Cyberscope

Audit Report

Magawa

June 2024

Network	BASE
Address	0x82D1509e4F1C7468FbAa702eC69149EFA0b45574
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
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Analysis 1
Diagnostics 2
Table of Contents 3
Review 4
Audit Updates 4
Source Files 4
Findings Breakdown 5
L04 - Conformance to Solidity Naming Conventions 6
Description 6
Recommendation 7
L09 - Dead Code Elimination 8
Description 8
Recommendation 8
L13 - Divide before Multiply Operation 10
Description 10
Recommendation 10
L15 - Local Scope Variable Shadowing 11
Description 11
Recommendation 11
L17 - Usage of Solidity Assembly 12
Description 12
Recommendation 12
L18 - Multiple Pragma Directives 13
Description 13
Recommendation 13
L19 - Stable Compiler Version 14
Description 14
Recommendation 14
Functions Analysis 15
Inheritance Graph 16
Flow Graph 17
Summary 18
Disclaimer 19
About Cyberscope 20

Review

Contract Name	Magawa
Compiler Version	v0.8.25+commit.b61c2a91
Optimization	200 runs
Explorer	https://basescan.org/address/0x82d1509e4f1c7468fbaa702ec69149efa0b45574
Address	0x82d1509e4f1c7468fbaa702ec69149efa0b45574
Network	BASE
Symbol	MAGAWA
Decimals	18
Total Supply	1,000,000,000
Badge Eligibility	Yes

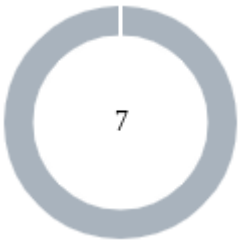
Audit Updates

Initial Audit	13 Jun 2024

Source Files

Filename	SHA256
Magawa.sol	bc583594f145f0b779f45f256d2afb02af156ad48542629643f5ad2387c285cf

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	7

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	7	0	0	0

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Magawa.sol#L1138,1149,1421,2229
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.

```
function _EIP712Name() internal view returns (string memory) {
        return _name.toStringWithFallback(_nameFallback);
    }


function _EIP712Version() internal view returns (string memory)
{
        return _version.toStringWithFallback(_versionFallback);
    }
}
...
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	Magawa.sol#L42,141,151,161,171,181,201,211,335,358,365,373,382,472,479,487,498,514,598,612,648,659,701,712,750,763,793,803,832,857,864,873,892,899,934,953,967,1211,1244,1255,1267,1614
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function _useCheckedNonce(address owner, uint256 nonce) internal
virtual {
        uint256 current = _useNonce(owner);
        if (nonce != current) {
            revert InvalidAccountNonce(owner, current);
        }
    }

function getAddressSlot(bytes32 slot) internal pure returns
(AddressSlot storage r) {
        /// @solidity memory-safe-assembly
        assembly {
            r.slot := slot
        }
    }

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	Magawa.sol#L560,563,575,579,580,581,582,583,584,590
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.

```
denominator := div(denominator, twos)
inverse *= 2 - denominator * inverse
```

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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	Magawa.sol#L2188
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
constructor(string memory name) EIP712(name, "1") {}
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	Magawa.sol#L143,153,163,173,183,193,203,213,287,521,838,936,982,1219
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
            r.slot := slot
        }

assembly {
            r.slot := store.slot
...
            mstore(add(str, 0x20), sstr)
        }

assembly {
                let mm := mulmod(x, y, not(0))
                prod1 := sub(sub(mm, prod0), lt(mm, prod0))
            }

...
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	Magawa.sol#L5,55,87,224,349,395,813,909,997,1159,1336,1428,1593,1624,1726,1808,1836,2154,2239,2279
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity ^0.8.20;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	Magawa.sol#L5,55,87,224,349,395,813,909,997,1159,1336,1428,1593,1624,1726,1808,1836,2154,2239,2279
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 ^0.8.20;
...
```

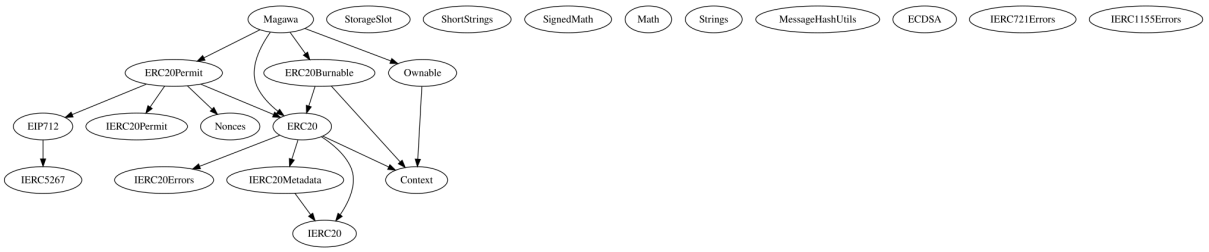
Recommendation

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.

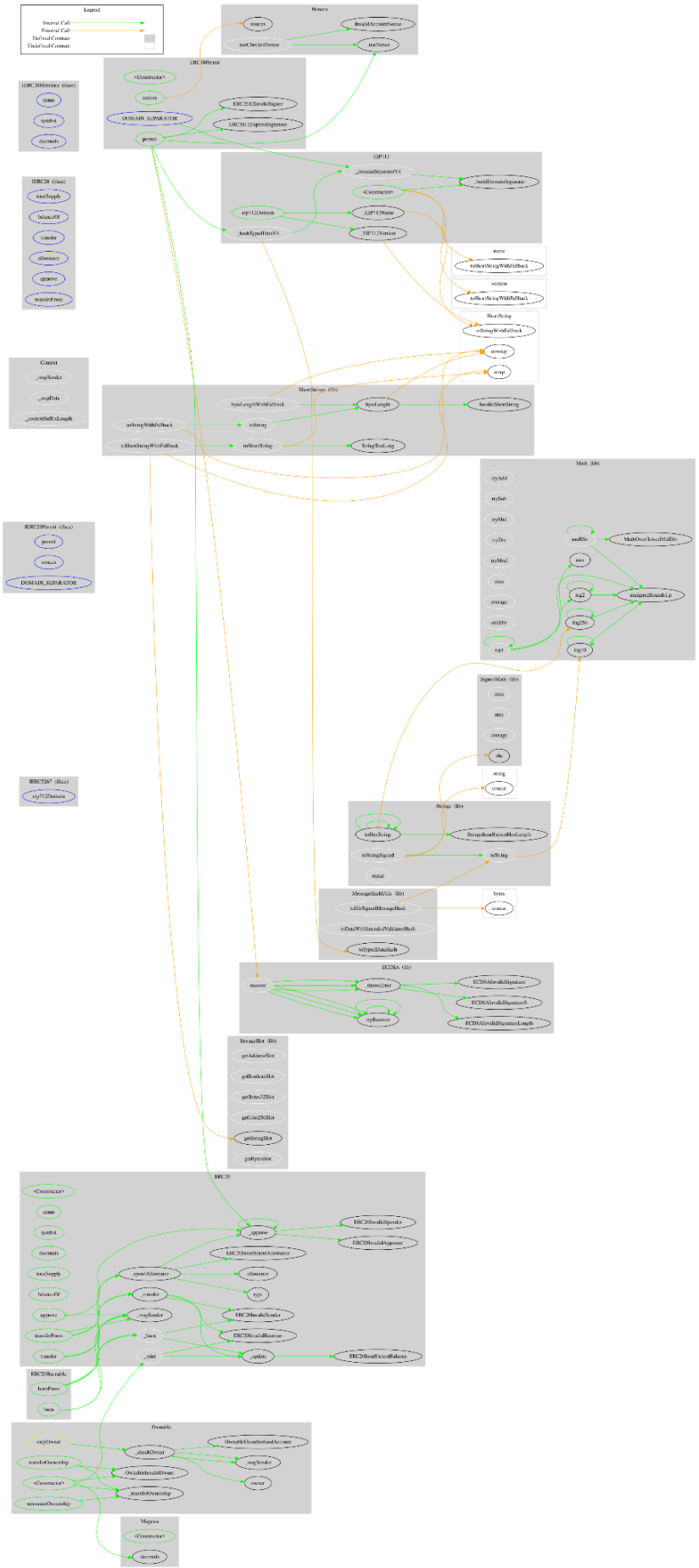
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Magawa	Implementation	ERC20, ERC20Burnable, ERC20Permit, Ownable		
		Public	✓	ERC20 ERC20Permit Ownable

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Nonces
nonces
invalidAccountNonce
maxCheckerOutOfGas
maxNonce
ERC20Permit
<Constructor>
nonces
DOMAIN_SEPARATOR
permit
ERC2612InvalidSigner
ERC2612ExpiredSignature
IERC20Metadata (Interface)
name
symbol
decimals
EIP712
_domainSeparatorV4
<Constructor>
buildDomainSeparator
eip712Domain
hashTypedDataV4
EIP712Name
EIP712Version
IERC20 (Interface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
name
toShortStringWithFallback
version
toShortStringWithFallback
ShortString
toString
Context
_msgSender
_msgData
_contextSuffixLength
ShortStrings (lib)
byteLength
byteLength
InvalidShortString
toString
toShortString
StringTooLong
Math (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
max
average
ceilDiv
sqrt
mulDiv
MathOverflowedMulDiv
min
log2
unsignedRoundsUp
log256
log10
IERC20Permit (Interface)
permit
nonces
DOMAIN_SEPARATOR
SignedMath (lib)
max
min
average
abs
IERC5267 (Interface)
eip712Domain
string
concat
Strings (lib)
toHexString
StringInsufficientHexLength
toString
toString
equal
MessageHashUtils (lib)
toEthSignedMessageHash
toDataWithIntendedValidatorHash
toTypedDataHash
bytes
concat
ECDSA (lib)
recover
tryRecover
ECDSAInvalidSignature
ECDSAInvalidSignatureS
ECDSAInvalidSignatureLength
StorageSlot (lib)
getAddressSlot
getBooleanSlot
getBytes32Slot
getUint256Slot
getStringSlot
getBytesSlot
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
approve
transferFrom
transfer
_approve
ERC20InvalidSpender
ERC20InvalidApprover
IERC20InsufficientAllowance
spendAllowance
allowance
_transfer
type
_msgSender
ERC20InvalidSender
_burn
ERC20InvalidReceiver
_mint
_update
ERC20InsufficientBalance
ERC20Burnable
burnFrom
burn
Ownable
onlyOwner
_checkOwner
OwnableUnauthorizedAccount
transferOwnership
<Constructor>
OwnableInvalidOwner
_transferOwnership
renounceOwnership
_msgSender
owner
Magawa
<Constructor>
decimals

Summary

Magawa contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Magawa is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions.

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