SMART CONTRACT AUDIT

interfinetwork

hello@interfi.network

https://interfi.network

PREPARED FOR

APEESCAPE MINER CONTRACT

INTRODUCTION

Auditing Firm	InterFi Network
Client Firm	ApeEscape
Methodology	Automated Analysis, Manual Code Review
Language	Solidity
Contract	0x8737b780dbef3639529b550170f112e3545d2bd0
Blockchain	Binance Smart Chain
Centralization	Active Ownership
Commit	0836b98b4dab1ec21cae89dd18e4b30e80f4e6e0
Website	https://apeescape.io
Banana Farm	https://bananafarm.apeescape.io
X	https://x.com/escapeonchain
Telegram	https://t.me/ape_escape
Report Date	February 27, 2025

Verify the authenticity of this report on our website: https://www.github.com/interfinetwork

EXECUTIVE SUMMARY

InterFi has performed the automated and manual analysis of solidity codes. Solidity codes were reviewed for common contract vulnerabilities and centralized exploits. Here's a quick audit summary:

Status	Critical 🔴	Major 🟠	Medium 🟡	Minor 🟢	Unknown 🟤
Open	0	0	0	6	0
Acknowledged	0	0	3	1	1
Resolved	0	1	0	1	0
Important Functions	`hatchEggs, sellEggs, buyEggs`				
Major 🟠 Privileges	`seedMarket, updateTreasuryWallet, updateDevWallet`				

ℹ️ Please note that smart contracts deployed on blockchains aren't resistant to exploits, vulnerabilities and/or hacks. Blockchain and cryptography assets utilize new and emerging technologies. These technologies present a high level of ongoing risks. For a detailed understanding of risk severity, source code vulnerability, and audit limitations, kindly review the audit report thoroughly.

ℹ️ Please note that centralization privileges regardless of their inherited risk status - constitute an elevated impact on smart contract safety and security.

ℹ️ Please note that the absence of public KYC verification of the project owners, team members, or deployers associated with ApeEscape Miner. Typically, third-party KYC processes are instrumental in ensuring the transparency and accountability of a project's leadership, thereby enhancing user trust and regulatory compliance. Without external KYC verification by reputable providers, users may face increased risks related to rug pull.

TABLE OF CONTENTS

SCOPE OF WORK

InterFi was consulted by ApeEscape Miner to conduct the smart contract audit of their solidity source codes. The audit scope of work is strictly limited to mentioned solidity file(s) only:

- ApeEscapeBananaFarm.sol

If source codes are not deployed on the main net, they can be modified or altered before main-net deployment. Verify the contract's deployment status below:

Public Contract Link	
https://bscscan.com/address/0x8737b780dbef3639529b550170f112e3545d2bd0#code	
Contract Name	ApeEscapeBananaFarm
Compiler Version	0.8.28
License	MIT

AUDIT METHODOLOGY

Smart contract audits are conducted using a set of standards and procedures. Mutual collaboration is essential to performing an effective smart contract audit. Here's a brief overview of InterFi's auditing process and methodology:

CONNECT

- The onboarding team gathers source codes, and specifications to make sure we understand the size, and scope of the smart contract audit.

AUDIT

- Automated analysis is performed to identify common contract vulnerabilities. We may use the following third-party frameworks and dependencies to perform the automated analysis:
  - Remix IDE Developer Tool
  - Open Zeppelin Code Analyzer
  - SWC Vulnerabilities Registry
  - DEX Dependencies, e.g., Pancakeswap, Uniswap
- Simulations are performed to identify centralized exploits causing contract and/or trade locks.
- A manual line-by-line analysis is performed to identify contract issues and centralized privileges. We may inspect below mentioned common contract vulnerabilities, and centralized exploits:

Centralized Exploits	○ Token Supply Manipulation ○ Access Control and Authorization ○ Assets Manipulation ○ Ownership Control ○ Liquidity Access ○ Stop and Pause Trading ○ Ownable Library Verification

Common Contract Vulnerabilities	○ Integer Overflow ○ Lack of Arbitrary limits ○ Incorrect Inheritance Order ○ Typographical Errors ○ Requirement Violation ○ Gas Optimization ○ Coding Style Violations ○ Re-entrancy ○ Third-Party Dependencies ○ Potential Sandwich Attacks ○ Irrelevant Codes ○ Divide before multiply ○ Conformance to Solidity Naming Guides ○ Compiler Specific Warnings ○ Language Specific Warnings

REPORT

- The auditing team provides a preliminary report specifying all the checks which have been performed and the findings thereof.
- The client's development team reviews the report and makes amendments to solidity codes.
- The auditing team provides the final comprehensive report with open and unresolved issues.

PUBLISH

- The client may use the audit report internally or disclose it publicly.

ℹ️ It is important to note that there is no pass or fail in the audit, it is recommended to view the audit as an unbiased assessment of the safety of solidity codes.

RISK CATEGORIES

A successful external attack may allow the external attacker to directly exploit. A successful centralization-related exploit may allow the privileged role to directly exploit. All risks which are identified in the audit report are categorized:

Risk Type	Definition
Critical 🔴	These risks pose immediate and severe threats, such as asset theft, data manipulation, or complete loss of contract functionality. They are often easy to exploit and can lead to significant, irreparable damage. Immediate fix is required.
Major 🟠	These risks can significantly impact code performance and security, and they may indirectly lead to asset theft and data loss. They can allow unauthorized access or manipulation of sensitive functions if exploited. Fixing these risks are important.
Medium 🟡	These risks may create attack vectors under certain conditions. They may enable minor unauthorized actions or lead to inefficiencies that can be exploited indirectly to escalate privileges or impact functionality over time.
Minor 🟢	These risks may include inefficiencies, lack of optimizations, code-style violations. These should be addressed to enhance overall code quality and maintainability.
Unknown 🟤	These risks pose uncertain severity to the contract or those who interact with it. Immediate fix is required to mitigate risk uncertainty.

All statuses which are identified in the audit report are categorized here:

Status Type	Definition
Open	Risks are open.
Acknowledged	Risks are acknowledged, but not fixed.
Resolved	Risks are acknowledged and fixed.

CENTRALIZED PRIVILEGES

Centralization risk is the most common cause of cryptography asset loss. When a smart contract has a privileged role, the risk related to centralization is elevated.

There are some well-intended reasons have privileged roles, such as:

- Privileged roles can be granted the power to `pause()` the contract in case of an external attack.
- Privileged roles can use functions like, `include()`, and `exclude()` to add or remove wallets from fees, swap checks, and transaction limits. This is useful to run a presale and to list on an exchange.

Authorizing privileged roles to externally-owned-account (EOA) is dangerous. Lately, centralization-related losses are increasing in frequency and magnitude.

- The client can lower centralization-related risks by implementing below mentioned practices:
- Privileged role's private key must be carefully secured to avoid any potential hack.
- Privileged role should be shared by multi-signature (multi-sig) wallets.
- Authorized privilege can be locked in a contract, user voting, or community DAO can be introduced to unlock the privilege.
- Renouncing the contract ownership, and privileged roles.
- Remove functions with elevated centralization risk.

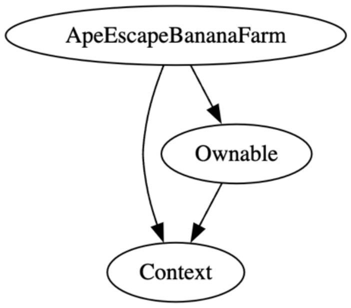
ℹ️ Understand the project's initial asset distribution. Assets in the liquidity pair should be locked. Assets outside the liquidity pair should be locked with a release schedule.

AUTOMATED ANALYSIS

Symbol	Definition
🛑	Function modifies state
💵	Function is payable
🔒	Function is internal
🔐	Function is private
❗	Function is important

```
| **Context** | Implementation |  |||
| └ | _msgSender | Internal 🔒 |   | |
||||||
| **Ownable** | Implementation | Context |||
| └ | <Constructor> | Public ❗ | 🛑  |NO❗ |
| └ | owner | Public ❗ |   |NO❗ |
| └ | renounceOwnership | Public ❗ | 🛑  | onlyOwner |
| └ | transferOwnership | Public ❗ | 🛑  | onlyOwner |
||||||
| **ApeEscapeBananaFarm** | Implementation | Context, Ownable |||
| └ | <Constructor> | Public ❗ | 🛑  |NO❗ |
| └ | <Receive Ether> | External ❗ |  💵 |NO❗ |
| └ | hatchEggsInternal | Internal 🔒 | 🛑  | |
| └ | hatchEggs | Public ❗ | 🛑  |NO❗ |
| └ | sellEggs | Public ❗ | 🛑  |NO❗ |
| └ | buyEggs | Public ❗ |  💵 |NO❗ |
| └ | getBalance | Public ❗ |   |NO❗ |
| └ | getMyMiners | Public ❗ |   |NO❗ |
| └ | getMyEggs | Public ❗ |   |NO❗ |
| └ | getEggsSinceLastHatch | Public ❗ |   |NO❗ |
| └ | calculateEggSell | Public ❗ |   |NO❗ |
| └ | calculateEggBuy | Public ❗ |   |NO❗ |
| └ | calculateEggBuySimple | Public ❗ |   |NO❗ |
| └ | devFee | Private 🔐 |   | |
| └ | bananaRewards | Public ❗ |   |NO❗ |
| └ | seedMarket | Public ❗ |  💵 | onlyOwner |
| └ | calculateTrade | Private 🔐 |   | |
| └ | min | Private 🔐 |   | |
| └ | getMinerValueInBNB | Public ❗ |   |NO❗ |
| └ | updateTreasuryWallet | External ❗ | 🛑  | onlyOwner |
| └ | updateDevWallet | External ❗ | 🛑  | onlyOwner |
```

INHERITANCE GRAPH



MANUAL REVIEW

Identifier	Definition	Severity
CEN-01	Centralized privileges	Medium 🟡
CEN-01-01	Privileged role can initialize market	

Important `onlyOwner` centralized privileges are listed below:

```
renounceOwnership
transferOwnership
seedMarket
updateTreasuryWallet
updateDevWallet
```

RECOMMENDATION

Securing private keys or access credentials of deployers, contract owners, operators, and other roles with privileged access is crucial to prevent single points of failure that can compromise contract security.

Use of multi-signature wallets is recommended – These wallets require multiple authorizations to execute sensitive contract functions, reducing the risk associated with single-party control.

Use of decentralized governance model is recommended – This model allows token holders and stakeholders to actively participate in decision-making, such as contract upgrades and parameter adjustments, enhancing overall security and resilience.

ACKNOWLEDGEMENT

ApeEscape team argued that centralized and controlled privileges are used as required.

Identifier	Definition	Severity
CEN-02	First Depositor Advantage (Market Initialization Vulnerability)	Major 🟠

`seedMarket()` function initializes `marketEggs` to a fixed value (`108000000000`). The `calculateTrade()` function is the core of the buy/sell logic. The problem is that the very first deposit after `seedMarket()` has an enormously disproportionate effect on the price. Since the contract balance starts at very low, if `seedMarket()` is called with some BNB, the first buyer gets a massive amount of eggs for a tiny amount of BNB. This is because the calculation heavily favors the `rs - marketEggs` when `bs` is small.

Exploitation Scenario: A malicious actor could:

Deploy the contract.

Call seedMarket() with no initial BNB.

Immediately call buyEggs() with a small amount of BNB (e.g., 0.01 BNB). This gives them a huge number of miners.

Wait for other users to deposit, increasing the contract's balance.

Sell their eggs for a much higher price, draining the contract and leaving other users with significantly devalued miners.

RECOMMENDATION

`seedMarket()` should be modified to accept an initial BNB deposit along with setting the initial `marketEggs`. This deposit should be substantial enough to mitigate the first depositor advantage. The best practice is to seed the market with an amount of BNB and eggs that reflects a reasonable starting price. This initial BNB must be sent along with the `seedMarket()` call, making it part of the `address(this).balance` before any user can interact with `buyEggs`.

RESOLUTION

ApeEscape team has seeded market. There's little to none first depositor advantage for privileged role at current stage.

https://bscscan.com/tx/0xc86e1502f2d9b759cfdaabf62181188b01efd03dfd958d80f38da180f2f2520a

Identifier	Definition	
CEN-04	Lack of proxy and upgradeable contracts	

Privileged role cannot authorize contract upgrade. Contract upgradeability allows privileged roles to change current contract implementation in case of a hack.

RECOMMENDATION

Employ proxy mechanism to enable future upgradeability. This design allows for seamless upgrades to contract functionality, ensuring better contract maintainability and adaptability.

Identifier	Definition	Severity
LOG-01	Inaccurate sell penalty calculation	Minor 🟢

`sellPercentage` calculation in `sellEggs()` intends to penalize users who sell a large portion of their potential earnings. However, the calculation is flawed. It compares the `eggValue` to the potential earnings of the *current* miners. This doesn't accurately reflect the *proportion* of potential earnings being sold.

RECOMMENDATION

Compare `hasEggs` to `getMyEggs(msg.sender)` before `claimedEggs[msg.sender]` is reset to zero, which includes both accumulated and unclaimed eggs.

ACKNOWLEDGEMENT

ApeEscape team commented that smart contract's penalty calculation fairly assesses a player's selling behavior by using the formula `sellPercentage = (eggValue * 100) / calculateEggSell(currentMiners * EGGS_TO_HATCH_1MINERS)`. This approach scales penalties with a player's current mining power, reflecting their immediate production capacity rather than speculative lifetime earnings. Their team argues this is appropriate since earnings vary due to reinvestment, claiming habits, and inflation, making long-term estimates impractical.

Identifier	Definition	Severity
LOG-02	Potential front-running	Medium 🟡

Potential front-running happens when an attacker observes a transaction swapping tokens or adding liquidity without setting restrictions on slippage or minimum output amount. The attacker can manipulate the exchange rate by front-running a transaction to purchase assets and make profits by back-running a transaction to sell assets. Below mentioned functions are called without setting restrictions on slippage or minimum output:

```
hatchEggs
sellEggs
buyEggs
```

RECOMMENDATION

Implement commit-reveal schemes or transaction ordering to protect against front-running.

ACKNOWLEDGEMENT

According to ApeEscape team, front-running is not a concern in this contract due to its specific design and mechanics. Transactions adhere to a predefined contract logic, which ensures that users cannot manipulate the execution order to gain an unfair advantage. Additionally, egg-related transactions—such as selling or hatching—are tied to an individual user's state, meaning these actions are personal and remain unaffected by other users attempting to jump ahead in the transaction queue. Smart contract also prevents users from directly selling their full position, adding another layer of protection against potential front-running exploits.

Identifier	Definition	Severity
LOG-03	Re-entrancy	Medium 🟡

Below mentioned functions are used without Re-entrancy guard (CRIT Evaluation):

```
sellEggs
buyEggs
```

RECOMMENDATION

Use Checks-Effects-Interactions (CEI) pattern when transferring control to external entities. This design pattern ensures that all state changes are completed before external interactions occur. Additionally, implement re-entrancy guard to block recursive calls from external contracts.

RE-EVALUATION FROM CRIT TO MED

- In `sellEggs`, all state changes precede transfers. In `buyEggs`, key state variables (`totalDeposits`, `marketEggs`) are updated before transfers, though `claimedEggs` and `hatchEggsInternal` come after. Since the transfers are to EOAs (bullet point 3), this design minimizes re-entrancy risk.
- Critical state `totalDeposits` is updated before transfers, ensuring an attacker couldn't re-enter and exploit outdated state.
- EOAs can't initiate re-entrant calls during `.transfer`, unlike smart contracts, which could execute a fallback function.

ACKNOWLEDGEMENT

Assuming `treasuryWallet` and `devWallet` remain EOAs, no direct re-entrancy risk exists. However, privileged roles can update these EOAs to contracts, enabling re-entrancy attacks and introducing critical vulnerabilities. This becomes a more of a centralization issue in this case, CEN-01.

Identifier	Definition	Severity
LOG-05	Logical non-conformities	Minor 🟢

`buyEggs()` function limits purchases to 10% of the contract balance. This anti-whale measure is easily bypassed. A determined whale can simply make multiple purchases in separate transactions. There's no per-address limit or time-based restriction.

RECOMMENDATION

Introduce a mapping to track the total BNB deposited by each address and enforce a maximum.

Identifier	Definition	Severity
COD-02	Timestamp dependence	Minor 🟢

Be aware that the timestamp of the block can be manipulated by miners. Since miners can slightly adjust the timestamp, they may influence contract outcomes to their advantage.

RECOMMENDATION

Avoid relying solely on timestamp of the block for critical contract functions. Follow 15 seconds rule, and scale time dependent events accordingly.

Identifier	Definition	Severity
COD-10	Direct and indirect dependencies	Unknown ●

Scope of this audit treats `Ownable` and `Context` contracts as black boxes and assumes their functional correctness. The security and stability of the BNB Chain are also outside the scope of this audit. We assume that `treasuryWallet` and `devWallet` are securely managed. While this contract avoids direct interaction with complex external DeFi protocols, the broader ecosystem risks associated with blockchain technology and cryptocurrency still apply.

RECOMMENDATION

Inspect third party dependencies regularly, and mitigate severe impacts whenever necessary.

ACKNOWLEDGEMENT

ApeEscape team will inspect third party dependencies regularly, and push upgrades whenever required.

Identifier	Definition	Severity
COD-12	Lack of event-driven architecture	Minor 🟢

Smart contract uses function calls to update state, which can make it difficult to track and analyze changes to the contract over time.

```
seedMarket
updateTreasuryWallet
updateDevWallet
```

RECOMMENDATION

Use events to track state changes. Events improve transparency and provide a more granular view of contract activity.

Identifier	Definition	Severity
VOL-01	Redundant `hatchEggsInternal` function	Minor 🟢

Code for `hatchEggsInternal` and `hatchEggs` is identical except for the added time restriction in `hatchEggs (require(block.timestamp >= lastHatch[msg.sender] + 1800, "Wait 30 minutes before compounding again");)`. This creates unnecessary code duplication.

RECOMMENDATION

Remove `hatchEggsInternal` entirely and move the `require` statement for the time restriction to the beginning of the `hatchEggs()`.

Identifier	Definition	Severity
VOL-03	Division before multiplication	Minor ●

Solidity integer division may truncate, and provide unexpected results.

```
hatchEggs
sellEggs
fee calculations
```

RECOMMENDATION

Where possible, rearrange the calculations to perform multiplication before division, unless doing so would risk overflow.

Identifier	Definition	Severity
COM-01	Solidity version consistency	Minor 🟢

Compiler is set to:

```
pragma solidity ^0.8.28;
```

RECOMMENDATION

Using a specific version of solidity using ^ can make the contract behavior change unexpectedly when a new version is released.

RESOLUTION

Smart contract is deployed with stable compiler.

Identifier	Definition	Severity
COM-04	Gas optimization	Minor ●

In `buyEggs` function, contract calculates `contractBalance - msg.value`. This subtraction is unnecessary because the `contractBalance` variable is already defined as `address(this).balance` and `msg.value` is part of it.

RECOMMENDATION

Optimize functions, arrays, and loops identified as high gas consumers by simplifying logic, minimizing state changes, and limiting loop iterations where possible.

DISCLAIMERS

InterFi Network provides the easy-to-understand audit of solidity source codes (commonly known as smart contracts).

The smart contract for this particular audit was analyzed for common contract vulnerabilities, and centralization exploits. This audit report makes no statements or warranties on the security of the code. This audit report does not provide any warranty or guarantee regarding the absolute bug-free nature of the smart contract analyzed, nor do they provide any indication of the client's business, business model or legal compliance. This audit report does not extend to the compiler layer, any other areas beyond the programming language, or other programming aspects that could present security risks. Cryptographic tokens are emergent technologies, they carry high levels of technical risks and uncertainty. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. This audit report could include false positives, false negatives, and other unpredictable results.

CONFIDENTIALITY

This report is subject to the terms and conditions (including without limitations, description of services, confidentiality, disclaimer and limitation of liability) outlined in the scope of the audit provided to the client. This report should not be transmitted, disclosed, referred to, or relied upon by any individual for any purpose without InterFi Network's prior written consent.

NO FINANCIAL ADVICE

This audit report does not indicate the endorsement of any particular project or team, nor guarantees its security. No third party should rely on the reports in any way, including to make any decisions to buy or sell a product, service or any other asset. The information provided in this report does not constitute investment advice, financial advice, trading advice, or any other sort of advice and you should not treat any of the report's content as such. This audit report should not be used in any way to make decisions around investment or involvement. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort.

FOR AVOIDANCE OF DOUBT, SERVICES, INCLUDING ANY ASSOCIATED AUDIT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

TECHNICAL DISCLAIMER

ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, INTERFI NETWORK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO SERVICES, AUDIT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, INTERFI NETWORK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM THE COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

WITHOUT LIMITING THE FOREGOING, INTERFI NETWORK MAKES NO WARRANTY OF ANY KIND THAT ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET THE CLIENT'S OR ANY OTHER INDIVIDUAL'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE.

TIMELINESS OF CONTENT

The content contained in this audit report is subject to change without any prior notice. InterFi Network does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following the publication.

LINKS TO OTHER WEBSITES

This audit report provides, through hypertext or other computer links, access to websites and social accounts operated by individuals other than InterFi Network. Such hyperlinks are provided for your reference and convenience only and are the exclusive responsibility of such websites' and social accounts' owners. You agree that InterFi Network is not responsible for the content or operation of such websites and social accounts and that InterFi Network shall have no liability to you or any other person or entity for the use of third-party websites and social accounts. You are solely responsible for determining the extent to which you may use any content at any other websites and social accounts to which you link from the report.

ABOUT INTERFI NETWORK

InterFi Network provides intelligent blockchain solutions. We provide solidity development, testing, and auditing services. We have developed 150+ solidity codes, audited 1000+ smart contracts, and analyzed 500,000+ code lines. We have worked on major public blockchains e.g., Ethereum, Binance, Cronos, Doge, Polygon, Avalanche, Metis, Fantom, Bitcoin Cash, Velas, Oasis, etc.

InterFi Network is built by engineers, developers, UI experts, and blockchain enthusiasts. Our team currently consists of 4 core members, and 6+ casual contributors.

Website: https://interfi.network

Email: hello@interfi.network

GitHub: https://github.com/interfinetwork

Telegram (Engineering): https://t.me/interfiaudits

Telegram (Onboarding): https://t.me/interfisupport