SECUREVERSE

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

For
zkPiggyAI

Table of Contents

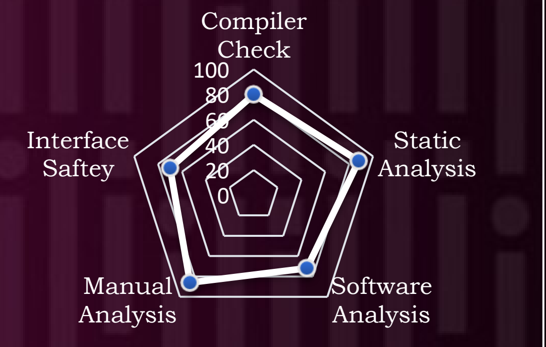
Disclaimer

The Secureverse team examined this smart contract in accordance with industry best practices. We made every effort to secure the code and provide this report. audits done by smart contract auditors and automated algorithms; however, it is crucial to remember that you should not rely entirely on this report. The smart contract may have flaws that allow for hacking. As a result, the audit cannot ensure the explicit security of the audited smart contracts. The Secureverse and its audit report do not encourage readers to consider them as providing any project-related financial or legal advice.

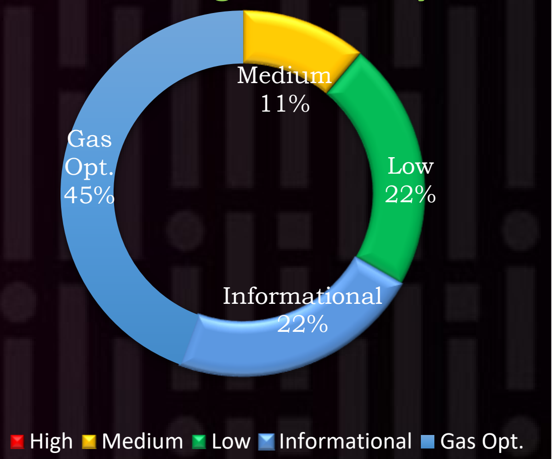
Executive Summary

Project Name	zkPiggyAI
Project Type	Meme Coin
Audit Scope	Check security and code quality
Audit Method	Static and Manual
Audit Timeline	25th April, 2023 to 28th April 2023
Source Code	https://bscscan.com/address/0x81e1291fcbc7f13557d38d710a776eb090b38669#code

Security Score



Finding Summary



Issue Tracking Table

	High	Medium	Low	Informational	Gas Optimization
Open Issues	-	-	-	-	-
Acknowledged Issues	-	1	2	2	4
Resolved Issues	-	-	-	-	-

Types of Severities

- **High:** The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.

- **Medium:** The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.

- **Low:** The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.

- **Informational:** The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth.

Types of Issues

- **Open:** Security vulnerabilities identified that must be resolved and are currently unresolved.

- **Acknowledged:** The way in which it is being used in the project makes it unnecessary to address the vulnerabilities. This means that the way it has been acknowledged has no effect on its security.

- **Resolved:** These are the issues identified in the initial audit and have been successfully fixed.

Checked Vulnerabilities

- ❖ Re-entrancy
- ❖ Access control
- ❖ Denial of service
- ❖ Timestamp Dependence
- ❖ Integer overflow/Underflow
- ❖ Transaction Order Dependency
- ❖ Requirement Violation
- ❖ Functions Visibility Check
- ❖ Mathematical calculations
- ❖ Dangerous strict equalities
- ❖ Unchecked Return values
- ❖ Hard coded information
- ❖ Safe Ether Transfer
- ❖ Gas Consumption
- ❖ Incorrect Inheritance Order
- ❖ Centralization
- ❖ Unsafe external calls
- ❖ Business logic and specification
- ❖ Input validation
- ❖ Incorrect Modifier
- ❖ Missing events
- ❖ Assembly usage
- ❖ ERC777 hooks
- ❖ Token handling

Methods

Audit at Secureverse is performed by the experts and they make sure that audited project must comply with the industry security standards.

Secureverse audit methodology includes following key:

- In depth review of the white paper
- In depth analysis of project and code documentation.
- Checking the industry standards used in Code/Project.
- Checking and understanding Core Functionality of the Code.
- Comparing the code with documentation.
- Static analysis of the code.
- Manual analysis of the code.
- Gas Optimization and Function Testing.
- Verification of the overall audit.
- Report writing.

The following techniques, methods and tools were used to review all the smart contracts.

Static Analysis

Static analysis has been done by using the open source and state of the art automatic smart contract vulnerability scanning tools.

Manual Analysis

Manual analysis is done by our smart contract auditors' team by performing in depth analysis of the smart contract and identify potential vulnerabilities. Auditor also review and verify all the static analysis results to prevent the false positives identified by automated tools.

Gas Consumption and Function Testing

Function testing done by auditors by manually writing customized test cases for the smart contract to verify the intended behavior as per code and documentation. Gas Optimization done by reviews potential gas consumption by contract in production.

Tools and Platforms used for Audit

- Remix IDE
- Hardhat
- Mythril
- Truffle Team
- Solhint
- Solidityscan.com
- Slither
- Consensys Surya
- Open Zeppelin Code Analyzer
- Manticore

Findings

Medium Severity Issues:

[M-01] Owner can burn another user's token:

Reference:
https://bscscan.com/address/0x81E1291fcbc7f13557d38D710A776eb090b38669#code#L218

Description:
This contract has an internal function `_burn()`

```
    function _burn(address account, uint256 value) internal whenNotPaused
{   ////////////////// @audit ownerr can burn others token
        require(account != address(0), "ERC20: burn from the zero address");

        _totalSupply = _totalSupply.sub(value);
        _balances[account] = _balances[account].sub(value);
        emit Transfer(account, address(0), value);
    }
```

Although this is not called in this contract context, but assuming that this `zkPiggyAI` will be a part of some other project, in which this contract will be inherited.
In that particular time owner will able to burn others user fund. A small code walkthrough shown below

```
contract Test is zkPiggyAI{

    event BurnSuccessful();

    function burn(address _addr, uint256 amount)public{
        _burn(_addr, amount);
        emit BurnSuccessful();
    }
}
```

Here Test will be main contract which inheriting zkPiggyAI token contract, then developer create a function burn(or named any) then call internal function `_burn()` from zkPiggyAI and able to burn any other user funds.
If there is no access control on burn as shown above so any user can burn other user funds. This open up a future bug

Remediation:
Those `internal` functions which are not called in this contract should be removed from code base.

Status: Acknowledged

Low Severity Issues:

[L-1] Transfer of OwnerShip should be 2-step process

Description:
Lack of two-step procedure for critical operations (like ownership change) leaves them error-prone. Consider adding a two-step procedure on the critical functions. For example:

- `address public upcomingOwner`
- `approveOwnerShip()`
  A function which called by current owner and set new owner address to \`upcomingOwner\`
- `getOwnerShip()`
  Only called by newOwner(via checking \`upcomingOwner\` statevariable) and set owner to newOwner and upcomingOwner to Zero address.

Status: Acknowledged

[L-2] Owner can mint as much token as he can.

Reference:
https://bscscan.com/address/0x81E1291fcbc7f13557d38D710A776eb090b38669#code#L210

Description:
Like explained in [M-02] Issue, there is an \`internal\` function named \`_mint()\`, So owner can inherit this contract to main contract and able to mint as much as token he can. That will impact tokenomics of project.

Remediation:
There should be a max capped value on token minting amount. Those which \`internal\` functions not called in this contract should be removed from code base.

Status: Acknowledged

Informational Issues

[NC-1] Solidity above 0.8.0 by default comes with Integer Overflow & Underflow protection, so no need to use SafeMath library:

Remediation:
Remove SafeMath library

Status: Acknowledged

[NC-2] `interface` should declared in separate file.

Remediation:
`interface` should declared in separate file and then import to current main file. That’s how it will looks more structured and clean code.

Status: Acknowledged

Gas Optimization

[G-1] Use solidity version 0.8.19 to gain some gas boost:

Remediation:
Upgrade to the latest solidity version 0.8.19 to get additional gas savings. See latest release for reference: https://blog.soliditylang.org/2023/02/22/solidity-0.8.19-release-announcement/

Status: Acknowledged

[G-2] Bool Storage overhead:

Reference:
https://bscscan.com/address/0x81E1291fcbc7f13557d38D710A776eb090b38669#code#L89

```
bool private _paused;
```

Remediation:
Instead of bool, here you can use uint256 and uint256(1)/ uint256(2) for true/false respectively to avoid a Gwarmaccess (**100 gas**) for the extra SLOAD, and to avoid Gsset (**20000 gas**) when changing from false to true(In case pausing contract), after having been true in the past.

Status: Acknowledged

[G-3] Instead of `public`, some functions could be `external`:

Reference:

File	Line Number
https://bscscan.com/address/0x81E1291fcbc7f13557d38D710A776eb090b38669#code	183, 192, 197, 243, 246, 250

Description:
Those Function which are not called inside contract can make as `external` instead of `public`.
Contracts are allowed to override their parents' functions and change the visibility from external to public and can save gas by doing so.

Status: Acknowledged

[G-4] Functions those will revert when called by normal user should marked as `payable`:

Reference:

File	Line Number
https://bscscan.com/address/0x81E1291fcbc7f13557d38D710A776eb090b38669#code	151, 155, 183

Description:

If a function modifier such as `onlyOwner` is used, the function will revert if a normal user tries to pay the function. The extra opcodes avoided are `CALLVALUE`(2), `DUP1`(3), `ISZERO`(3), `PUSH2`(3), `JUMPI`(10), `PUSH1`(3), `DUP1`(3), `REVERT`(0), `JUMPDEST`(1), `POP`(2), which costs an average of about 21 gas per call to the function, in addition to the extra deployment cost.

Remediation:

Marking the function as `payable` will lower the gas cost for legitimate callers because the compiler will not include checks for whether a payment was provided.

Status: Acknowledged

Automated Testing

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this audit, we examined the COMEARTH's smart contract with our framework, and we discovered several medium, low, and informational flaws in the smart contract. We have included solutions and recommendations in the audit report to improve the quality and security posture of the code. All of the findings and solutions have been acknowledged by the project team. In summary, we find that the codebase with the latest version greatly improved on the initial version. We believe that the overall level of security provided by the codebase in its current state is reasonable, so we have marked it as **Secure** and the Customer's smart contract has the following score: 8

1	2	3	4	5	6	7	8	9	10

About Secureverse

Secureverse is the Singapore and India based emerging Web3 Security solution provider. We at Secureverse provides the Smart Contract audit, Blockchain infrastructure Penetration testing and the Cryptocurrency forensic services with very affordable prices.

To Know More

Twitter: https://twitter.com/secureverse

LinkedIn: https://www.linkedin.com/company/secureverse/

Telegram: https://t.me/secureverse

Email Address: http://secureverse@protonmail.com/