BIG$HIBAAI
Smart Contract Audit Report
June 2023

Submitted By
Antier Solutions

Contents

Disclaimer

This is a limited audit report based on our analysis of the BIG$HIBAAI Smart Contract. It covers industry best practices as of the date of this report, concerning: smart contract best coding practices, cybersecurity vulnerabilities, issues in the frameworkand algorithms based on white paper, code, the details of which are set out in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Smart contracts are deployed and executed on the blockchain. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks.

You are advised to read the full report to get a full view of our analysis. While we did our best in producing this report, it is important to note that you should not rely on this report, and cannot claim against us, based on what it says or does not say, or how we produced it, and you need to conduct your independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy all copies of this report downloaded and/or printed by you.

The report is provided "as is," without any condition, warranty, or other terms of any kind except as set out in this disclaimer. TAS hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose, and the use of reasonable care and skill) which, but for this clause, might affect the report.

Executive Summary

BIG$HIBAAI commissioned Antier Solutions to perform an end-to-end source code review of their Solidity Smart Contract. Team Antier Solutions (referred to as TAS throughout the report) performed the audit on 29th June 2023.

The following report discusses severity issues and their scope of rectification through change recommendations. It also highlights activities that are successfully executed and others that need total reworking (if any).

The report emphasizes best practices in coding and the security vulnerabilities if any.

The information in this report should be used to understand the overall code quality, security, and correctness of the Smart Contract. The analysis is static and entirely limited to the Smart Contract code.

In the audit, we reviewed the Smart Contract's code that implements the token mechanism.

Scope

We performed an independent technical audit to identify Smart Contract uncertainties. This shall protect the code from illegitimate authorization attempts or external & internal threats of any type. This also ensures end-to-end proofing of the contract from frauds.

The audit was performed semi-manually. We analysed the Smart Contract code line-by-line and used an automation tool to report any suspicious code.

We considered the following standards for the Smart Contract code review:

- ERC20 [Token Contract best practices]

We used the following tools to perform automated tests:

- Manual Testing tool :
  Hardhat
- Framework :
  Remix Ethereum
- Automation tools:
  - Slither
  - Surya
  - Mythril

Audit Goals

The focus of the audit was to verify that the Smart Contract system was secure, resilient, and worked according to the specifications provided to the Auditing team.

TAS grouped the audit activities in the following three categories:

- **Security**
  Identifying security-related issues within each contract and the system of contract
- **Architecture**
  Evaluation of the system architecture against smart contracting conventions and general software best practice
- **Code Correctness and Quality**
  A full review of the contract source code. The primary areas of focus include:
  - Correctness
  - Readability
  - Sections of code with high complexity
  - Quantity and quality of test coverage

Result

The audit was conducted on the single contract file provided to the Auditing team.

The following table provides an overall picture of the security posture. ✔ means no bugs were identified.

#	Smart Contract Penetration Test (Objectives)	Audit Subclass	Result
1	Overflow	-	✔
2	Race Condition	-	✔
3	Permissions		✔
4	Safety Design	Openzeppelin Safemath	✔
5	Ddos Attack	Call Function Security	✔
Overall Security Posture		Secure	

Recommendations

The BIG$HIBAAI development team demonstrated high technical capabilities, both in the design of the architecture and implementation of the Smart Contract. Overall, the code includes effective use of abstraction, separation of concerns, and modularity.

Code Quality and Readability

The code follows all coding conventions with no functional defects, no missing patterns, or corner cases. The comments should be provided to enhance the readability quotient.

```
/**
 * @dev See {BEP20-transferFrom}.
 *
 * Emits an {Approval} event indicating the updated allowance. This is not
 * required by the EIP. See the note at the beginning of {BEP20};
 *
 * Requirements:
 * - `sender` and `recipient` cannot be the zero address.
 * - `sender` must have a balance of at least `amount`.
 * - the caller must have allowance for `sender`'s tokens of at least
 * `amount`.
 */
function transferFrom(address sender, address recipient, uint256 amount) external returns (bool) {
  _transfer(sender, recipient, amount);
  _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "BEP20: transfer amount exceeds allowance"));
  return true;
}

/**
 * @dev Atomically increases the allowance granted to `spender` by the caller.
 *
 * This is an alternative to {approve} that can be used as a mitigation for
 * problems described in {BEP20-approve}.
 *
 * Emits an {Approval} event indicating the updated allowance.
 *
 * Requirements:
 *
 * - `spender` cannot be the zero address.
 */
function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
  _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
  return true;
}
```

Function's comments should include @dev, @notice, @param for total compliance with Solidity standards.

Code security

TAS performed a static analysis of the code to identify possible loopholes. This verified whether the contract adhered to the Solidity best practices.

Implementation instructions

Include Unit Test cases for better understanding and testing of Gas limits. All the implementation instructions should be written in the README file.

Tokens locked forever

The "transfer" functions will allow you to send tokens to the contract address but there is no way to retrieve them back from the contracts that can't handle it. That means, tokens will be locked up into the contract forever. ***"60% of total circulating supply was accidentally transferred to the token smart contract, those tokens are locked forever in the smart contract and no one can withdraw/retrieve them"***. The locked tokens are like they're -burned forever.

With this way you are going to lock them or we can say burn them but without reducing the total supply.

Severity Level References

The following severity levels will describe the degree of every issue:

High severity issues

The issue puts the majority of, or large numbers of, users' sensitive information at risk, or are reasonably likely to lead to a catastrophic impact on the client's reputation or serious financial implications for the client and users.

Medium severity issues

The issue puts a subset of individual users' sensitive information at risk; exploitation would be detrimental to the client's reputation or is reasonably likely to lead to moderate financial impact.

Low severity issues

The risk is relatively low and could not be exploited regularly, or it's a risk not indicated as important or impactful by the client because of the client's business circumstances.

Informational

The issue does not pose an immediate threat to continued operation or usage but is relevant for security best practices, software engineering best practices, or defensive redundancy.

Optimization issue

The issue does not pose an immediate threat to continued operation or usage but is relevant for code optimization and gas efficiency best practices.

Number of vulnerabilities per severity

High	Medium	Low	Informational
0	2	1	1

Medium Severity Vulnerabilities

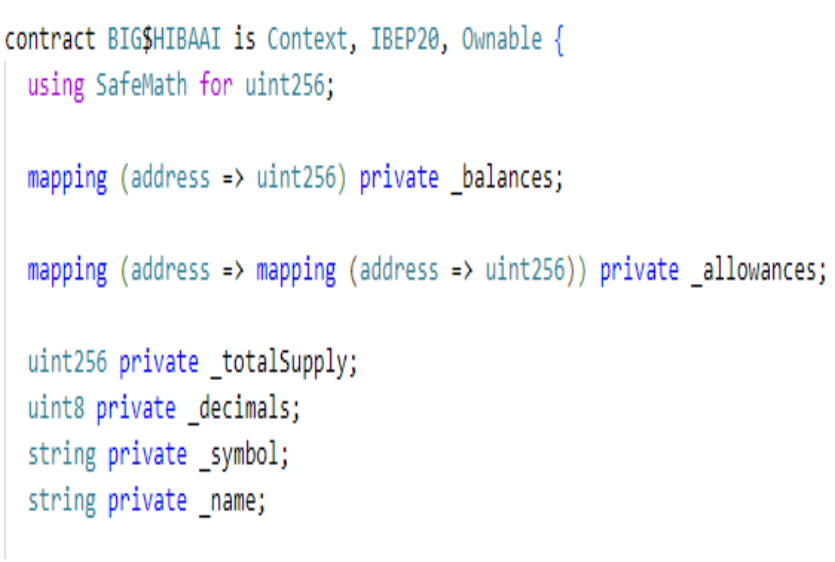
Severity	Medium
Contract	BIG$HIBAAI.sol *_supply * 18**_decimals*
Description	The arithmetic operator can overflow. It is possible to cause an integer overflow or underflow in the arithmetic operation.
Recommendation	It is recommended to use vetted safe math libraries for arithmetic operations consistently throughout the smart contract system.
Status	

Severity	Medium

Contract	SafeMath *assert(c >= a)*
Description	An assertion violation was triggered. It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values)
Recommendation	Consider whether the condition checked in the assert() is actually an invariant. If not, replace the assert() statement with a require() statement. If the exception is indeed caused by unexpected behaviour of the code, fix the underlying bug(s) that allow the assertion to be violated.
Status	

Low Severity Vulnerabilities

Severity	Low
Description	**Lack of zero-check**

Code	
Recommendation	Constructor should have a method to check owner address given is not a zero address
Status	

```
contract BIG$HIBAAI is Context, IBEP20, Ownable {
  using SafeMath for uint256;

  mapping (address => uint256) private _balances;

  mapping (address => mapping (address => uint256)) private _allowances;

  uint256 private _totalSupply;
  uint8 private _decimals;
  string private _symbol;
  string private _name;
```

Informational Severity Vulnerabilities

Severity	**Informational**
Recommendation	**A public function that could be declared external**
Function	BIG$HIBAAI.mint(address,uint256)
Description	Gas can be optimized to avoid over-public declaration.
Status	Safe

Technical Analysis

The following is our automated and manual analysis of the BIG$HIBAAI Smart Contractcode:

Checked Vulnerabilities

We checked BIG$HIBAAI Smart Contract for commonly known and specific business logic vulnerabilities. Following is the list of vulnerabilities tested in the Smart Contract code:

- Reentrancy
- Timestamp Dependence
- Gas limit and loops
- DoS with (unexpected) throw
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fall back function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level
- Address hardcoded
- Using delete for arrays
- Integer overflow/underflow
- Locked money
- Private modifier
- Revert/require functions
- Using var
- Visibility

Automation Report

TAS used Slither and Mythril to generate the automation report.

Slither

Slither is a Solidity static analysis framework written in Python 3.11 It runs a suite of vulnerability detectors, prints visual information about contract details, and provides an API to write custom analyses. Slither enables developers to find vulnerabilities, enhance their code comprehension, and quickly prototype custom analyses.

```
INFO:Printers:
Compiled with Builder
Number of lines: 269 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 8 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 8
Number of informational issues: 32
Number of low issues: 1
Number of medium issues: 0
Number of high issues: 0

ERCs: ERC20

+----------+-------------+-------+------------+--------------+----------+
```

Limitations on Disclosure and Use of this Report

This report contains information concerning potential details of BIG$HIBAAI and methods for exploiting them. Antier Solutions recommends that precautions should be taken to protect the confidentiality of this document and the information contained herein.

Security Assessment is an uncertain process based on experiences, currently available information, and known threats. All information security systems, which by their nature are dependent on human beings, are vulnerable to some degree. Therefore, although Antier Solutions has identified major security vulnerabilities of the analysed system, there can be no assurance that any exercise of this nature will identify all possible vulnerabilities or propose exhaustive and operationally viable recommendations to mitigate those exposures.

As technologies and risks change over time, the vulnerabilities associated with the operation of the BIG$HIBAAI Smart Contract described in this report, as well as the actions necessary to reduce the exposure to such vulnerabilities will also change. Antier Solutions makes no undertaking to supplement or update this report based on the changed circumstances or facts of which Antier Solutions becomes aware after thedate hereof.

This report may recommend that Antier Solutions use certain software or hardware products manufactured or maintained by other vendors. Antier Solutions bases these recommendations on its prior experience with the capabilities of those products. Nonetheless, Antier Solutions does not and cannot warrant that a particular product will work as advertised by the vendor, nor that it will operate in the manner intended.

The Non-Disclosure Agreement (NDA) in effect between Antier Solutions and BIG$HIBAAI Ltd. governs the disclosure of this report to all other parties, including product vendors and suppliers.

www.antiersolutions.com | info@antiersolutions.com | India I UK I US I UAE