Cyberscope

# Audit Report

# Edma

April 2025

| | |
|---|---|
| Network | SEPOLIA |
| Address | 0x91d4f39e63F1A770C0C7A1a6F97eF4fFB6Ed2B55 |
| 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 |
|---|---|---|---|
| ● | CCR | Contract Centralization Risk | Unresolved |
| ● | UAR | Unexcluded Address Restrictions | Unresolved |
| ● | ISV | Inconsistent Secondary Vesting | Unresolved |
| ● | L04 | Conformance to Solidity Naming Conventions | Unresolved |
| ● | L13 | Divide before Multiply Operation | Unresolved |
| ● | L19 | Stable Compiler Version | Unresolved |

# Table of Contents

**Analysis** 1
**Diagnostics** 2
**Table of Contents** 3
**Risk Classification** 5
**Review** 6
Audit Updates 6
Source Files 6
**Findings Breakdown** 7
CCR - Contract Centralization Risk 8
Description 8
Recommendation 8
UAR - Unexcluded Address Restrictions 9
Description 9
Recommendation 9
ISV - Inconsistent Secondary Vesting 10
Description 10
Recommendation 10
L04 - Conformance to Solidity Naming Conventions 11
Description 11
Recommendation 11
L13 - Divide before Multiply Operation 12
Description 12
Recommendation 12
L19 - Stable Compiler Version 13
Description 13
Recommendation 13
**Functions Analysis** 14
**Inheritance Graph** 16
**Flow Graph** 17
**Summary** 18
**Disclaimer** 19
**About Cyberscope** 20

# Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

| Severity | Likelihood / Impact of Exploitation |
|---|---|
| ● Critical | Highly Likely / High Impact |
| ● Medium | Less Likely / High Impact or Highly Likely/ Lower Impact |
| ● Minor / Informative | Unlikely / Low to no Impact |

# Review

| | |
|---|---|
| **Contract Name** | EDMA |
| **Compiler Version** | v0.8.20+commit.a1b79de6 |
| **Optimization** | 200 runs |
| **Explorer** | https://sepolia.etherscan.io/address/0x91d4f39e63f1a770c0c7a1a6f97ef4ffb6ed2b55 |
| **Address** | 0x91d4f39e63F1A770C0C7A1a6F97eF4fFB6Ed2B55 |
| **Network** | SEPOLIA |
| **Symbol** | EDM |
| **Decimals** | 18 |
| **Total Supply** | 500,000,000 |

## Audit Updates

| | |
|---|---|
| **Initial Audit** | 10 Apr 2025<br>https://github.com/cyberscope-io/audits/blob/main/1-edm/v1/audit.pdf |
| **Corrected Phase 2** | 25 Apr 2025 |

## Source Files

| **Filename** | SHA256 |
|---|---|
| **EDMA.sol** | 2c0ff8b94f4364318d1bdf31e9079ddfb231cfc79ca1ce9e8b34e591883b3421 |

# Findings Breakdown


| Severity | Unresolved | Acknowledged | Resolved | Other |
|---|---|---|---|---|
| Critical | 0 | 0 | 0 | 0 |
| Medium | 0 | 0 | 0 | 0 |
| Minor / Informative | 6 | 0 | 0 | 0 |

## CCR - Contract Centralization Risk

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | edma.sol#L267,272 |
| **Status** | Unresolved |

### Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

```
function endPresale() external onlyOwner activePresale returns
(bool){
presaleActive = false;
presaleEndTime = block.timestamp;
emit PresaleEnded(presaleEndTime);
return true;
}
```

```
function setPresale(address newPresaleAddress) external onlyOwner
activePresale validAddress(newPresaleAddress) {
emit PresaleAddressUpdated(presaleAddress, newPresaleAddress);
presaleAddress = newPresaleAddress;
}
```

### Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

## UAR - Unexcluded Address Restrictions

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | EDMA.sol#L340 |
| **Status** | Unresolved |

### Description

The contract incorporates operational restrictions on transactions, which can hinder seamless interaction with decentralized applications (dApps) such as launchpads, presales, lockers, or staking platforms. In scenarios where an external contract, such as a launchpad factory, needs to integrate with the contract, it should be exempt from the limitations to ensure uninterrupted service and functionality. Failure to provide such exemptions can block the successful process and operation of services reliant on this contract.

```
function _transfer(address sender, address recipient, uint256 amount) internal
validAddress(recipient) validAddress(sender) {

if(amount > 0) {
checkIfCanSpend(sender, amount);
}
_balances[sender] -= amount;
_balances[recipient] += amount;

emit Transfer(sender, recipient, amount);
}
```

### Recommendation

It is advisable to modify the contract by incorporating functionality that enables the exclusion of designated addresses from transactional restrictions. This enhancement will allow specific addresses, such as those associated with decentralized applications (dApps) and service platforms, to operate without being hindered by the standard constraints imposed on other users. Implementing this feature will ensure smoother integration and functionality with external systems, thereby expanding the contract's versatility and effectiveness in diverse operational environments.

## ISV - Inconsistent Secondary Vesting

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | EDMA.sol#L224 |
| **Status** | Unresolved |

### Description

The contract allows the vesting of tokens received from the `preSaleAddress` in multiple stages. If tokens are vested after the complete release of the previously locked balance, the released amounts from the newly vested balances are calculated proportionally to the previously locked amounts. This may result in the early release of the newly vested amount. For example, if a user receives 100 tokens that are released over the span of 5 periods, and then the user receives an additional 25 tokens, these can be released in the span of a single period. This is because the calculation of the amount to be released includes the already released balance of 100 tokens. As a result, the release schedule of new vesting periods may not follow the expected design.

```
vesting[recipient].totalLocked =  vesting[recipient].totalLocked + amount;
```

### Recommendation

It is advisable to include measures that properly implement the release schedule for vested amounts at all times. This would prevent inconsistencies in the system.

## L04 - Conformance to Solidity Naming Conventions

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | edma.sol#L134,135,136 |
| **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.

```
string private constant _name = "EDMA"
string private constant _symbol = "EDM"
uint8 private constant _decimals = 18
```

### 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/stable/style-guide.html#naming-conventions.

## L13 - Divide before Multiply Operation

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | edma.sol#L245,255 |
| **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.

```
uint256 intervalPassed = ((block.timestamp - presaleEndTime) /
vestingInterval)
uint256 totalReleasable  = (vs.totalLocked * (intervalPassed * 20)) / 100
```

### 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.

## L19 - Stable Compiler Version

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | edma.sol#L4 |
| **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

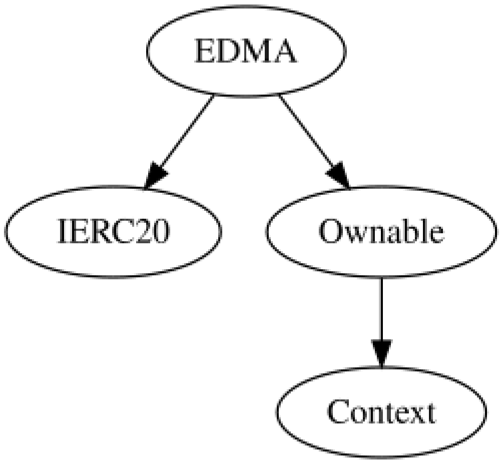
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest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

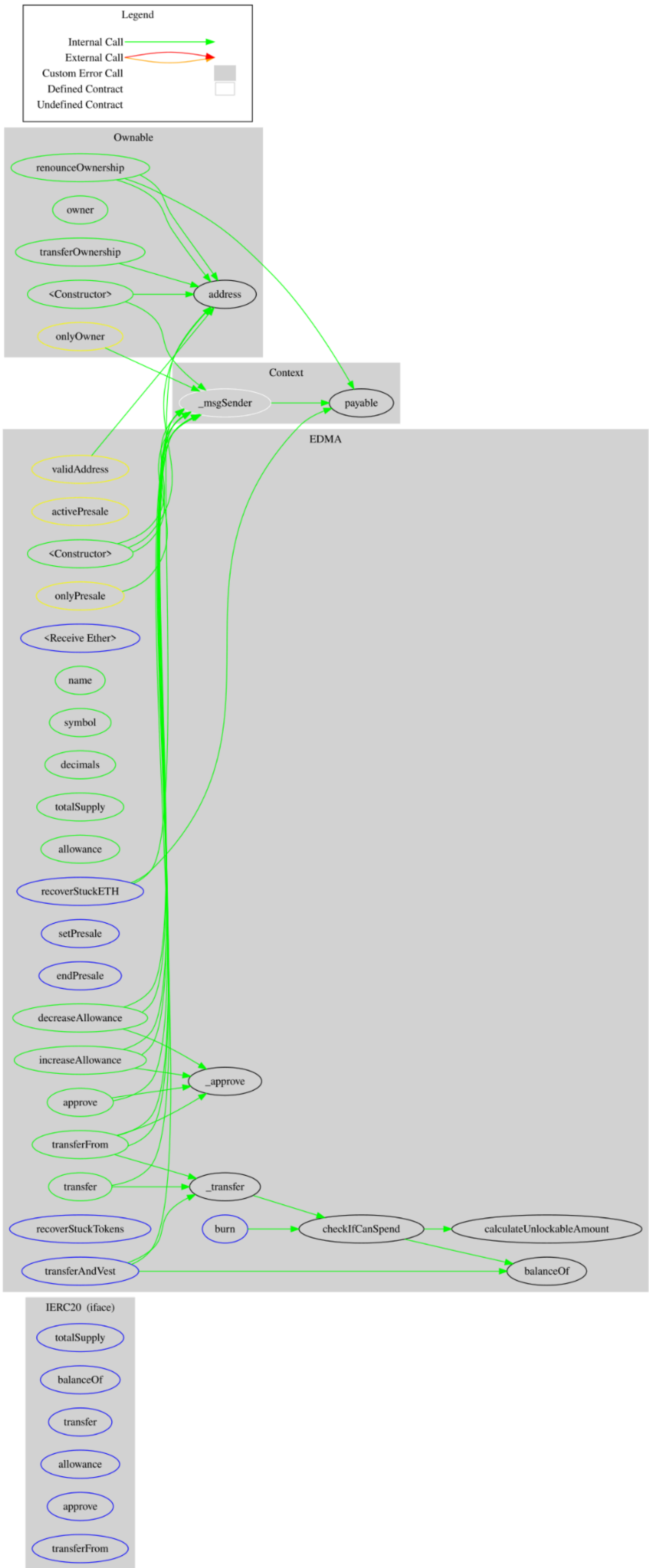
| Contract | Type | Bases | | |
|---|---|---|---|---|
| | Function Name | Visibility | Mutability | Modifiers |
| | | | | |
| IERC20 | Interface | | | |
| | totalSupply | External | | - |
| | balanceOf | External | | - |
| | transfer | External | ✓ | - |
| | allowance | External | | - |
| | approve |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

| | | | | |
|---|---|---|---|---|
| | | External | Payable | - |
| | name | Public | | - |
| | symbol | Public | | - |
| | decimals | Public | | - |
| | totalSupply | Public | | - |
| | balanceOf | Public | | - |
| | allowance | Public | | - |
| | burn | External | ✓ | - |
| | transferAndVest | External | ✓ | onlyPresale activePresale validAddress |
| | calculateUnlockableAmount | Public | | - |
| | setPresale | External | ✓ | onlyOwner activePresale validAddress |
| | endPresale | External | ✓ | onlyOwner activePresale |
| | approve | Public | ✓ | - |
| | transfer | Public | ✓ | - |
| | transferFrom | Public | ✓ | - |
| | increaseAllowance | Public | ✓ | - |
| | decreaseAllowance | Public | ✓ | - |
| | recoverStuckTokens | External | ✓ | onlyOwner validAddress |
| | recoverStuckETH | External | ✓ | onlyOwner validAddress |
| | checkIfCanSpend | Internal | ✓ | |
| | _transfer | Internal | ✓ | validAddress validAddress |
| | _approve | Internal | ✓ | validAddress validAddress |

# Inheritance Graph



# Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
Ownable
renounceOwnership
owner
transferOwnership
<Constructor>
address
onlyOwner
Context
_msgSender
payable
EDMA
validAddress
activePresale
<Constructor>
onlyPresale
<Receive Ether>
name
symbol
decimals
totalSupply
allowance
recoverStuckETH
setPresale
endPresale
decreaseAllowance
increaseAllowance
_approve
approve
transferFrom
transfer
_transfer
recoverStuckTokens
burn
checkIfCanSpend
calculateUnlockableAmount
transferAndVest
balanceOf
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom

# Summary

Edma contract implements a token and vesting mechanism. This audit investigates security issues, business logic concerns and potential improvements. The Smart Contract analysis reported no compiler error or critical issues.

# 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**

cyberscope.io