our shielding · Your smart contracts, our shielding · Your smart c

shieldify

Swappee

SECURITY REVIEW

Date: 22 April 2025

CONTENTS

1. About Shieldify

Positioned as the first hybrid Web3 Security company, Shieldify shakes things up with a unique subscription-based auditing model that entitles the customer to unlimited audits within its duration, as well as top-notch service quality thanks to a disruptive 6-layered security approach. The company works with very well-established researchers in the space and has secured multiple millions in TVL across protocols, also can audit codebases written in Solidity, Rust, Go, Vyper, Move and Cairo.

Learn more about us at shieldify.org.

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About Swappee

Swappee is a zero-friction rewards sweeper built for Berachain natives. It's designed for those tired of juggling multiple tokens and wasting time manually claiming validator rewards. Backed by Smilee Finance and powered under the hood by Ooga Booga, Swappee turns dust into wealth — automagically.

Learn more about Smilee's concept and the technicalities behind it here.

4. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1 Impact

- **High** - results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** - results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** - losses will be limited but bearable - and covers vectors similar to griefing attacks that can be easily repaired

4.2 Likelihood

- **High** - almost certain to happen and highly lucrative for execution by malicious actors
- **Medium** - still relatively likely, although only conditionally possible
- **Low** - requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Security Review Summary

The security review was conducted over the span of 2 days by the core Shieldify team.

Overall, the code is well-written. The security review identified two Medium-severity issues related to a potential accounting miscalculations caused by unhandled fee-on-transfer tokens and unsafe transfers. Additionally, two issues of low severity regarding unused variables and duplicate assignments.

5.1 Protocol Summary

Project Name	**Swappee**
Repository	swappee-smart-contracts
Type of Project	DeFi, Proof of Liquidity game
Audit Timeline	2 days
Review Commit Hash	16315aa674ffce54e36fadca66da3cf6785150de
Fixes Review Commit Hash	73f03942932a1b7ebdd6534ac300ba999beac62f

5.2 Scope

The following smart contracts were in the scope of the security review:

File	nSLOC
src/Swappee.sol	149
src/interfaces/ISwappee.sol	42
src/interfaces/external/IBGTIncentiveDistributor.sol	36
src/interfaces/external/IOBRouter.sol	43
Total	**270**

6. Findings Summary

The following number of issues have been identified, sorted by their severity:

- **Medium** issues: **2**
- **Low** issues: **2**
- **Gas** issues: **1**

ID	Title	Severity	Status
[M-01]	Fee-on-Transfer Tokens Can Cause Accounting Errors in Swappee Contract	Medium	Acknowledged
[M-02]	Use `safeTransfer()` and `safeTransferFrom` Instead of `transfer()` and `transferFrom()`	Medium	Fixed
[L-01]	Unused `SWAP_ROLE` Constant Variable	Low	Fixed
[L-02]	Duplicate Assignment of Swap Parameters in `swappee()` and `_swapToken()`	Low	Fixed
[G-01]	Redundant `tokenOut` Parameter in `_swapToken()` Increases Gas Costs	Gas Optimization	Fixed

7. Findings

[M-01] Fee-on-Transfer Tokens Can Cause Accounting Errors in Swappee Contract

Severity

Medium Risk

Description

The `Swappee` contract contains a vulnerability in its handling of Fee-on-Transfer (FOT) tokens. The issue manifests in the swappee function, where the contract assumes the actual received amount of tokens will equal the transferred amount, without accounting for potential transfer fees. Specifically, when processing token transfers via `IERC20(inputToken).transferFrom()`, the contract uses the pre-transfer amount (`amountsClaimedPerWallet[inputToken][msg.sender]`) for all subsequent calculations and approvals, rather than checking the actual received balance. This assumption breaks for FOT tokens, where the received amount may be less than the transferred amount due to built-in transfer fees.

Location of Affected Code

File: src/Swappee.sol#L121

```
function swappee(
    IBGTIncentiveDistributor.Claim[] calldata claims,
    RouterParams[] memory routerParams,
    address tokenOut
)
    public
    invariantCheck
{

// code

    IERC20(inputToken).transferFrom(msg.sender, address(this), amount);

    unchecked {
        amountsClaimedPerWallet[inputToken][msg.sender] -= amount;
    }
    if (routerParam.swapTokenInfo.inputAmount != amount) {
        revert InvalidAmount();
    }
    IERC20(inputToken).approve(aggregator, routerParam.swapTokenInfo.
       inputAmount);

// Override router params to avoid tempered inputs
    routerParam.swapTokenInfo.outputReceiver = address(this);
    routerParam.swapTokenInfo.outputToken = tokenOut;

    uint256 amountOut = _swapToken(
        routerParam.swapTokenInfo,
        routerParam.pathDefinition,
        routerParam.executor,
        routerParam.referralCode,
        tokenOut
    );

// code
}
```

Impact

When users attempt to swap FOT tokens, the contract will: incorrectly calculate fees based on the pre-fee amount, potentially approve more tokens than necessary to the aggregator, and may trigger reverts when attempting to swap the full pre-fee amount that isn't actually available.

Recommendation

The contract should implement proper FOT token handling by: measuring actual received balances before and after transfers, using the delta as the effective amount for swaps.

Team Response

Acknowledged

[M-02] Use `safeTransfer()` and `safeTransferFrom()` Instead of `transfer()` and `transferFrom()`

Severity

Medium Risk

Description

Tokens that do not comply with the ERC20 specification could return false from the transfer function call to indicate the transfer fails, while the calling contract would not notice the failure if the return value is not checked. Checking the return value is a requirement, as written in the EIP-20 specification:

"Callers MUST handle false from returns (bool success). Callers MUST NOT assume that false is never returned!"

Some tokens do not return a bool (e.g. USDT, BNB, OMG) on ERC20 methods. This will make the call break, making it impossible to use these tokens.

Location of Affected Code

File: src/Swappee.sol

```
IERC20(inputToken).transferFrom(msg.sender, address(this), amount);
```

```
IERC20(outputToken).transfer(msg.sender, amountOut);
```

```
IERC20(token).transfer(msg.sender, amount);
```

Impact

It would not revert even though the transaction failed.

Recommendation

Use `SafeTransferLib` or `SafeERC20`, replace transfer with `safeTransfer()` and `transferFrom()` with `safeTransferFrom()` when transferring `ERC20` tokens.

Team Response

Fixed

[L-01] Unused `SWAP_ROLE` Constant Variable

Severity

Low Risk

Description

The `Swappee` contract declares a constant `SWAP_ROLE` that is never utilized anywhere in the contract's functionality. This role is defined with `bytes32 public constant SWAP_ROLE = keccak256("SWAP_ROLE")` but is neither assigned to any addresses nor checked in any modifiers or functions.

Location of Affected Code

File: src/Swappee.sol#L18

```
bytes32 public constant SWAP_ROLE = keccak256("SWAP_ROLE");
```

Impact

While this unused constant doesn't pose a direct security risk, it has several negative implications. First, it unnecessarily increases the contract's deployment cost by adding to the bytecode size. Second, it creates potential confusion for developers and auditors who might expect this role to be functional somewhere in the contract.

Recommendation

The unused SWAP_ROLE constant should be removed from the contract entirely.

Team Response

Fixed

[L-02] Duplicate Assignment of Swap Parameters in `swappee()` and `_swapToken()`

Severity

Low Risk

Description

The `swappee()` function in the Swappee contract redundantly sets `swapTokenInfo.outputReceiver` and `swapTokenInfo.outputToken` twice: - First assignment occurs in `swappee()` before calling `_swapToken()`. - The second assignment happens inside `_swapToken()`.

This results in unnecessary gas consumption since the same storage slots are written multiple times within the same transaction.

Location of Affected Code

File: src/Swappee.sol#L195

```
function _swapToken(
    IOBRouter.swapTokenInfo memory swap,
    bytes memory pathDefinition,
    address executor,
    uint32 referralCode,
    address tokenOut
)
    internal
    returns (uint256)
{
// @audit can remove this as set in earlier function
    swap.outputReceiver = address(this);
    swap.outputToken = tokenOut;
    return IOBRouter(aggregator).swap(swap, pathDefinition, executor,
       referralCode);
}
```

File: src/Swappee.sol#L134

```
routerParam.swapTokenInfo.outputReceiver = address(this);
routerParam.swapTokenInfo.outputToken = tokenOut;

uint256 amountOut = _swapToken(
    routerParam.swapTokenInfo,
    routerParam.pathDefinition,
    routerParam.executor,
    routerParam.referralCode,
    tokenOut
);
```

Impact

Unnecessary duplicate assignment of swap parameters in `swappee()` and `_swapToken()`

Recommendation

Remove the duplicate assignments from one of the functions

Team Response

Fixed

[G-01] Redundant `tokenOut` Parameter in `_swapToken()` Increases Gas Costs

Severity

Gas Optimization

Description

In the `swappee()` function, the internal `_swapToken()` call includes a `tokenOut` parameter that is passed explicitly:

```
uint256 amountOut = _swapToken(
    routerParam.swapTokenInfo,
    routerParam.pathDefinition,
    routerParam.executor,
    routerParam.referralCode,
    tokenOut
);
```

However, this parameter is redundant because the output token is already encapsulated within the `swapTokenInfo.outputToken`. By including this extraneous argument, the function introduces an avoidable overhead in calldata and computation. This redundancy not only increases the transaction's calldata size but also leads to higher gas consumption.

Location of Affected Code

File: src/Swappee.sol

Impact

While this issue does not compromise the protocol's integrity or lead to a loss of funds, it does cause users and integrators to pay higher-than-necessary gas fees.

Recommendation

To address this inefficiency, the `tokenOut` parameter should be removed from the `_swapToken()` call. The swap logic can directly rely on the `outputToken` embedded in `swapTokenInfo`, which already defines the intended asset flow. The corrected call should appear as follows:

```
uint256 amountOut = _swapToken(
    routerParam.swapTokenInfo,
    routerParam.pathDefinition,
    routerParam.executor,
    routerParam.referralCode
-   tokenOut
);
```

Team Response

Fixed

our shielding · Your smart contracts, our shielding · Your smart c

shieldify

Thank you!