BEOSIN
Blockchain Security

wandfi

Smart Contract Security Audit

No. 202403081621

Mar 8th, 2024

SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

Summary of Audit Results

After auditing, 2 Medium, 1 Low-risk and 1 Info items were identified in the wandfi project. Specific audit details will be presented in the Findings section. Users should pay attention to the following aspects when interacting with this project:

Medium	Fixed:2 Acknowledged: 0
Low	Fixed:1 Acknowledged: 0
Info	Fixed:1 Acknowledged: 0

- **Project Description:**

Business overview

In the wandfi project, it mainly consists of the following modules:

protocol module:

Owner can add a supported vault to the protocol contract and the permission to modify the configuration in the ProtocolSettings contract.

vault module:

Users can choose to mortgage the corresponding assetToken in the vault contract to obtain USB tokens and margin tokens (assetToken is stored in the TokenPot contract). Redeem the pledged assetToken by burning the USB and margin. Conversion between USB and margin tokens can also be done through the vault contract. margin is a standard ERC-20 token, and USB is a share token with standard ERC-20. The minting and burning of both tokens are controlled by the vault contract.

yield module:

vault corresponds to two PtyPool contracts

Users can stake assetToken in the PtyPoolSellHigh contract to obtain USB, and stake USB in the PtyPoolBuyLow contract to obtain assetToken. Rewards have nothing to do with the stake time. The increase in shares used to calculate rewards is generated when interacting with the vault contract (such as redeeming assetToken in the vault).

1 Overview

1.1 Project Overview

Project Name	wandfi
Project language	Solidity
Platform	Blast
Github Link	https://github.com/wandfi/wand-contract-blast
Commit hash	2c853b103c730136d6c3b534383a3c990dc47d66 d4c1386674325a742081da5ee8454070fc3465e3 9e9c9d62e5863153bb0c1c754b3130ef65970639

1.2 Audit Overview

Audit work duration: Feb 29, 2024 – Mar 8, 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
wandfi-01	Platform Token Lockup Risk	**Medium**	**Fixed**
wandfi-02	Centralization Risk	**Medium**	**Fixed**
wandfi-03	Repeatedly adding settings	**Low**	**Fixed**
wandfi-04	Redundant Code	**Info**	**Fixed**

Finding Details:

[wandfi-01] Platform Token Lockup Risk

Severity Level	Medium
Type	Business Security
Lines	PtyPool.sol#L65 & L129-140
Description	The `stake` function in the PtyPoolBuyLow and PtyPoolSellHigh contracts supports payable, but there is no corresponding function to extract the native token. The native token mistakenly sent to the PtyPool contract whose `_stakingToken` is not the native token will not be withdrawn.

```
  receive() external payable {}
 function stake(uint256 amount) external payable nonReentrant
onUserAction
    updateStakingYields(_msgSender())
updateMatchingYields(_msgSender()) updateTargetTokens(_msgSender())
{
    require(amount > 0, "Cannot stake 0");
    uint256 sharesAmount = _getStakingSharesByBalance(amount,
msg.value);
    _totalStakingShares = _totalStakingShares.add(sharesAmount);
    _userStakingShares[_msgSender()] =
_userStakingShares[_msgSender()].add(sharesAmount);
    TokensTransfer.transferTokens(_stakingToken, _msgSender(),
address(this), amount);
    emit Staked(_msgSender(), amount);
  }
```

Recommendation	It is recommended to add the function of extracting irrelevant tokens in the PtyPool contract, and determine whether it is a `stakingToken` or `targetToken` when withdrawing.
Status	**Fixed.**

```
 /**
  * Rescue tokens that are accidently sent to this contract
  */
 function rescue(address token, address recipient) external
nonReentrant onlyOwner {
   require(token != address(0) && recipient != address(0), "Zero address
```

```
detected");
    require(token != _stakingToken && token != _targetToken && token !=
_stakingYieldsToken && token != _matchingYieldsToken, "Cannot rescue
staking or yield tokens");
    uint256 amount;
    if (token == Constants.NATIVE_TOKEN) {
      amount = address(this).balance;
    }
    else {
      amount = IERC20(token).balanceOf(address(this));
    }
    require(amount > 0, "No tokens to rescue");
    TokensTransfer.transferTokens(token, address(this), recipient,
amount);
    emit TokenRescued(token, recipient, amount);
  }
```

[wandfi-02] Centralization Risk

Severity Level	**Medium**
Type	General Vulnerability
Lines	WandProtocol.sol #L66-75
Description	The owner in the project has the authority to modify relevant interactive contracts and basic parameters (such as the WandProtocol contract), and there is a risk of centralization.

```
  function setBlastPointsAddress(address _blastPointsAddress_, address
_blastPointsOperator_) external nonReentrant onlyOwner {
    // require(_blastPointsAddress_ != address(0) &&
_blastPointsOperator_ != address(0), "Zero address detected");
    _blastPointsAddress = _blastPointsAddress_;
    _blastPointsOperator = _blastPointsOperator_;
    for (uint256 i = 0; i < _vaults.length(); i++) {
      address vaultAddress = _vaults.at(i);
      IVault(vaultAddress).configureBlastPoints();
    }
  }
```

Recommendation	It is recommended to use multi-signature wallets and other methods to manage owner permissions.
Status	**Fixed.** The project states that https://safe.global will be used to create a multi-signature wallet for the owner account.

[wandfi-03] Repeatedly adding settings

Severity Level	Low
Type	Business Security
Lines	ProtocolSettings.sol #L115-127
Description	The `upsertParamConfig` function in the ProtocolSettings contract does not determine whether a new configuration already exists when adding a new configuration, which may result in duplicate additions.

```
  function upsertParamConfig(bytes32 param, uint256 defaultValue,
uint256 min, uint256 max) external nonReentrant onlyOwner {
    _upsertParamConfig(param, defaultValue, min, max);
  }
  function _upsertParamConfig(bytes32 param, uint256 defaultValue,
uint256 min, uint256 max) internal {
    require(param.length > 0, "Empty param name");
    require(min <= defaultValue && defaultValue <= max, "Invalid default
value");
    require(min <= max, "Invalid min and max");
    _paramsSet.add(param);
    _paramConfigs[param] = ParamConfig(defaultValue, min, max);
    emit UpsertParamConfig(param, defaultValue, min, max);
  }
```

Recommendation	It is recommended to add a judgment condition in `_upsertParamConfig` to check if the configuration to be added exists, and modify it if it exists instead of repeating the add.
Status	**Fixed.**

```
  function _upsertParamConfig(bytes32 param, uint256 defaultValue,
uint256 min, uint256 max) internal {
    require(param.length > 0, "Empty param name");
    require(min <= defaultValue && defaultValue <= max, "Invalid default
value");
    if (_paramsSet.contains(param)) {
      ParamConfig storage config = _paramConfigs[param];
      config.defaultValue = defaultValue;
      config.min = min;
      config.max = max;
    }
```

```
    else {
      _paramsSet.add(param);
      _paramConfigs[param] = ParamConfig(defaultValue, min, max);
    }
    emit UpsertParamConfig(param, defaultValue, min, max);
  }
```

[wandfi-04] Redundant Code

Severity Level	Info
Type	Coding Conventions
Lines	PtyPoolBuyLow .sol #L64-79 PtyPoolSellHigh.sol #L53-69 ProtocolSettings.sol #L119-127
Description	Regarding the mode of operation of PtyPoolBuyLow and PtyPoolSellHigh, for the `notifySellHighTriggered` and `notifySellHighTriggered` functions. It can be removed directly on the unsupported side.

```
  function notifyBuyLowTriggered(uint256 assetAmountAdded) external
override nonReentrant updateTargetTokens(address(0)) onlyVault {
    require(_vault.vaultMode() ==
Constants.VaultMode.AdjustmentBelowAARS, "Vault not in adjustment below
AARS mode");
    _targetTokensPerShare =
_targetTokensPerShare.add(assetAmountAdded.mul(1e18).div(_totalStaki
ngShares));
    emit MatchedTokensAdded(assetAmountAdded);
    if (_accruedMatchingYields > 0) {
      _matchingYieldsPerShare =
_matchingYieldsPerShare.add(_accruedMatchingYields.mul(1e18).div(_to
talStakingShares));
      _accruedMatchingYields = 0;
    }
  }
  function notifySellHighTriggered(uint256, uint256) external override
pure {
    revert("PtyPoolBuyLow: notifySellHighTriggered not allowed");
  }
```

The determination of min <= max in the ProtocolSettings contract is redundant code, and both have already been compared to `defaultValue`.

```
  function _upsertParamConfig(bytes32 param, uint256 defaultValue,
uint256 min, uint256 max) internal {
    require(param.length > 0, "Empty param name");
    require(min <= defaultValue && defaultValue <= max, "Invalid default
value");
    require(min <= max, "Invalid min and max");
```

```
    _paramsSet.add(param);
    _paramConfigs[param] = ParamConfig(defaultValue, min, max);
    emit UpsertParamConfig(param, defaultValue, min, max);
  }
```

Recommendation It is recommended to remove redundant functions and judgments.

Status **Fixed.**

```
  function _upsertParamConfig(bytes32 param, uint256 defaultValue,
uint256 min, uint256 max) internal {
    require(param.length > 0, "Empty param name");
    require(min <= defaultValue && defaultValue <= max, "Invalid default
value");
    if (_paramsSet.contains(param)) {
      ParamConfig storage config = _paramConfigs[param];
      config.defaultValue = defaultValue;
      config.min = min;
      config.max = max;
    }
    else {
      _paramsSet.add(param);
      _paramConfigs[param] = ParamConfig(defaultValue, min, max);
    }
    emit UpsertParamConfig(param, defaultValue, min, max);
  }
```

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