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

shieldify

Berabot

SECURITY REVIEW

Date: 27 January 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 Berabot

BerabotRouter.sol with ReferralSystem.sol

- **Purpose**: The BerabotRouter contract serves as a decentralized exchange (DEX) aggregator, interfacing with both UniswapV2 and UniswapV3 protocols. It facilitates the buying and selling of tokens whose liquidity pools (LPs) are whitelisted by our team. The contract is designed to collect a percentage of fees on both buy and sell transactions. These fees are deducted from the ETH amount during a purchase or from the ETH received during a sale.
- **Features**:
  - **DEX Aggregation**: Supports swaps across UniswapV2 and UniswapV3, allowing users to find the best rates.
  - **Fee Collection**: Implements a fee mechanism that collects a percentage of the transaction value, enhancing the protocol's revenue model.
  - **Referral System**: Integrates a referral system that rewards users for bringing new participants to the platform.
  - **Upgradability**: The contract is upgradable, allowing for future enhancements and bug fixes. Each logical update will be followed by a new audit to ensure security and functionality.

Berabot.sol

- **Purpose**: The Berabot contract is the ERC20 token contract for the Berabot token. It includes mechanisms for buy and sell taxes, which are used to fund the protocol's operations and development.
- **Features**:
  - **Buy/Sell Tax**: Implements a tax on transactions to generate revenue for the protocol.
  - **Max Wallet Limiter**: Initially, a limiter is in place to prevent users from purchasing a large percentage of the LP, ensuring fair distribution. This limiter will be removed after the initial launch phase.

- **Fee Whitelisting**: Certain addresses can be whitelisted to bypass transaction fees, providing flexibility for specific use cases or partners.

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 lasted 4 days with a total of 128 hours dedicated by 4 researchers from the Shieldify team.

Overall, the code is well-written. The audit report identified one high-severity issue, five medium-severity issues, and several low-severity vulnerabilities, along with informational recommendations. The vulnerabilities primarily stem from misconfigurations, fee theft risks, and ineffective implementation of referral mechanics.

The Berabot team has been highly responsive to the Shieldify research team's inquiries and promptly implemented all recommendations.

5.1 Protocol Summary

Project Name	**Berabot**
Repository	berabot-product
Type of Project	DEX Aggregator , Uniswap V2/V3
Audit Timeline	4 days
Review Commit Hash	6f1f6d446eeccc7f49d494be45db93e70e998bc0
Fixes Review Commit Hash	0b1fa0d93abc93c3b7c9d9ed7108cfe2d855282e

5.2 Scope

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

File	nSLOC
src/Berabot.sol	208
src/BerabotRouter.sol	318
src/ReferralSystem.sol	66
Total	**592**

6. Findings Summary

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

- **Critical & High** issues: **1**
- **Medium** issues: **5**
- **Low** issues: **3**
- **Informational** issues: **5**

ID	**Title**	**Severity**	**Status**
[H-01]	`BerabotRouter` Calculation for Paths Longer Than Two Is Broken	High	Fixed
[M-01]	Slippage During Transfer Fee Swap	Medium	Acknowledged
[M-02]	Users Can Always Grief the Protocol	Medium	Acknowledged
[M-03]	Not Being Able To Manage Configurations Appropriately Allows Users to Avoid Fees	Medium	Fixed
[M-04]	Berachain Users Cannot Use Flashloans from Uni V2 `WBERA/BerabotToken` Pool	Medium	Acknowledged
[M-05]	`BerabotRouter` Cannot Make Swaps with V3 Pools with Fee Different than 500	Medium	Fixed
[L-01]	Referrals Might Be Able to Claim Their Fees When Not Supposed to	Low	Fixed
[L-02]	Ineffective Referral Mechanics	Low	Acknowledged
[L-03]	In `transfer()` the `swapSellFees` and `swapBuyFees` Are Swapped	Low	Fixed
[I-01]	`BerabotRouter` Can Be Initialised By An Attacker	Info	Acknowledged
[I-02]	`ReferralSystem` Does Not Have Storage Gaps	Info	Fixed
[I-03]	Functions Not Emitting Events	Info	Acknowledged
[I-04]	Usage of Hardcoded Values and Magic Numbers	Info	Acknowledged
[I-05]	Typography	Info	Fixed

7. Findings

[H-01] `BerabotRouter` Calculation for Paths Longer Than Two Is Broken

Severity

High Risk

Description

The `BerabotRouter` mimics the functionality of the Uniswap V2 router when a V2 router is selected for the swaps.

Swaps between tokens that do not have a pool together are supported by defining a path for the swap. Example: `WETH -> Doge -> DAI` defined in function arguments as `path[0]`, `path[1]` and `path[2]`.

```
function _executeV2Swap(
    uint256 amountIn,
    address[] calldata path,
    address router
)
    internal
    virtual
    returns (uint256 amountOut)
{
    address _router = router;
    for (uint256 i = 0; i < path.length - 1; i++) {
        address tokenIn = path[i];
        address tokenOut = path[i + 1];

        address pair = _calculatePairAddress(tokenIn, tokenOut, _router);
        if (pair.code.length == 0) revert PairDoesNotExist();

        (uint256 reserveIn, uint256 reserveOut, bool zeroForOne) =
           _getReservesAndDirection(pair, path[i], path[i + 1]);

@>      amountOut = _calculateAmountOut(reserveIn, reserveOut, amountIn,
   _router);
```

```
        address to = i < path.length - 2 ? _calculatePairAddress(tokenOut
            , path[i + 2], _router) : address(this);

        IPair(pair).swap(
            zeroForOne ? 0 : amountOut, // amount0Out
            zeroForOne ? amountOut : 0, // amount1Out
            to, // to
            new bytes(0) // data
        );
    }
}
```

Uniswap V2 swap function is optimistic and it needs to be provided with the `amountOut` value that will send to the `to` address. How much `amountOut` should be is calculated inside `_calculateAmountOut()` where the function is provided with the pool reserves, the `amountIn` and the router address.

The issue is that when we loop through the paths we use the same `amountIn` value that represents the amount in of the first swap (that is WETH (`path[0]`) towards Doge `path[1]`).

During the second swap Doge -> DAI the amount out calculate function will be provided with the wrong `amountIn` amount:

calculateAmountOut(reserveInDoge, reserveInDAI, amountInWETH (instead of amountIn-Doge), router)

Impact

Users will not be able to use the multi-path functionality because it will revert either in the pair swap function or when amountOutMin verification is made.

Recommendation

Consider applying the following changes:

```
function _executeV2Swap(
    uint256 amountIn,
    address[] calldata path,
    address router
)
    internal
    virtual
    returns (uint256 amountOut)
{
    address _router = router;
+   uint256 _amountIn = amountIn;
```

```
    for (uint256 i = 0; i < path.length - 1; i++) {
        address tokenIn = path[i];
        address tokenOut = path[i + 1];

        address pair = _calculatePairAddress(tokenIn, tokenOut, _router);
        if (pair.code.length == 0) revert PairDoesNotExist();

        (uint256 reserveIn, uint256 reserveOut, bool zeroForOne) =
           _getReservesAndDirection(pair, path[i], path[i + 1]);

-       amountOut = _calculateAmountOut(reserveIn, reserveOut, amountIn,
   _router);
+       amountOut = _calculateAmountOut(reserveIn, reserveOut, _amountIn,
    _router);

        address to = i < path.length - 2 ? _calculatePairAddress(tokenOut
           , path[i + 2], _router) : address(this);

        IPair(pair).swap(
            zeroForOne ? 0 : amountOut, // amount0Out
            zeroForOne ? amountOut : 0, // amount1Out
            to, // to
            new bytes(0) // data
        );
+       _amountIn = amountOut;
    }
}
```

Team Response

Fixed.

[M-01] Slippage During Transfer Fee Swap

Severity

Medium Risk

Description

If `Berabot` contains at least 0.02% of the `totalSupply` then a swap can be made during `transfer()`. Such swaps can be vulnerable to sandwich attacks which can cause the `feeRecipient` to receive way less `WBERA` than the normal market price for the swapped tokens.

This is possible in 2 scenarios:

- Every fee-whitelisted address can perform the sandwich attack since they will not trigger an internal swap themselves when they increase the balance of the pool.
- If the `Berabot` balance is larger than 0.04%+ or 0.4%+ then again the front-running swap will be executed as normal but the second swap will cause the `feeRecipient` to get far fewer native tokens than expected.

Impact

Token `feeRecipient` fee value can be extracted by an attacker.

Recommendation

It is a design decision. One way to fix this is by separating the swapping of the tokens from the `transfer()` function and implementing specifying minimum amount out.

Team Response

Acknowledged.

[M-02] Users Can Always Grief the Protocol

Severity

Medium Risk

Description

Users can reduce the amount of fees that go to the protocol by always specifying a referral address that is not able to claim these fees. Such addresses could be `address(0xdead)`, etc.

Impact

Protocol fees will be reduced and the reduced amount will not be accessible anymore.

Recommendation

Consider redirecting fees to referrals only if they have some input with the protocol, either staking or other action.

Team Response

Acknowledged.

[M-03] Not Being Able To Manage Configurations Appropriately Allows Users to Avoid Fees

Severity

Medium Risk

Description

The `isAmmPair` variable is used to track the AMMs in order for the Berabot token to know when to apply fees on transfers from and to such pools. The uniswap V2 WBERA/BerabotToken pool is the one being set in the constructor but there are no other functions that can add other pools to the `isAmmPair`.

Users can decide to avoid this pool and use another WBERA/BerabotToken pool for their swaps to not pay fees to the protocol.

On a separate note:

- Addresses can be added to the fee whitelist (`_feesWhitelisted`) but cannot be removed.
- `_isMaxWalletExcluded` can't be updated outside of the constructor.
- Trading can be turned on but not stopped (`isTradingOpen`) and `limitMaxWallet` can be turned off but not on after that.
- In BerabotRouter, `addRouterV2()` and `addRouterV3()` once the configurations are set they cannot be changed. If they are set incorrectly the contract might need to be upgraded in order to change the configs.

Impact

Users can avoid paying fees by using different pools.

Once set many configurations cannot be changed without needing an upgrade or redeploy. If the token contract is already widely used redeploying might not be possible.

Recommendation

It is recommended that all settings be adjustable by the owner.

Team Response

Fixed.

[M-04] Berachain Users Cannot Use Flashloans from Uni V2 `WBERA/BerabotToken` Pool

Severity

Medium Risk

Description

`Berabot` automatically swaps accumulated fees for `WBERA` in the `transfer()` function by calling the `swapBack()` function when specific conditions are met.

```
if (
    canSwap && // contract balance is above or equal 0.02% of total
       supply
    !swapping && // this is a normal transfer not caused by a inner fee
       swap by `swapBack()`
    !isAmmPair[from] && // `from` must not be the swap pool
    !_feesWhitelisted[from] && // `from` must not be whitelisted
    !_feesWhitelisted[to] // `to` must not be whitelisted
) {
    swapping = true;
    swapBack();
    swapping = false;
}
```

This mechanism is designed to ensure that accumulated fees are swapped into `WBERA` automatically under normal transfer conditions. However, this implementation introduces an unintended vulnerability that blocks users from utilizing flash loans via the `WBERA/BerabotToken` liquidity pool.

In Uniswap V2, users can take flash loans by calling the swap function on a pool contract. The swap function allows users to borrow tokens as long as they return an equivalent value by the end of the transaction. To prevent reentrancy attacks, Uniswap V2 employs a reentrancy lock mechanism that ensures the swap function cannot be called recursively during execution.

```
function swap(uint amount0Out, uint amount1Out, address to, bytes
    calldata data) external lock { // <----
     // code

     {
         // code

         if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out); //
            optimistically transfer tokens
         if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out); //
            optimistically transfer tokens
@>       if (data.length > 0) IUniswapV2Callee(to).uniswapV2Call(msg.
    sender, amount0Out, amount1Out, data);

         // code

         require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0)
            .mul(_reserve1).mul(1000**2), 'UniswapV2: K');
     }

     // code
}
```

Example:

If a user attempts to take a flash loan of `BerabotToken` from the `WBERA/BerabotToken` pool and then transfers these tokens to another address within the same transaction (e.g., for arbitrage or liquidation purposes), the following occurs:

1. The Uniswap V2 swap function is called, initiating the flash loan.
2. The flash-loaned `BerabotToken` is transferred to the user's address.
3. During this transfer, the `BerabotToken` contract checks whether it should trigger `swapBack()` to convert accumulated fees to WBERA.
4. Since the conditions for `swapBack()` are met (e.g., sufficient contract balance, non-whitelisted sender and receiver, and normal transfer), the `swapBack()` function is called.
5. Inside `swapBack()`, the contract attempts to execute a swap on the `WBERA/BerabotToken` pool by calling `pool.swap(...)`.
6. Uniswap V2's reentrancy lock mechanism prevents `pool.swap(...)` from being called recursively within the same transaction, causing the entire transaction to revert.

Impact

Users are unable to utilize flash loans of the BerabotToken by using the hardcoded token `WBERA/BerabotToken` pool.

Recommendation

Depends on the design decisions the developer wishes to implement.

Team Response

Acknowledged.

[M-05] `BerabotRouter` Cannot Make Swaps with V3 Pools with Fee Different than 500

Severity

Medium Risk

Description

The issue lies in the `uniswapV3SwapCallback()` function:

```
function uniswapV3SwapCallback(int256 amount0Delta, int256 amount1Delta,
   bytes calldata data) external virtual {
    if (amount0Delta <= 0 && amount1Delta <= 0) revert V3InvalidSwap();

    assembly {
        if iszero(data.length) {
            mstore(0x20, amount0Delta)
            mstore(0x40, amount1Delta)
            revert(0x20, 64)
        }
    }

    (address tokenIn, address tokenOut, address payer) = abi.decode(data,
        (address, address, address));

    bool isValid = false;

    for (uint256 i = 0; i < whitelistedRouterV3.length; i++) {
        address currentRouter = whitelistedRouterV3[i];
@>      address pool = _calculateV3PoolAddress(currentRouter, tokenIn,
   tokenOut, 500); // @audit hardcoded pool fee
        if (msg.sender == pool) {
            isValid = true;
            break;
        }
    }

    if (!isValid) revert V3InvalidCaller();
    uint256 amountToPay = uint256(amount0Delta > 0 ? amount0Delta :
       amount1Delta);

    if (tokenIn == WBERA) {
        IWETH(WBERA).deposit{ value: amountToPay }();
        TransferHelper.safeTransfer(WBERA, msg.sender, amountToPay);
    } else {
        TransferHelper.safeTransferFrom(tokenIn, payer, msg.sender,
           amountToPay);
    }
}
```

This causes the incorrect pool to be calculated (for V3 pools with a fee different than 500) and will cause the swap to revert. Moreover the `getAmountOutAndRouter()` function loops over V3 router fee versions which implies that fees will differ.

Impact

The router does not support V3 swaps other than pools with a fee of 500.

Recommendation

Provide the correct fee variable. Take inspiration from how the Uniswap V3 Router does it.

Team Response

Fixed.

[L-01] Referrals Might Be Able to Claim Their Fees When Not Supposed to

Severity

Low Risk

Description

In case the `requiredToken` in ReferralSystem is changed, there might be a window where referrals could be able to claim their fees when they are not supposed to.

Example:

The required token is ETH and `minimumBalance` is 1e18 but then the required token is changed to USDC and users have the opportunity to claim their referral rewards since 1 ETH costs thousands of dollars in value and 1 USDC is 1 dollar. The difference in value can make obtaining the required amount far easier.

The same statement could be made also if the initial required token is in 6 decimals and the new token is in 18. The minimal required amount the referral must hold in this case is going to be in incorrect decimals.

Impact

Referrals could be presented with the opportunity to claim their pending fees by getting an incorrect required amount of tokens.

Recommendation

Consider having 1 setter function for both `requiredToken` and `minimumBalance`.

Team Response

Fixed.

[L-02] Ineffective Referral Mechanics

Severity

Low Risk

Description

Referrals need a to have certain amount of the required tokens in order to claim their fees. This is not a very effective mechanism since they can just use a loan to get the amount or buy and immediately sell after claiming the fees.

Also, multiple referrals can transfer the required token amount between them in order to claim their fees which does not benefit the protocol in any way.

Impact

Referrals can game the system without the protocol being able to retain value.

Recommendation

Consider redirecting percent of the fee rewards to referrals only if they stake the required token amount for a specific period of time in order to make the required token more valuable.

Team Response

Acknowledged.

[L-03] In `_transfer()` the `swapSellFees` and `swapBuyFees` Are Swapped

Severity

Low Risk

Description

Whenever users buy or sell tokens via a pool recorded in `isAmmPair` they pay a fee.

In cases where the `from` address is the pool, this means that the user is buying the token for the exchange of another token. Example: the user sends WETH and receives a token.

In the opposite case, the user sends the token to the pool (`isAmmPair[to]`) and receives WETH, effectively selling the token.

However, when a user buys (`isAmmPair[from]`) he is charged the `swapSellFees` and when a user sells he is charged the `swapBuyFees` . Initially, both are set to 3% but in the future, if the owner wants to direct users via increasing/decreasing fees in either direction he could get the opposite effect than the one he wanted.

```
    if (
@>      isAmmPair[from] &&
@>      swapSellFees > 0
    ) {
        // from pool
        // fees more than 0
        fees = amount * swapSellFees / 1000;

    } else if (
@>      isAmmPair[to] &&
@>      swapBuyFees > 0
    ) {
        // to pool
        // fees more than 0
        fees = amount * swapBuyFees / 1000;
    }
```

Impact

Fees are swapped and can cause confusion for the owner and its users.

Recommendation

Swap the positions of the fees (where they are used) inside `transfer()` .

Team Response

Fixed.

[I-01] `BerabotRouter` Can Be Initialised By An Attacker

Severity

Informational

Description

The `BerabotRouter` uses the UUPS upgradability pattern. Since `_disableInitialisers()` is not called in the constructor there is a risk that the implementation contract can be initialised by a malicious user, set himself as the owner and upgrade the implementation.

Impact

An attacker can set himself as the owner and upgrade the implementation.

Recommendation

Consider adding a constructor with `_disableInitialisers()` inside of it.

Team Response

Fixed.

[I-02] `ReferralSystem` Does Not Have Storage Gaps

Severity

Informational

Description

`ReferralSystem` is an abstract contract inherited by `BerabotRouter`. In the case of upgrading the implementation of the `ReferralSystem` contract and introducing new variables, this might not be possible without overwriting existing storage variables due to the absence of storage gaps.

Impact

Missing storage gaps can make upgrading `ReferralSystem` difficult or cause overwriting of existing storage variables.

Recommendation

Introduce storage gaps.

Team Response

Acknowledged.

[I-03] Functions Not Emitting Events

Severity

Informational

Description

Through out the codebase, many state-changing functions are not emitting events. It is considered a good coding practice to have them in order for external systems to be able to track state changes or execute logic.

Impact

External systems can't track state changes or execute logic based on changes.

Recommendation

Consider implementing events for every state-changing function.

Team Response

Acknowledged.

[I-04] Usage of Hardcoded Values and Magic Numbers

Severity

Informational

Description

Hardcoded addresses are considered a bad coding practice since addresses on one chain can differ from another chain. The most relevant example would be a contract having a different address on testnet and mainnet.

Also, each hardcoded number used throughout the codebase should be implemented as constants for better readability and avoiding coding mistakes.

Impact

Addresses on different chains can differ.

Recommendation

Consider implementing the suggestions mentioned above.

Team Response

Acknowledged.

[I-05] Typography

Description

- In BerabotRouter `FeeTierAlreadyExists`, `InsufficientFees`, `SwapFailed`, `ExcessiveOutputAmount`, `FailedToSendEther`, `V3TooMuchRequested` errors are not used.

- In ReferralSystem `TransferFailed` error is not used.
- In `BerabotRouter._calculateAmountOut()` code and comments do not match: \codex{fee = 9970; // Beraswap uses 0.25% LP fee}

Recommendation

Consider fixing the topics mentioned above.

Team Response

Fixed.

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

shieldify

Thank you!