HashEx
BLOCKCHAIN SECURITY

MAGACOIN FINANCE

smart contracts
final audit report

March 2025

hashex.org

contact@hashex.org

Contents

1. Disclaimer	3
2. Overview	4
3. Project centralization risks	6
4. Found issues	7
5. Contracts	9
6. Conclusion	13
Appendix A. Issues' severity classification	14
Appendix B. Issue status description	15
Appendix C. List of examined issue types	16
Appendix D. Centralization risks classification	17

1. Disclaimer

2. Overview

HashEx was commissioned by the Magacoin team to perform an audit of their smart contract. The audit was conducted between 15/03/2025 and 19/03/2025.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The code is available in the Sepolia testnet at 0xE78a5D27aC8A2F2c0ce90486eD075bD584dCf9D6 and 0xFE979f253a29a780D13849CE9252F44cf9772304.

The contracts are deployed with TransparentUpgradeableProxy from OpenZeppelin repository. The audited implementations are available at 0x201Ea7f721015c06DF2C48141a50b60aec01497a (MagaToken) and 0xC142ac4174cF2d65c48D3cD8C59D74566b95FbB3 (MagaSale).

Update. The contracts were deployed to Ethereum mainnet with TransparentUpgradeableProxy to addresses 0x4fd6b7af49597ad1103bba25694de772ee44db7e and 0x2e2b809a3c46b20bab4177b8d1068ff29f926adb. The audited implementations at Ethereum mainnet are available at 0x808B888092400BB8BA84eee4D74492AE9bE7649A and 0x87152A700A40f55Ec5113Fb8a42eA8c53D8f3140.

2.1 Summary

Project name	MAGACOIN FINANCE
URL	https://magacoinfinance.com
Platform	Ethereum
Language	Solidity
Centralization level	● High
Centralization risk	● High

2.2 Contracts

Name	Address
MagaToken	`0x808B888092400BB8BA84eee4D74492AE9bE7649A`
MagaSale	`0x87152A700A40f55Ec5113Fb8a42eA8c53D8f3140`

3. Project centralization risks

The project contracts are designed to be deployed with proxies. Users have no choice but to trust the owners, who can update the contracts at their will.

We recommend secure the ownership with combination of timelock as updater and multisig smart account as Timelock's admin.

Ca4CR3f Owner privileges

The owner can upgrade the token implementation to mint new tokens or manipulate existing balances.

The owner can pause and unpause all transfers unless sender is included in the whitelist.

Ca5CR40 Owner privileges

The owner can upgrade the implementation to modify contract's logic.

The owner can update signer's list and minimum required signatures.

The owner can update payment receiving address.

Ther owner can transfer out ERC20 tokens from contract's balance.

4. Found issues

8
Total issues

●	Low	4 (50%)
●	Info	4 (50%)

Ca4. MagaToken

ID	Severity	Title	Status
Ca4I25	● Low	Gas optimizations	Acknowledged
Ca4I26	● Low	Inconsistent token name and symbol	Resolved
Ca4I24	● Info	Missing disableInitializers in constructor	Acknowledged

Ca5. MagaSale

ID	Severity	Title	Status
Ca5I20	● Low	Gas optimizations	Acknowledged
Ca5I21	● Low	Lack of input validation	Acknowledged
Ca5I23	● Info	Lack of monitoring tools	Acknowledged
Ca5I27	● Info	Incorrect nonReentrant modifier placement	Acknowledged

Ca5I22	● Info	Events parameters	✓ Acknowledged

5. Contracts

Ca4. MagaToken

Overview

An ERC20 standard token contract. The initial supply is fixed, i.e. there's no active minting functionality after deployment. The token contract is inherited from OpenZeppelin's ERC20 implementation, which is considered the best practice.

Issues

Ca4I25 Gas optimizations ● Low ✓ Acknowledged

The function `initialize()` can be declared as `external` instead of `public`.

Ca4I26 Inconsistent token name and symbol ● Low ✓ Resolved

The contract is initialized with the name **"MAGACOIN"** and symbol **"MAGACOIN"**, but it overrides these values with **"MAGACOIN FINANCE"** and **"MAGAFINANCE"** in the `name()` and `symbol()` functions.

According to the provided documentation, the expected token name should be **"MAGACOIN FINANCE"** and the symbol should be **"MAGAFINANCE"**, but these values are not reflected in the contract.

Recommendation

Provide the correct name and symbol from documentation, remove unnecessary `name()` and `symbol()` overrides.

Ca4I24 Missing disableInitializers in constructor ● Info ✓ Acknowledged

The contract lacks a call to `_disableInitializers()` in the constructor. If the contract deployed as a non-proxy, anyone can call `initialize()` after deployment and set himself a the owner of the contract.

```
constructor() {
    //
}
```

Recommendation

Modify the constructor to include `_disableInitializers()` to prevent unintended re-initialization:

```
constructor() {
    _disableInitializers();
}
```

Ca5. MagaSale

Overview

A sale contract allowing users to buy or claim ERC20 tokens (MagaToken) for one of supported payment methods (stablecoins or native ETH).

The purchase prices are defined by authorized signer's, i.e., a backend service.

Issues

Ca5I20 Gas optimizations ● Low ✓ Acknowledged

1. Unnecessary reads from storage in the `isPaymentToken()` function: payment token status could be checked by using a mapping.

2. Unnecessary `if(...) return true else return false` condition in the `isPaymentToken()` function.

Ca5I21 Lack of input validation ● Low ✓ Acknowledged

In the `changeMinSignatures()`, there's no safety check against `minSignatures` being greater than total number of signers. Setting it incoherent would cause a malfunction of signing authorization requiring additional operations from default admins.

Ca5I23 Lack of monitoring tools ● Info ✓ Acknowledged

The basic version of the AccessControl contract does not allow to get all members of certain roles. The AccessControlEnumerable extension contains additional getters.

Ca5I27 Incorrect nonReentrant modifier placement ● Info ✓ Acknowledged

The `nonReentrant` modifier is placed after other modifiers in function declarations, reducing its effectiveness as a security control. When nonReentrant is not the first modifier, other modifiers may execute code before the reentrancy guard is activated, potentially allowing reentrancy attacks.

```
function recoverERC20(
    address token,
    uint256 amount
) external onlyRole(DEFAULT_ADMIN_ROLE) nonReentrant {...}
```

Ca5I22 Events parameters ● Info ✓ Acknowledged

Emitted events don't use indexed parameters, which could be useful for off-chain monitoring.

For example, `Claim` and `Buy` events could use user address as indexed parameter to allow reconstruction of user's history.

6. Conclusion

4 low severity issues were found during the audit. 1 low issue was resolved in the update. The reviewed contracts are highly dependent on the owner's account. See the centralization risks chapter.

This audit includes recommendations on code improvement and the prevention of potential attacks.

Appendix A. Issues’ severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Informational.** Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

Appendix B. Issue status description

Resolved. The issue has been completely fixed.

Partially fixed. Parts of the issue have been fixed but the issue is not completely resolved.

Acknowledged. The team has been notified of the issue, no action has been taken.

Open. The issue remains unresolved.

Appendix C. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

Appendix D. Centralization risks classification

Centralization level

- 🔴 **High.** The project owners can manipulate user's funds, lock user's funds on their will (reversible or irreversible), or maliciously update contracts parameters or bytecode.
- 🟠 **Medium.** The project owners can modify contract's parameters to break some functions of the project contract or contracts, but user's funds remain withdrawable.
- 🟢 **Low.** The contract is trustless or its governance functions are safe against a malicious owner.

Centralization risk

- 🔴 **High.** Lost ownership over the project contract or contracts may result in user's losses. Contract's ownership belongs to EOA or EOAs, and their security model is unknown or out of scope.
- 🟠 **Medium.** Contract's ownership is transferred to a contract with not industry-accepted parameters, or to a contract without an audit. Also includes EOA with a documented security model, which is out of scope.
- 🟢 **Low.** Contract's ownership is transferred to a well-known or audited contract with industry-accepted parameters.

contact@hashex.org

@hashex_manager

blog.hashex.org

linkedin

github

twitter

#HashEx

BLOCKCHAIN SECURITY