Cyberscope

Audit Report

Edma

April 2025

Network ETH

Address 0xe3249ec8902db06bb3d75e2e9b3998f7f2a09cc5

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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	IRA	Inconsistent Released Amounts	Unresolved
	ISV	Inconsistent Secondary Vesting	Unresolved
	MMN	Misleading Method Naming	Unresolved
	MEE	Missing Events Emission	Unresolved
	MU	Modifiers Usage	Unresolved
	NWES	Nonconformity with ERC-20 Standard	Unresolved
	PGA	Potential Griefing Attack	Unresolved
	PSU	Potential Subtraction Underflow	Unresolved
	RCS	Redundant Conditional Statements	Unresolved
	RTR	Release Time Reset	Unresolved
	UAR	Unexcluded Address Restrictions	Unresolved
	UVI	Unrestricted Voting Interval	Unresolved
	ZD	Zero Division	Unresolved
	L02	State Variables could be Declared Constant	Unresolved

●	L06	Missing Events Access Control	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

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://etherscan.io/address/0xe3249ec8902db06bb3d75e2e9b3998f7f2a09cc5
Address	0xe3249ec8902db06bb3d75e2e9b3998f7f2a09cc5
Network	ETH
Symbol	EDM
Decimals	18
Total Supply	500,000,000

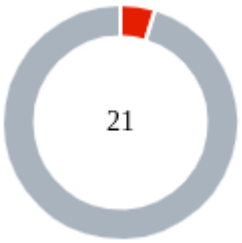
Audit Updates

Initial Audit	10 Apr 2025

Source Files

Filename	SHA256
EDMA.sol	793818c403d6985dd1ee2c78cc9dfd2e7e04da556b7b4ad94ee5af913853d91b

Findings Breakdown



Critical	1
Medium	0
Minor / Informative	20

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	20	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	EDMA.sol#L273
Status	Unresolved

Description

The contract owner has the authority to stop transactions for all users. The owner may take advantage of it by setting the `vestingInterval` to zero or a relatively large value. As a result, certain transactions may be stopped.

```
function setvestingInterval(uint256 second) external onlyOwner {
vestingInterval = second;
    }
```

Additionally, the contract owner has the authority to stop transactions, as described in detail in sections RTR , PGA , UVI and ZD.

Recommendation

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.

IRA - Inconsistent Released Amounts

Criticality	Minor / Informative
Location	EDMA.sol#L322
Status	Unresolved

Description

The contract invokes the `checkIfCanSpend` function during each transfer to confirm whether the sender is allowed to transfer tokens. During this execution, the expected release amount is calculated based on the total locked amount and the elapsed time since the last transfer. If the user is subject to any restrictions, the `lastReleasedTime` is updated to the current timestamp. If the user transfers less than the released amount and wishes to initiate a second transfer, the new calculation will be based on the last timestamp rather than the start of the vesting period. As a result, the previously released amount will now appear as locked. This design introduces inconsistencies and unpredictability for the user, who may not be able to access their balance when expected.

```
function checkIfCanSpend(address sender, uint256 amount) internal
returns(bool) {
uint256 toRelease = calculateToBeUnlockNext(sender);
if (toRelease > 0) {
vesting[sender].totalReleased = vesting[sender].totalReleased +
toRelease;
vesting[sender].lastRelasedTime = block.timestamp;
}
...
}
```

Recommendation

It is advised to consistently track the released amount in a manner that ensures they remain accessible to the user throughout all instances after the release period. This can be achieved by maintaining state information about the released amounts for each user.

ISV - Inconsistent Secondary Vesting

Criticality	Minor / Informative
Location	EDMA.sol#L230
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.

MMN - Misleading Method Naming

Criticality	Minor / Informative
Location	EDMA.sol#L241
Status	Unresolved

Description

Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand. Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. In particula the contract declares the method `calculateToBeUnlockNext(address recepient)`, however this function targets the sender of a transcation instead of the recipient. Misleading method names can lead to confusion, making the code more difficult to read and understand.

```
function calculateToBeUnlockNext(address recepient) public view
returns (uint256 toUnlockAmount) {...}
```

Recommendation

It's always a good practice for the contract to contain method names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	EDMA.sol#L263
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 setWhitelist(address user, bool isWhitelisted) external
onlyOwner {
whitelist[user] = isWhitelisted;
}
// set Presale address control
function setPresale(address presale) external onlyOwner {
preSaleAddress = presale;
whitelist[presale] = true;
}

// set Vesting Interval time (second based 1minute = 60);
function setvestingInterval(uint256 second) external onlyOwner {//!
validation
vestingInterval = second;
}
```

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.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	EDMA.sol#L341,357
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
require(sender != address(0), "EDMA: transfer from zero address");
require(recipient != address(0), "EDMA: transfer to zero address");
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

NWES - Nonconformity with ERC-20 Standard

Criticality	Minor / Informative
Location	EDMA.sol#L346
Status	Unresolved

Description

The contract is not fully conforming to the ERC20 Standard. Specifically, according to the standard, transfers of 0 values must be treated as normal transfers and fire the Transfer event. However the contract implements, a conditional check that prohibits transfers of 0 values.

This discrepancy between the contract's implementation and the ERC20 standard may lead to inconsistencies and incompatibilities with other contracts.

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

...
require(amount > 0, "EDMA: amount must not be zero");
...
}
```

Recommendation

The incorrect implementation of the ERC20 standard could potentially lead to problems when interacting with the contract, as other contracts or applications that expect the ERC20 interface may not behave as expected. The team is advised to review and revise the implementation of the transfer mechanism to ensure full compliance with the ERC20 standard. https://eips.ethereum.org/EIPS/eip-20.

PGA - Potential Griefing Attack

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

Description

The contract includes functionality designed to enforce specific conditions on transactions. However, this design is vulnerable to griefing attacks, where malicious actors can exploit the contract's logic to interfere with legitimate user operations.

In this case, the contract enforces transactional limits based on on-chain activity. Specifically, the `setPresale` method can be called by the owner to update the `preSaleAddress` . If the `preSaleAddress` transfers tokens to an externally owned account (EOA) using the `transferAndVest` function, the `lastReleasedTime` object is updated for that address. This prevents the user from transferring tokens until the vesting interval has passed.

```
function transferAndVest(address recipient, uint256 amount)
external onlyPreSale returns (bool) {
...
} else {...
vesting[recipient].lastRelasedTime =  block.timestamp;
...
}
```

Recommendation

The team is advised to review the transfer mechanism to ensure that all legitimate operations are processed as intended. This will help maintain the integrity of user activities and strengthen trust in the system.

PSU - Potential Subtraction Underflow

Criticality	Minor / Informative
Location	EDMA.sol#L244
Status	Unresolved

Description

The contract subtracts two values, the second value may be greater than the first value if the contract owner misuses the configuration. As a result, the subtraction may underflow and cause the execution to revert.

```
if(vestinDetails.totalLocked == 0 || vestinDetails.totalLocked -
vestinDetails.totalReleased <= 0) {
return 0;
}
```

Recommendation

The team is advised to properly handle the code to avoid underflow subtractions and ensure the reliability and safety of the contract. The contract should ensure that the first value is always greater than the second value. It should add a sanity check in the setters of the variable or not allow executing the corresponding section if the condition is violated.

RCS - Redundant Conditional Statements

Criticality	Minor / Informative
Location	EDMA.sol#L248
Status	Unresolved

Description

The contract contains redundant conditional statements that can be simplified to improve code efficiency and performance. Conditional statements that merely return the result of an expression are unnecessary and lead to larger code size, increased memory usage, and slower execution times. By directly returning the result of the expression, the code can be made more concise and efficient, reducing gas costs and improving runtime performance. Such redundancies are common when simple comparisons or checks are performed within conditional statements, leading to redundant operations. In this case, `intervalPassed` is a `uint256` variable and its scope will always be `>=0` .

```
if(intervalPassed <= 0) {
...
}
```

Recommendation

It is recommended to refactor conditional statements that return results by eliminating unnecessary code structures and directly returning the outcome of the expression. This practice minimizes the number of operations required, reduces the code footprint, and optimizes memory and gas usage. Simplifying such statements makes the code more readable and improves its overall performance.

RTR - Release Time Reset

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

Description

During instances where an address has already received funds from the presale address, the transferAndVest function updates the `lastReleasedTime` for that user, effectively resetting the interval period for the entire balance, including portions received in previous transfers. As a result the user's balance may remain vested longer than expected.

```
function transferAndVest(address recipient, uint256 amount)
external onlyPreSale returns (bool) {
} else {
vesting[recipient].totalLocked =  vesting[recipient].totalLocked +
amount;
vesting[recipient].lastRelasedTime =  block.timestamp;
}
...
}
```

Recommendation

To enhance user experience, it is advisable to ensure that funds are released consistently across predetermined vesting periods. Funds received at different time instances should not affect the vesting of past deposits.

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 {
require(sender != address(0), "EDMA: transfer from zero address");
require(recipient != address(0), "EDMA: transfer to zero address");
require(amount > 0, "EDMA: amount must not be zero");

if (!whitelist[sender]) {
require(tradingEnabled, "EDMA: trading is not enabled");
}

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.

UVI - Unrestricted Voting Interval

Criticality	Minor / Informative
Location	EDMA.sol#L277
Status	Unresolved

Description

The contract implements the setvestingInterval to update the `vestingInterval` variable. If this variable is set to a high value, users may not be able to transfer their tokens for an indefinite time.

```
function setvestingInterval(uint256 second) external onlyOwner {
vestingInterval = second;
}
```

Recommendation

It is advisable to ensure the vesting interval does not exceed a reasonable value to ensure consistency of operations and enhance users' trust.

ZD - Zero Division

Criticality	Minor / Informative
Location	EDMA.sol#L246
Status	Unresolved

Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert.

```
uint256 intervalPassed = (block.timestamp -
vestinDetails.lastRelasedTime) / vestingInterval;
```

Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero, or should not allow the execution of the corresponding statements.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	EDMA.sol#L137,138,139
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.

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

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.

L06 - Missing Events Access Control

Criticality	Minor / Informative
Location	EDMA.sol#L268
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. There are functions that have no event emitted, so it is difficult to track off-chain changes.

```
preSaleAddress = presale
```

Recommendation

To avoid this issue, 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.

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	EDMA.sol#L274
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.

```
vestingInterval = second
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	EDMA.sol#L245,251
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 -
vestinDetails.lastRelasedTime) / vestingInterval
uint256 toRelease  = (vestinDetails.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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	EDMA.sol#L268,317
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.

```
preSaleAddress = presale
payable(recipient).transfer(balnce)
```

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.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	EDMA.sol#L6,86
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	EDMA.sol#L86
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

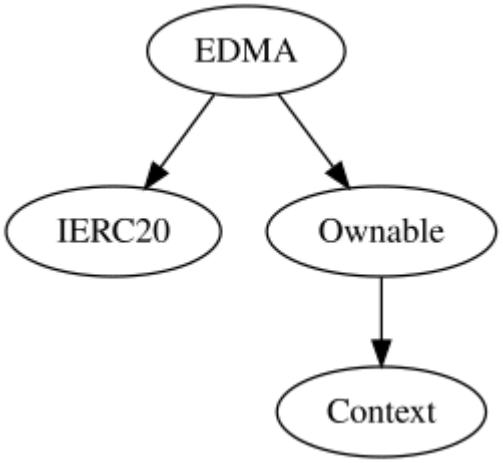
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest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

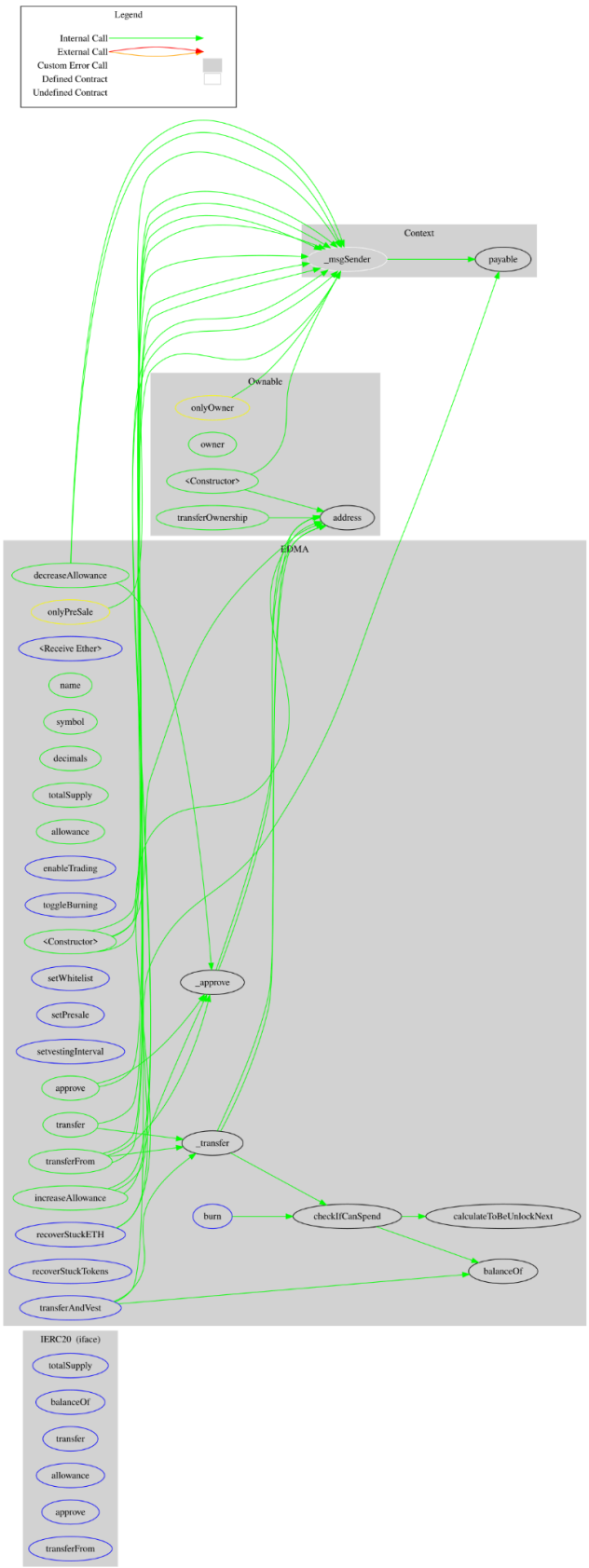
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Context	Implementation			
	_msgSender	Internal		
Ownable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

	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	allowance	Public		-
	burn	External	✓	-
	enableTrading	External	✓	onlyOwner
	toggleBurning	External	✓	onlyOwner
	transferAndVest	External	✓	onlyPreSale
	calculateToBeUnlockNext	Public		-
	setWhitelist	External	✓	onlyOwner
	setPresale	External	✓	onlyOwner
	setvestingInterval	External	✓	onlyOwner
	approve	Public	✓	-
	transfer	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	recoverStuckTokens	External	✓	onlyOwner
	recoverStuckETH	External	✓	onlyOwner
	checkIfCanSpend	Internal	✓	
	_transfer	Internal	✓	
	_approve	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
Context
_msgSender
payable
Ownable
onlyOwner
owner
<Constructor>
transferOwnership
address
EDMA
decreaseAllowance
onlyPreSale
<Receive Ether>
name
symbol
decimals
totalSupply
allowance
enableTrading
toggleBurning
<Constructor>
setWhitelist
setPresale
setvestingInterval
approve
transfer
transferFrom
increaseAllowance
recoverStuckETH
recoverStuckTokens
transferAndVest
_approve
_transfer
burn
checkIfCanSpend
calculateToBeUnlockNext
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. There are some functions that can be abused by the owner like stop transactions. 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

cyberscope.io