguarding its status in the exception list. By recognizing and allowing these essential addresses the flexibility to hold more tokens than typical limits, the contract can maintain seamless transaction flows and uphold the liquidity and stability of the ecosystem. This modification is vital for avoiding disruptions that could impact the functionality and security of the contract.

RFM - Redundant Fee Mechanism

Criticality	Minor / Informative
Location	contracts/Mforge.sol#L238
Status	Unresolved

Description

The contract is designed to calculate and accumulate marketing and liquidity fees separately based on transactions. It specifies the use of distinct fee variables, to compute fees for marketing and liquidity respectively. These fees are then added to their corresponding pools, `tokensForMarketing` and `tokensForLiquidity`. However, the contract further processes the accumulated fees to calculate additional fees based on a percentage of the total accumulated fees. This results in a redundancy, as the initial separation into marketing and liquidity fees does not influence the subsequent calculations or fee distributions, but rather, all are derived from the total fees computed initially. This not only adds unnecessary complexity but also leads to inefficiencies in fee management and use.

```
   uint256 newTokensForMarketing = (amount * sellMarketingFee) /
feeDenominator;
    uint256 newTokensForLiquidity = (amount * sellLiquidityFee) /
feeDenominator;
    fees = newTokensForMarketing + newTokensForLiquidity;

    if (fees > 0) {
        uint256 adminFee = (fees * 500 ) / feeDenominator;
        uint256 burnFee = (fees * 200 ) / feeDenominator;
        uint256 boostLpFee = (fees * 100 ) / feeDenominator;
        uint256 lpProviderFee = (fees * 200 ) / feeDenominator;
    tokensForMarketing += newTokensForMarketing;
    tokensForLiquidity += newTokensForLiquidity;

} else if (isAMM[from] && buyTotalFees > 0) {
    uint256 newTokensForMarketing = (amount * buyMarketingFee) /
feeDenominator;
    uint256 newTokensForLiquidity = (amount * buyLiquidityFee) /
feeDenominator;
    fees = newTokensForMarketing + newTokensForLiquidity;

    if (fees > 0) {
        uint256 adminFee = (fees * 600 ) / feeDenominator;
        uint256 burnFee = (fees * 100 ) / feeDenominator;
        uint256 rewardPoolFee = (fees * 200 ) / feeDenominator;
        uint256 lpProviderFee = (fees * 100 ) / feeDenominator;
    ...
    tokensForMarketing += newTokensForMarketing;
    tokensForLiquidity += newTokensForLiquidity;
```

Recommendation

It is recommended to remove the marketing and liquidity fees since the contract does not base any functionality on these fees. Instead, the contract could calculate a general fee variable and then split the fee as needed. Simplifying the fee structure in this way will reduce complexity and potential errors, streamline transactions, and improve contract efficiency. This approach would allow the contract to maintain flexibility in fee allocation while reducing the overhead associated with managing multiple fee variables that ultimately serve no distinct purpose.

RFU - Redundant Functionalities Usage

Criticality	Minor / Informative
Location	contracts/Mforge.sol#L158,195,199
Status	Unresolved

Description

The contract is designed to contain functions that allow for the updating of specific contract variables. However, these variables are not utilized in any significant operational functions or logical processes within the contract's implementation. The presence of such functions not only contributes to increased contract complexity but also introduces potential security risks. These include unnecessary exposure to address modification functions and the maintenance of variable states that do not contribute to the contract's functionality or behavior. The inclusion of these redundant functions and variables may lead to potential misinterpretations of the contract's intended capabilities and unnecessarily complicate its architecture.

```
function setLPTokenReceiver(address newReceiver) external
onlyOwner {
    require(LPTokenReceiver != newReceiver);
    isExcludedFromFee[newReceiver] = true;
    isExcludedFromWalletLimits[newReceiver] = true;
    LPTokenReceiver = newReceiver;
}
function setRouter(address router_) external onlyOwner {
    router = IUniswapV2Router02(router_);
}
function setLiquidityPair(address pairAddress) external
onlyOwner {
    liquidityPair = pairAddress;
}
```

Recommendation

It is recommended to thoroughly review and remove such functions that update unused variables and any associated unused variables from the contract. Eliminating these redundancies will streamline the contract by removing unnecessary code, reducing the

attack surface related to unused functionalities, and minimizing potential confusion regarding the contract's actual functionalities and their purposes. Simplifying the contract in this manner enhances its security, maintainability, and efficiency, aligning it more closely with its operational objectives.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/Mforge.sol#L177,181,188
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
    function setAMM(address ammAddress, bool isAMM_) external
onlyOwner {
        isAMM[ammAddress] = isAMM_;
    }

    function setWalletExcludedFromLimits(
        address wallet,
        bool isExcluded
    ) external onlyOwner {
        isExcludedFromWalletLimits[wallet] = isExcluded;
    }

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

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before

proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

RC - Repetitive Calculations

Criticality	Minor / Informative
Location	contracts/Mforge.sol#L104
Status	Unresolved

Description

The contract contains methods with multiple occurrences of the same calculation being performed. The calculation is repeated without utilizing a variable to store its result, which leads to redundant code, hinders code readability, and increases gas consumption. Each repetition of the calculation requires computational resources and can impact the performance of the contract, especially if the calculation is resource-intensive.

```
maxTransactionAmount = (totalSupply * 1000) / 1000;
maxWallet = (totalSupply * 1000) / 1000;
swapThreshold = (totalSupply * 1000) / 1000;
```

Recommendation

To address this finding and enhance the efficiency and maintainability of the contract, it is recommended to refactor the code by assigning the calculation result to a variable once and then utilizing that variable throughout the method. By storing the calculation result in a variable, the contract eliminates the need for redundant calculations and optimizes code execution.

Refactoring the code to assign the calculation result to a variable has several benefits. It improves code readability by making the purpose and intent of the calculation explicit. It also reduces code redundancy, making the method more concise, easier to maintain, and gas effective. Additionally, by performing the calculation once and reusing the variable, the contract improves performance by avoiding unnecessary computations

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/Mforge.sol#L39,41
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.

```
uint256 public feeDenominator = 1000
bool private swapping
```

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	contracts/Mforge.sol#L9,171
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.

```
address public LPTokenReceiver
address LpProviderWallet
address RewardsPoolWallet
address AdminWallet
address BoostLiquidityWallet
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	contracts/Mforge.sol#L121,135,149
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
buyMarketingFee = marketingFee
sellMarketingFee = marketingFee
maxTransactionAmount = maxTransactionAmount_
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/Mforge.sol#L65,66,67,68,69,70,172,173,174,175,200
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.

```
LPTokenReceiver = LPTokenReceiver_
marketingReceiver = marketingReceiver_
adminWallet = AdminWallet
boostLiquidityWallet = BoostLiquidityWallet
lpProviderWallet = LpProviderWallet
rewardsPoolWallet = RewardsPoolWallet
liquidityPair = pairAddress
```

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	contracts/Mforge.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 ^0.8.17;
```

Recommendation

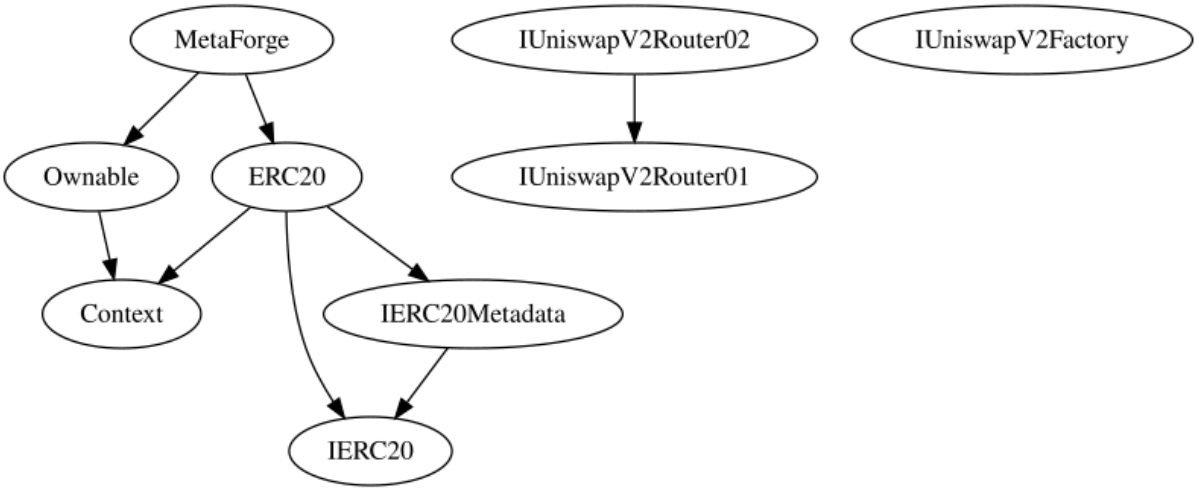
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest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

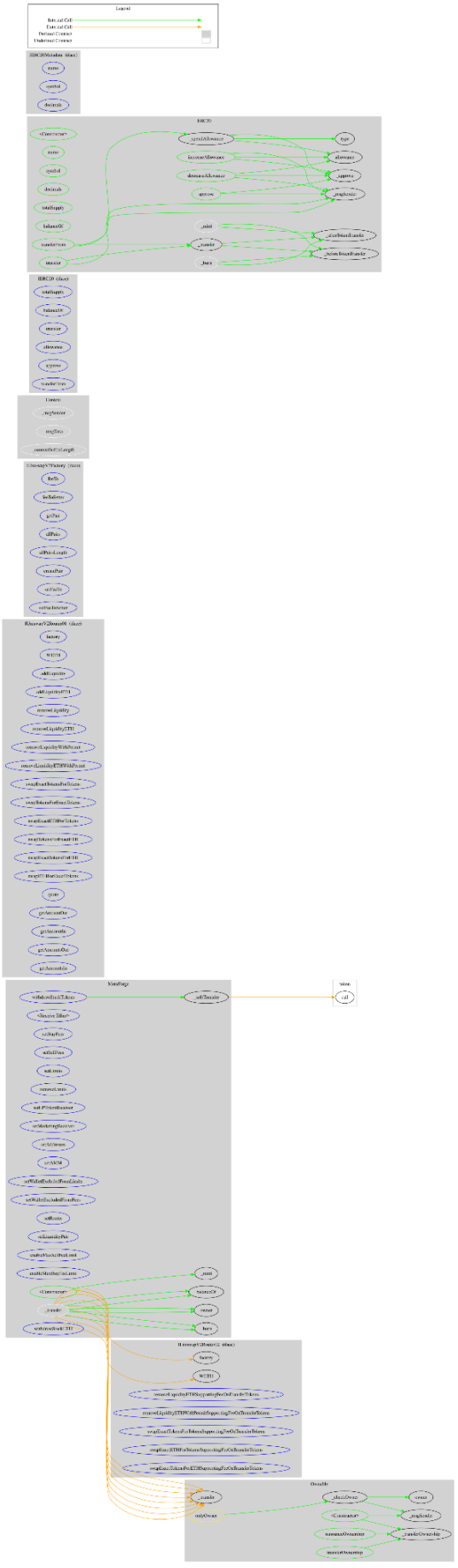
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
MetaForge	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	setBuyFees	External	✓	onlyOwner
	setSellFees	External	✓	onlyOwner
	setLimits	External	✓	onlyOwner
	removeLimits	External	✓	onlyOwner
	setLPTokenReceiver	External	✓	onlyOwner
	setMarketingReceiver	External	✓	onlyOwner
	setAddresses	External	✓	onlyOwner
	setAMM	External	✓	onlyOwner
	setWalletExcludedFromLimits	External	✓	onlyOwner
	setWalletExcludedFromFees	External	✓	onlyOwner
	setRouter	External	✓	onlyOwner
	setLiquidityPair	External	✓	onlyOwner
	enableMaxSellFeeLimit	External	✓	onlyOwner
	enableMaxBuyFeeLimit	External	✓	onlyOwner
	_transfer	Internal	✓	
	withdrawStuckTokens	External	✓	-

	withdrawStuckETH	External	✓	-
	_safeTransfer	Private	✓	

Inheritance Graph



Flow Graph



Summary

MagnumBit contract implements a token 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 and manipulate the fees. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

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