



Audit Report December ,2024



Table of Content

Executive Summary..... 02

Number of Security Issues per Severity 03

Checked Vulnerabilities..... 04

Techniques and Methods..... 05

Types of Severity 06

Types of Issues 06

Findings Details..... 07

 1. Plaintext private keys exposed

 2. Incorrect user balances

 3. No upper limit for fee percentage

 4. Improper error Handling

Functional Tests Cases 09

Automated Tests..... 09

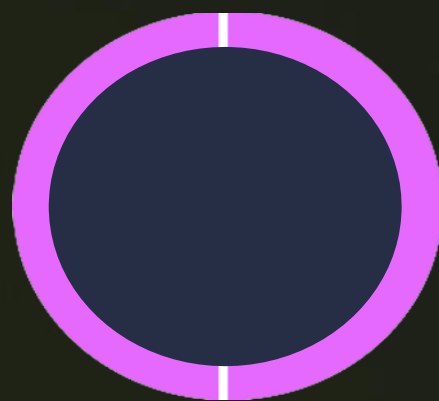
Closing Summary..... 10

Disclaimer..... 10

Executive Summary

Project Name	KarenPsycho-Bot
Project URL	KarenPsycho-Bot/
Overview	A time-boxed security review of the KarenPsycho-Bot was performed by SafeEdges , focusing on the security aspects of the smart contract's implementation.
Audit Scope	https://github.com/D4Musketeers/KarenPsycho-SwapScript https://github.com/D4Musketeers/KarenPsycho-Bot
Language	Sway ,Typescript
Blockchain	Fuel
Method	Manual Analysis, Functional Testing, Automated Testing
First Review	24th Dec 2024 - 31th Dec 2024
Updated Code Received	29 Dec 2024
Second Review	31th oct 2024

Number of Security Issues per Severity



High



Medium



Low



Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	1	2	1	0

Checked Vulnerabilities

- ✓ Arbitrary write to storage
- ✓ Centralization of control
- ✓ Ether theft
- ✓ Improper or missing events
- ✓ Logical issues and flaws
- ✓ Arithmetic Computations Correctness
- ✓ Race conditions/front running
- ✓ Re-entrancy
- ✓ Malicious libraries
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ ERC's conformance
- ✓ Missing Zero Address Validation
- ✓ Revert/require functions
- ✓ Upgradeable safety
- ✓ Using inline assembly
- ✓ Style guide violation
- ✓ Parallel Execution safety
- ✓ UTXO Model Verification
- ✓ FuelVM Opcodes
- ✓ Cross-Chain Interactions
- ✓ Modular Design
- ✓ Access Control Vulnerabilities
- ✓ Denial of Service (DoS)
- ✓ Oracle Manipulation
- ✓ Signature Replay Attacks
- ✓ Improper Handling of External Calls
- ✓ Proxy Storage Collision
- ✓ Use of Deprecated Functions

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments, match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

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

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity statistic analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Finding Details

F-1. Storing User Private Keys in Plaintext Logs

SEVERITY: High

REMEDIATION: See DESCRIPTION

STATUS: fixed

Steps to Reproduce:

1. Navigate to the "Export Private Key" page.
2. Export the private key, ensuring it is sent in an encrypted format.
3. Click the back button to exit the page.
4. Observe the HTTP request sent, which contains the plaintext private key.
5. Check the logs to confirm that the plaintext private key is stored.

Impact:

Leaking Private key in logs

F-2. Incorrect amount_out Calculation After Fee Deduction in Sell Function

SEVERITY: Medium

REMEDIATION: see [description](#)

STATUS: fixed

DESCRIPTION:

The amount_out value is not updated after fees are deducted in the sell function. While the fee is transferred correctly, the user's remaining balance (amount_out) does not reflect the deducted amount

```
if i == pools.len() - 1 {
    if !is_buy {
        let fee_deduction_amount = (amount_out * FEE_PERCENTAGE) / FEE_DENOMINATOR; // x% fee of the last output
        transfer(FEE_ADDR, asset_out, fee_deduction_amount);
    }
}
i += 1;
}
```

amounts_out

Steps to Reproduce:

1. Perform a sell operation.
2. Observe the amount transferred to the user after the fee deduction.
3. Confirm that the amount_out does not account for the deducted fee.

Impact:

- Wrong amount_out logic in sell

Recommended Fix:

- Recalculate and update amount_out after the fee deduction

```
if i == pools.len() - 1 {
  if !is_buy {
    let fee_deduction_amount = (amount_out * FEE_PERCENTAGE) / FEE_DENOMINATOR;
    amount_out -= fee_deduction_amount;
    transfer(FEE_ADDR, asset_out, fee_deduction_amount);
  }
}
```

F-3 No Upper Limit on Fee Percentage

SEVERITY: **Medium**

REMEDIATION: see [description](#)

STATUS: [fixed](#)

DESCRIPTION:

There is a lack of reentrancy guards in functions that perform external calls. The code does not consistently follow the Checks-Effects-Interactions (CEI) pattern.

```
// calculate the fee (x% of the amount_in)
let fee = (amount_in * FEE_PERCENTAGE) / FEE_DENOMINATOR; // x% fee
let mut amount_in_after_fee = amount_in;
if is_buy {
  // If it's a "buy", deduct the x% fee before doing the calculations
  amount_in_after_fee = amount_in - fee;
}
// fee calc end
```

Impact:

- Users lose all their funds due to excessive fees.

Recommended Fix:

- Introduce a maximum limit for FEE_PERCENTAGE (e.g., 5%).
- Validate fee percentages during configuration and runtime.

F-4 Missing Validation or Error Message for Invalid Token Address

SEVERITY: **Low**

REMEDIATION: see [description](#)

STATUS: [fixed](#)

DESCRIPTION:

The system does not validate Ethereum token addresses when users attempt to buy eth tokens. This leads BOT stuck.

Recommended Fix

- Add ERROR HANDLING
- Provide clear and descriptive error messages for invalid inputs.

Automated Tests

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 report, we have considered the security of Bot. We performed our audit according to the procedure described above.

Some issues of informational severity were found, which the Bot Team has Fixed.

Disclaimer

SafeEdges Smart contract security audit provides services to help identify and mitigate potential security risks in Bot. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. SafeEdges audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Bot Safe. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

SafeEdges cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of Bot to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

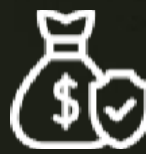
About SAFE EDGES

SafeEdges is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over \$30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.



500+

Audits Completed



\$3B

Secured

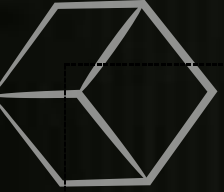


600k+

Lines of Code Audited

Follow Our Journey







Audit Report December, 2024



Safe Edges

 <https://safeedges.in>

 info@safeedges.in