Cyberscope

Audit Report

Barin Mineral Token

May 2025

Files :

9708f423745d99492822331d38f95fe86b5c86c12e174ef4efdf4cead10c6b5e

3af682a1d1ddd8ed23426e0fd38dcbb7e4b492964e6a75ee91fa203849719bca

Audited by © cyberscope

Table of Contents

Table of Contents **1**
Risk Classification **3**
Review **4**
Audit Updates 4
Source Files 4
Overview **5**
Barin Mineral Token (BARIN) 5
Gasless Approvals with EIP-2612 5
Token Minting 5
Functionality 5
Inherited Functionality 5
BarinVesting Contract 6
Vesting Schedule Creation 6
Token Release 6
Revocation of Vesting Schedules 7
Token Vesting Calculation 7
Emergency Withdrawals 7
Functionality 7
Events 7
Errors 8
Constructor 8
Functions 8
View Functions 9
Findings Breakdown **10**
Diagnostics **11**
OVB - Omitted Vested Balance 12
Description 12
Recommendation 12
DVI - Duplicate Vesting Identifier 13
Description 13
Recommendation 13
CCR - Contract Centralization Risk 14
Description 14
Recommendation 15
EFO - External Function Optimization 16
Description 16
Recommendation 16
MEM - Misleading Error Message 17
Description 17

Recommendation 17
MC - Missing Check 18
Description 18
Recommendation 19
MEE - Missing Events Emission 20
Description 20
Recommendation 20
IOV - Inconsistent Openzeppelin Versioning 21
Description 21
Recommendation 21
PSU - Potential Subtraction Underflow 22
Description 22
Recommendation 23
RED - Redundant Error Declaration 24
Description 24
Recommendation 24
TSI - Tokens Sufficiency Insurance 25
Description 25
Recommendation 25
TCT - Transfer Contract Tokens 26
Description 26
Recommendation 26
L19 - Stable Compiler Version 27
Description 27
Recommendation 27
L20 - Succeeded Transfer Check 28
Description 28
Recommendation 28
Functions Analysis 29
Inheritance Graph 30
Flow Graph 31
Summary 32
Disclaimer 33
About Cyberscope 34

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

Audit Updates

Initial Audit	29 May 2025

Source Files

Filename	SHA256
BarinToken.sol	9708f423745d99492822331d38f95fe86b5c86c12e174ef4efdf4cead10c6b5e
BarinVesting.sol	3af682a1d1ddd8ed23426e0fd38dcbb7e4b492964e6a75ee91fa203849719bca

Overview

Barin Mineral Token (BARIN)

The `Barin` contract represents the Barin Mineral Token (BARIN), an ERC20 token that incorporates support for EIP-2612, enabling gasless approvals via the `ERC20Permit` extension. The token follows the standard ERC20 behavior and allows for efficient, secure transfers and approvals. The contract is built on OpenZeppelin's ERC20 and ERC20Permit contracts to provide industry-standard functionality while enabling users to approve token transfers without incurring gas fees for approval transactions.

Gasless Approvals with EIP-2612

By inheriting from `ERC20Permit`, the `Barin` contract supports the EIP-2612 standard for gasless approvals. This allows users to approve token transfers without needing to send a transaction on the blockchain, saving gas fees for approval actions. The `ERC20Permit` extension uses a signature-based method for granting approval, enabling users to sign off-chain messages to approve spending by other addresses.

Token Minting

Upon deployment, the contract mints a total supply of 1 billion `BARIN` tokens (1,000,000,000) and assigns the entire supply to the contract deployer's address. This ensures that the deployer has full control of the initial token balance.

The token supply is minted with the following formula:

- **Initial Mint**: `1,000,000,000 * 10^decimals()` tokens.

Functionality

Inherited Functionality

The contract inherits and extends the functionality of the following OpenZeppelin contracts:

- **`ERC20`**: Implements the ERC20 token standard, allowing for basic token functionality such as transferring, approving, and querying balances and allowances.

- `ERC20Permit`: Adds support for EIP-2612, enabling gasless approvals via permit signatures.

BarinVesting Contract

The `BarinVesting` contract is designed to manage multiple vesting schedules for the BARIN token. It provides functionality for creating and managing vesting schedules, releasing vested tokens, and revoking unvested portions, if allowed. The contract is flexible and enables the owner to configure various parameters. It ensures the gradual release of tokens according to predefined schedules. The contract utilizes OpenZeppelin's `Ownable` and `ReentrancyGuard` to enhance security and prevent reentrancy attacks.

Vesting Schedule Creation

The `createVestingSchedule` function allows the contract owner to create a single vesting schedule for a beneficiary. This function requires specifying the total amount of tokens to be vested, the cliff duration, the vesting duration, and whether the schedule is revocable. Additionally, the `batchCreateSchedules` function allows for batch creation of multiple vesting schedules for several beneficiaries in a single transaction, improving gas efficiency.

Each vesting schedule consists of the following key elements:

- **Cliff Duration**: The period before any tokens are released.
- **Vesting Duration**: The total period over which tokens are gradually released.
- **Revocability**: Determines whether the contract owner can revoke the vesting schedule and reclaim unvested tokens.

When a vesting schedule is created, an event is emitted, providing transparency and tracking.

Token Release

Beneficiaries can claim their vested tokens after the cliff duration has passed. The `release` function calculates how much can be claimed based on the elapsed time since the vesting started. It ensures that only the beneficiary or the contract owner can release the tokens. If no tokens are available for release, the contract throws an error.

The `batchRelease` function is an efficient method that allows the contract owner to release tokens for multiple beneficiaries in one transaction.

Revocation of Vesting Schedules

The `revoke` function allows the contract owner to revoke a vesting schedule, provided that it is marked as revocable. When a schedule is revoked, the unvested tokens are returned to the contract owner. Once a schedule is revoked, it becomes non-revocable, and no further action can be taken. The contract ensures that only unvested tokens are returned to the owner.

Token Vesting Calculation

The vesting process follows a linear model after the cliff period. Tokens are gradually released over the total vesting duration. The amount of tokens released at any given time is proportional to the elapsed time. If the cliff period hasn't passed, no tokens are released. Once the cliff period is over, the tokens are released linearly over the remaining vesting duration.

Emergency Withdrawals

The contract owner has the ability to withdraw tokens that were accidentally sent to the contract address. This ensures that the owner retains control over the contract's balance in unforeseen situations.

Functionality

Events

The contract emits several events for transparency and tracking. These events include:

- **`VestingScheduleCreated`** : Emitted when a new vesting schedule is created, showing the beneficiary and the total amount of tokens.
- **`TokensReleased`** : Emitted when tokens are released to a beneficiary, indicating the amount released.
- **`VestingRevoked`** : Emitted when a vesting schedule is revoked, detailing the amount of unvested tokens returned to the owner.

Errors

The contract uses custom error handling to ensure smooth operation and prevent incorrect actions. These errors include:

- `InvalidSchedule` : Thrown if a vesting schedule is invalid (e.g., missing beneficiary or insufficient balance).
- `CliffNotReached`
- `NothingToRelease` : Thrown if there are no tokens available for release.
- `NotBeneficiary` : Thrown if someone other than the beneficiary tries to release tokens.
- `ScheduleNotRevocable` : Thrown if an attempt is made to revoke a non-revocable schedule.
- `InsufficientBalance` : Thrown if there are not enough tokens in the contract to create a new vesting schedule.

Constructor

The constructor initializes the contract with two parameters:

- `_token` : The address of the BARIN token contract.
- `_vestingStart` : The start time for vesting, which defaults to the current block timestamp if not provided.

Functions

The contract provides several functions to manage vesting schedules:

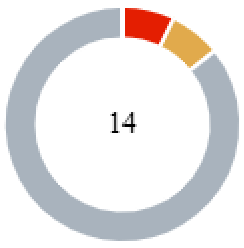
- `createVestingSchedule` : Creates a vesting schedule for a single beneficiary.
- `batchCreateSchedules` : Allows the contract owner to create multiple vesting schedules in a single transaction.
- `release` : Releases vested tokens for a specific schedule.
- `batchRelease` : Allows the contract owner to release tokens for multiple schedules at once.
- `revoke` : Revokes a vesting schedule and returns unvested tokens to the owner.
- `emergencyWithdraw` : Enables the owner to recover tokens that were accidentally sent to the contract.

View Functions

The contract also provides several view functions to retrieve schedule details:

- `getSchedule` : Returns the details of a specific vesting schedule.
- `releasableAmount` : Returns the amount of tokens that can be released from a specific schedule.
- `vestedAmount` : Returns the total vested amount for a specific schedule.
- `getBeneficiarySchedules` : Returns all vesting schedules for a specific beneficiary.

Findings Breakdown



●	Critical	1
●	Medium	1
●	Minor / Informative	12

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	1	0	0	0
●	Medium	1	0	0	0
●	Minor / Informative	12	0	0	0

Diagnostics

🔴 Critical 🟠 Medium ⚪ Minor / Informative

Severity	Code	Description	Status
🔴	OVB	Omitted Vested Balance	Unresolved
🟠	DVI	Duplicate Vesting Identifier	Unresolved
⚪	CCR	Contract Centralization Risk	Unresolved
⚪	EFO	External Function Optimization	Unresolved
⚪	MEM	Misleading Error Message	Unresolved
⚪	MC	Missing Check	Unresolved
⚪	MEE	Missing Events Emission	Unresolved
⚪	IOV	Inconsistent OpenZeppelin Versioning	Unresolved
⚪	PSU	Potential Subtraction Underflow	Unresolved
⚪	RED	Redundant Error Declaration	Unresolved
⚪	TSI	Tokens Sufficiency Insurance	Unresolved
⚪	TCT	Transfer Contract Tokens	Unresolved
⚪	L19	Stable Compiler Version	Unresolved
⚪	L20	Succeeded Transfer Check	Unresolved

OVB - Omitted Vested Balance

Criticality	Critical
Location	BarinVesting.sol#L104
Status	Unresolved

Description

`_createVestingSchedule` can be used to create new vesting schedules for beneficiaries. The function checks if the current balance of the contract is enough to cover the amount added but it does not account for the tokens already vested. This may create situations where the contract will not have enough tokens to cover all the vested amounts. Additionally, in the case of `batchCreateSchedules` each amount is calculated separately and does not account for the total amount vested for all the schedules to be created.

```
function _createVestingSchedule(
    address beneficiary,
    uint256 totalAmount,
    uint256 cliffDuration,
    uint256 vestingDuration,
    bool revocable
) internal returns (bytes32 scheduleId) {
    if (beneficiary == address(0) || totalAmount == 0 ||
vestingDuration == 0) {
        revert InvalidSchedule();
    }

    if (token.balanceOf(address(this)) < totalAmount) {
        revert InsufficientBalance();
    }
```

Recommendation

It is recommended that the team keeps track of the vested token amounts and account them in their calculations to ensure that the contract will always have enough balance to cover the created schedules.

DVI - Duplicate Vesting Identifier

Criticality	Medium
Location	BarinVesting.sol#L119
Status	Unresolved

Description

In `_createVestingSchedule` the identifier of a vesting schedule is created by using the beneficiary, the current timestamp and the amount to be vested. However the function can be called multiple times in the same block. In this case adding the same beneficiary and total amount will create the same `scheduleId` that will overwrite the existing vesting schedule and will be pushed as a duplicate in the beneficiary's schedules.

```
scheduleId = keccak256(abi.encodePacked(beneficiary,
block.timestamp, totalAmount));
_schedules[scheduleId] = VestingSchedule({
    totalAmount: totalAmount,
    cliffDuration: cliffDuration,
    vestingDuration: vestingDuration,
    startTime: vestingStart,
    withdrawnAmount: 0,
    beneficiary: beneficiary,
    revocable: revocable
});
_beneficiarySchedules[beneficiary].push(scheduleId);
```

Recommendation

It is recommended that the team adds an extra unique parameter for the creation of the schedule id to ensure that each id will be different.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	BarinVesting.sol#L62,75,139,161,172,242
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 createVestingSchedule(address beneficiary, uint256
totalAmount,  uint256 cliffDuration, uint256 vestingDuration,
bool revocable) external onlyOwner
function batchCreateSchedules(address[] calldata beneficiaries,
uint256[] calldata amounts, uint256[] calldata cliffDurations,
uint256[] calldata vestingDurations, bool[] calldata
revocables) external onlyOwner
function release(bytes32 scheduleId) external nonReentrant {
    if (schedule.beneficiary != msg.sender && msg.sender !=
owner()) {
        revert NotBeneficiary();
    }
}
function batchRelease(bytes32[] calldata scheduleIds) external
function revoke(bytes32 scheduleId) external onlyOwner
function emergencyWithdraw(address tokenAddress, uint256

amount) external onlyOwner
```

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.

EFO - External Function Optimization

Criticality	Minor / Informative
Location	BarinVesting.sol#L139,164
Status	Unresolved

Description

`batchRelease` calls the `release` function multiple times. The keyword `this` is used to call it since the function is declared as external. The team could instead declare the `release` function as public.

```
function release(bytes32 scheduleId) external nonReentrant
...
this.release(scheduleIds[i]);
```

Recommendation

It is recommended that the team declares the `release` function as public to increase the `batchRelease` s` efficiency.

MEM - Misleading Error Message

Criticality	Minor / Informative
Location	BarinVesting.sol#L143
Status	Unresolved

Description

Errors can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some error names that are too generic or do not clearly convey the underneath functionality.

Specifically, in `release` function the revert message only mentions that the `msg.sender` is not the beneficiary but does not mention the owner.

```
if (schedule.beneficiary != msg.sender && msg.sender !=
owner()) {
    revert NotBeneficiary();
}
```

Recommendation

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

MC - Missing Check

Criticality	Minor / Informative
Location	BarinVesting.sol#L51,121,139,161,172
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

In the constructor, a check is missing to ensure that `_vestingStart` is greater than the current timestamp.

```
vestingStart = _vestingStart > 0 ? _vestingStart :
block.timestamp;
```

In `_createVestingSchedule` a check is missing to ensure that the `vestingDuration` is greater than `cliffDuration` and both are within reasonable amounts.

```
_schedules[scheduleId] = VestingSchedule({
    totalAmount: totalAmount,
    cliffDuration: cliffDuration,
    vestingDuration: vestingDuration,
    startTime: vestingStart,
    withdrawnAmount: 0,
    beneficiary: beneficiary,
    revocable: revocable
});
```

In `release` , `batchRelease` , `revoke` checks are missing to ensure that `scheduleIds` are valid.

```
function release(bytes32 scheduleId) external nonReentrant
function batchRelease(bytes32[] calldata scheduleIds) external
function revoke(bytes32 scheduleId) external onlyOwner
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	BarinVesting.sol#L242
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 emergencyWithdraw(address tokenAddress, uint256
amount) external onlyOwner {
    IERC20(tokenAddress).transfer(owner(), amount);
}
```

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.

IOV - Inconsistent Openzeppelin Versioning

Criticality	Minor / Informative
Location	BarinVesting.sol#L8
Status	Unresolved

Description

The contract utilizes the OpenZeppelin library to import components such `Ownable` and `ReentrancyGuard` extensions. The contract is build using the version v5 of OZ; however it additionally imports components from the v4 version. Specifically, from v5 and above the `ReentrancyGuard` has been moved to `utils` folder while the contract searches the `security` folder which was the case on v4. These inconsistencies, in using a mixed OpenZeppelin version could lead to issues such as compilation errors.

```
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
```

Recommendation

It is recommended that the team uses one version of external libraries like openzeppelin to ensure combatability.

PSU - Potential Subtraction Underflow

Criticality	Minor / Informative
Location	BarinVesting.sol#L182,194,219
Status	Unresolved

Description

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

Specifically, if the owner uses the `revoke` function on a schedule, it will change the schedule's `totalAmount` to the vested amount. If the beneficiary has already claimed an amount this may create a time window where the vested amount will not be greater than the `withdrawnAmount` resulting in an underflow during the calculation of the releasable amount.

```
function revoke(bytes32 scheduleId) external onlyOwner {
    //..
    uint256 vested = _vestedAmount(scheduleId);
    //...
    schedule.totalAmount = vested;
    //...
}

function _releasableAmount(bytes32 scheduleId) private view
returns (uint256) {
    return _vestedAmount(scheduleId) -
_schedules[scheduleId].withdrawnAmount;
}

function _vestedAmount(bytes32 scheduleId) private view returns
(uint256) {
    //...
    return (schedule.totalAmount * elapsed) /
schedule.vestingDuration;
}
```

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.

RED - Redundant Error Declaration

Criticality	Minor / Informative
Location	BarinVesting.sol#L41
Status	Unresolved

Description

Errors are declared in the contract that are not used within the contract's functions. As a result, these declared errors are redundant and serve no purpose within the contract's current implementation.

```
error CliffNotReached();
```

Recommendation

To optimize contract performance and efficiency, it is advisable to regularly review and refactor the codebase, removing the unused errors declarations. This proactive approach not only streamlines the contract, reducing deployment and execution costs but also enhances readability and maintainability.

TSI - Tokens Sufficiency Insurance

Criticality	Minor / Informative
Location	BarinVesting.sol#L242
Status	Unresolved

Description

The contract is designed to receive the tokens externally. While external administration can provide flexibility, it introduces a dependency on the administrator's actions, which can lead to various issues and centralization risks. Additionally tokens from the contract can be withdrawn from the owner which may result in beneficiaries not being able to release their tokens.

```
function createVestingSchedule
...
function batchCreateSchedules
...
function release
...
function batchRelease
...
function emergencyWithdraw(address tokenAddress, uint256
amount) external onlyOwner {
    IERC20(tokenAddress).transfer(owner(), amount);
}
```

Recommendation

It is recommended to consider implementing a more decentralized and automated approach for handling the contract tokens. One possible solution could be to transfer the tokens in the contract during its initialization or during the creation of the vesting schedules. Additionally the team could consider the amount of tokens vested in the `emergencyWithdraw` function.

TCT - Transfer Contract Tokens

Criticality	Minor / Informative
Location	BarinVesting.sol#L242
Status	Unresolved

Description

The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the `emergencyWithdraw` function.

```
function emergencyWithdraw(address tokenAddress, uint256
amount) external onlyOwner {
    IERC20(tokenAddress).transfer(owner(), amount);
}
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	BarinVesting.sol#L2 BarinToken.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 ^0.8.20;
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	BarinVesting.sol#L153,186,243
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
token.transfer(schedule.beneficiary, releasable);
token.transfer(owner(), unvestedAmount);
IERC20(tokenAddress).transfer(owner(), amount);
```

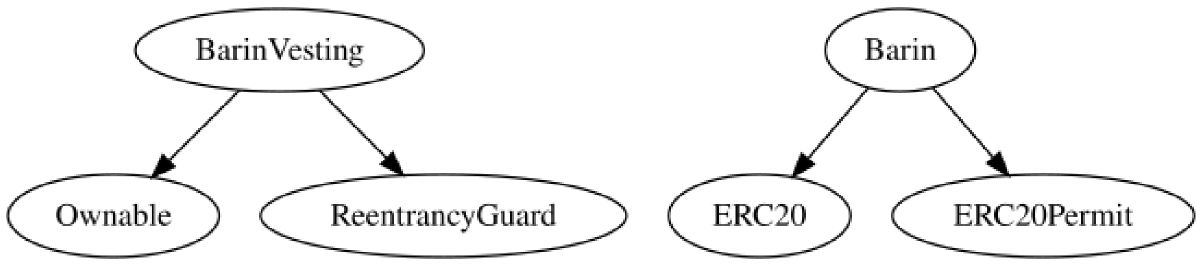
Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

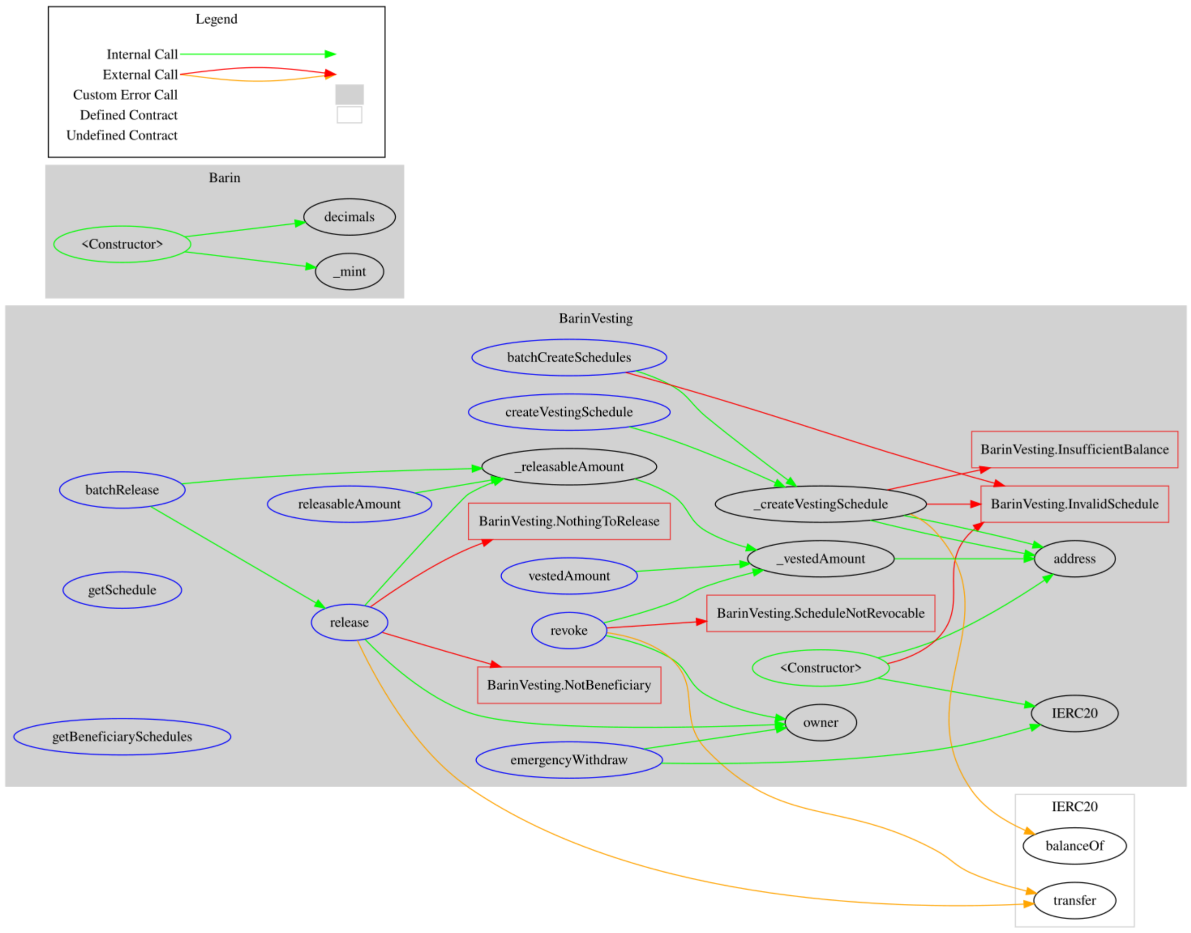
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
BarinVesting	Implementation	Ownable, ReentrancyGuard		
		Public	✓	-
	createVestingSchedule	External	✓	onlyOwner
	batchCreateSchedules	External	✓	onlyOwner
	_createVestingSchedule	Internal	✓	
	release	External	✓	nonReentrant
	batchRelease	External	✓	-
	revoke	External	✓	onlyOwner
	_releasableAmount	Private		
	_vestedAmount	Private		
	getSchedule	External		-
	releasableAmount	External		-
	vestedAmount	External		-
	getBeneficiarySchedules	External		-
	emergencyWithdraw	External	✓	onlyOwner
Barin	Implementation	ERC20, ERC20Permit		
		Public	✓	ERC20 ERC20Permit

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
Barin
<Constructor>
decimals
_mint
BarinVesting
batchCreateSchedules
createVestingSchedule
_releasableAmount
batchRelease
releasableAmount
BarinVesting.NothingToRelease
_createVestingSchedule
BarinVesting.InsufficientBalance
BarinVesting.InvalidSchedule
getSchedule
vestedAmount
_vestedAmount
address
release
revoke
BarinVesting.ScheduleNotRevocable
<Constructor>
BarinVesting.NotBeneficiary
owner
IERC20
getBeneficiarySchedules
emergencyWithdraw
IERC20
balanceOf
transfer

Summary

Barin Mineral Token contract implements a token and vesting mechanism. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

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