BEOSIN
Blockchain Security

WeatherXM

Smart Contract Security Audit

No. 202405061825

May 6th, 2024

SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

Summary of Audit Result

After auditing, 4 Info items were identified in the WeatherXM. Specific audit details will be presented in the Findings section. Users should pay attention to the following aspects when interacting with this project:

Info	
	Fixed : 4 Acknowledged : 0

Basic Token Information for WeatherXM

Token name	WeatherXM
Token symbol	WXM
Decimals	18
Total supply	100 million(burnable)
Token type	ERC-20

Table 1 WeatherXM token info

Business overview

This audited three contracts of the WeatherXM project: WeatherXM, RewardsVault and RewardPool.

WeatherXM is an ERC-20 token deployed on the Ethereum, with a total supply of 100 million tokens (non-mintable and burnable).

The RewardsVault contract is responsible for sending daily emissions to the rewards distributor, the RewardPool contract. The maximum daily emission is 14246 WXM tokens.

The RewardPool contract is mainly used for users to claim rewards. Users can claim their rewards by providing information such as cycle and proof. Except for claiming rewards for themselves, Anyone can claim rewards on behalf of a user by providing a signature from that user.

1 Overview

1.1 Project Overview

Project Name	WeatherXM
Project Language	Solidity
Platform	Ethereum, Arbitrum
Code base	https://github.com/weatherxm-network/smart-contracts/tree/develop https://github.com/weatherxm-network/smart-contracts/tree/fix/audit-suggestions
Audit scope	WeatherXM.sol, RewardsVault.sol, RewardPool.sol
Commit	e8de2a895b9b8ae8e203031572388f60bcffd6d8 c8d2c44062ca8b0d599e8028be853b46f37142ef

1.2 Audit Overview

Audit work duration: Apr 26, 2024 – May 6, 2024

Audit team: Beosin Security Team

1.3 Audit Method

The audit methods are as follows:

1. Formal Verification

Formal verification is a technique that uses property-based approaches for testing and verification. Property specifications define a set of rules using Beosin's library of security expert rules. These rules call into the contracts under analysis and make various assertions about their behavior. The rules of the specification play a crucial role in the analysis. If the rule is violated, a concrete test case is provided to demonstrate the violation.

2. Manual Review

Using manual auditing methods, the code is read line by line to identify potential security issues. This ensures that the contract's execution logic aligns with the client's specifications and intentions, thereby safeguarding the accuracy of the contract's business logic.

The manual audit is divided into three groups to cover the entire auditing process:

The Basic Testing Group is primarily responsible for interpreting the project's code and conducting comprehensive functional testing.

The Simulated Attack Group is responsible for analyzing the audited project based on the collected historical audit vulnerability database and security incident attack models. They identify potential attack vectors and collaborate with the Basic Testing Group to conduct simulated attack tests.

The Expert Analysis Group is responsible for analyzing the overall project design, interactions with third parties, and security risks in the on-chain operational environment. They also conduct a review of the entire audit findings.

3. Static Analysis

Static analysis is a method of examining code during compilation or static analysis to detect issues. Beosin-VaaS can detect more than 100 common smart contract vulnerabilities through static analysis, such as reentrancy and block parameter dependency. It allows early and efficient discovery of problems to improve code quality and security.

2 Findings

Index	Risk description	Severity level	Status
WeatherXM-01	Missing trigger event	Info	**Fixed**
WeatherXM-02	Redundant code	Info	**Partially Fixed**
WeatherXM-03	Centralization risk	Info	**Fixed**
WeatherXM-04	The _verify function does not need to have a return value	Info	**Fixed**

[WeatherXM-01] Missing trigger event

Severity Level	Info
Type	General Vulnerability
Lines	RewardsVault.sol#L90-92
Description	When the critical variable `rewardDistributor` is modified, the corresponding event is not triggered. `function setRewardDistributor(address _rewardDistributor) public onlyOwner { rewardDistributor = _rewardDistributor; }`
Recommendation	It is recommended to emit events when modifying critical variables as it provides a standardized way to capture and communicate important changes within the contract. Events enable transparency and allow external systems and users to easily track and react to these modifications.
Status	**Fixed.** `function setRewardDistributor(address _rewardDistributor) public onlyOwner { rewardDistributor = _rewardDistributor; emit DistributorUpdated(_rewardDistributor); }`

[WeatherXM-02] Redundant code

Severity Level	Info
Type	Coding Conventions
Lines	RewardPool.sol#L82-90;WeatherXM.sol#L16
Description	`validDestination` modifier and `TokenTransferWhilePaused` error are not used in the code,they are redundant code. `modifier validDestination(address to) { if (to == address(0x0)) { revert TargetAddressIsZero(); } if (to == address(this)) { revert TargetAddressIsContractAddress(); } _; } error TokenTransferWhilePaused();`
Recommendation	It is recommended to delete redundant code.
Status	**Partially Fixed.** The code of `validDestination` modifier has been deleted, but the code of `TokenTransferWhilePaused` error has not been deleted.

[WeatherXM-03] Centralization risk

Severity Level	Info
Type	Business Security
Lines	WeatherXM.sol#L18-20
Description	The project distributes all tokens to EOA accounts during deployment, which poses a certain centralization risk. `constructor(string memory _name, string memory _symbol) ERC20(_name, _symbol) ERC20Capped(maxSupply) { _mint(_msgSender(), maxSupply); }`
Recommendation	It is recommended to use a multi-signature wallet for token management.
Status	**Fixed.** The project team plans to use a multi-signature wallet for token management.

[WeatherXM-04] The _verify function does not need to have a return value

Severity Level	Info
Type	Business Security
Lines	RewardPool.sol#L211-220,187-202
Description	The `_verify` function is used only as a verification function. It does not modify the value of the input parameter `amount`, so there is no need to return the value of `amount`. In the `_allocatedRewardsForProofMinusRewarded` function that calls the `_verify` function, use the `amount` variable directly without creating a new variable `total`.

```
  function _verify(
    address account,
    uint256 amount,
    uint256 _cycle,
    bytes32[] calldata proof
  ) internal view returns (uint256) {
    bytes32 leaf =
keccak256(bytes.concat(keccak256(abi.encode(account, amount))));
    require(MerkleProof.verify(proof, roots[_cycle], leaf), "INVALID
PROOF");
    return amount;
  }
  function _allocatedRewardsForProofMinusRewarded(
    address account,
    uint256 amount,
    uint256 _cycle,
    bytes32[] calldata proof
  ) internal view returns (uint256) {
    if (amount == 0) {
      revert AmountRequestedIsZero();
    }
    uint256 total = _verify(account, amount, _cycle, proof);
    if (claims[account] < total) {
      return total.sub(claims[account]);
    } else {
      return 0;
```

```
    }
  }
```

Recommendation

It is recommended to modify the `_verify` function to a function with no return value, and use the variable `amount` directly in the `_allocatedRewardsForProofMinusRewarded` function without creating a `total` variable.

Status

Fixed.

```
  function _verify(address account, uint256 amount, uint256 _cycle,
bytes32[] calldata proof) internal view {
    bytes32 leaf =
keccak256(bytes.concat(keccak256(abi.encode(account, amount))));
    require(MerkleProof.verify(proof, roots[_cycle], leaf), "INVALID
PROOF");
  }
  function _allocatedRewardsForProofMinusRewarded(
    address account,
    uint256 amount,
    uint256 _cycle,
    bytes32[] calldata proof
  ) internal view returns (uint256) {
    if (amount == 0) {
      revert AmountRequestedIsZero();
    }
    _verify(account, amount, _cycle, proof);
    if (claims[account] < amount) {
      return amount.sub(claims[account]);
    } else {
      return 0;
    }
  }
```

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1(Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Likelihood \ Impact	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	Medium	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.3 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.4 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Compiler Version Security
		Deprecated Items
		Redundant Code
		require/assert Usage
		Gas Consumption
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		call/delegatecall Security
		Returned Value Security
		tx.origin Usage
		Replay Attack
		Overriding Variables
		Third-party Protocol Interface Consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Asset Tradability
		Arbitrage Attack

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in blockchain.

3.4 About Beosin

Beosin is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions. Beosin has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, Beosin has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

BEOSIN
Blockchain Security

Official Website
https://www.beosin.com

Telegram
https://t.me/beosin

Twitter
https://twitter.com/Beosin_com

Email
service@beosin.com