SHELLBOXES

Crowdswap

Smart Contract Security Audit

Prepared by ShellBoxes

January 18th, 2023 – January 24th, 2023

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	Crowdswap
Version	1.0
Classification	Public

Scope

Repository	Commit Hash
https://github.com/CrowdSwap/Opportunities	e2793ca83555520b2945cfd57055d1c14bca9df0

Re-Audit

Repository	Commit Hash
https://github.com/CrowdSwap/Opportunities	82fcabc6f691df24760edaf6aed0ab537b2bcce0

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction 5

1.1 About Crowdswap 5

1.2 Approach & Methodology 5

1.2.1 Risk Methodology 6

2 Findings Overview 7

2.1 Summary 7

2.2 Key Findings 7

3 Finding Details 9

SHB.1 The Investor Can Lose a Part Of His Provided Liquidity 9

SHB.2 The Investor Can Lose a Part Of his Swapped Tokens 14

SHB.3 Rounding Error Can Prevent the Stakers From Getting the Rewards 17

SHB.4 The Fees Can Be Bypassed 19

SHB.5 Possible Desynchronization Between tokenA, tokenB and pair 21

SHB.6 Front-run In The Contract's Initialization 23

SHB.7 The Investors' Native Tokens Can Get Locked 25

SHB.8 Missing Value Verification 27

SHB.9 Missing Address Verification 30

SHB.10 Potential Denial of Service (DoS) and Compatibility Issues Due to Use of transfer For Sending Ether 31

SHB.11 Missing User Address Validation in Investment Functions 34

4 Best Practices 36

BP.1 Utilize more expressive error messages 36

BP.2 Remove unnecessary initializations 36

BP.3 Usage of pre-increment 37

BP.4 Public Function Can Be Called External 37

BP.5 Remove tautology 39

5 Tests 41

6 Conclusion 51

7 Scope Files 52

7.1 Audit 52
7.2 Re-Audit 53

8 Disclaimer 55

1 Introduction

Crowdswap engaged ShellBoxes to conduct a security assessment on the Crowdswap beginning on January 18th, 2023 and ending January 24th, 2023. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About Crowdswap

CrowdSwap is a cross-chain opportunity optimization and automation platform. It aims to reach mass adoption in crypto for every human being and overcome actual problems that reside from a fast-growing business space like DeFi.

Issuer	Crowdswap
Website	`https://crowdswap.org`
Type	Solidity Smart Contract
Whitepaper	`https://crowdswap.org/wp-content/uploads/2022/08/CrowdSwapWhitepaper.pdf`
Audit Method	Whitebox

1.2 Approach & Methodology

ShellBoxes used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

1.2.1 Risk Methodology

Vulnerabilities or bugs identified by ShellBoxes are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.

Impact	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low

Likelihood

2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the Crowdswap implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include 1 critical-severity, 3 high-severity, 2 medium-severity, 5 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
SHB.1. The Investor Can Lose a Part Of His Provided Liquidity	CRITICAL	Fixed
SHB.2. The Investor Can Lose a Part Of his Swapped Tokens	HIGH	Fixed
SHB.3. Rounding Error Can Prevent the Stakers From Getting the Rewards	HIGH	Fixed
SHB.4. The Fees Can Be Bypassed	HIGH	Acknowledged
SHB.5. Possible Desynchronization Between tokenA, tokenB and pair	MEDIUM	Fixed
SHB.6. Front-run In The Contract's Initialization	MEDIUM	Fixed
SHB.7. The Investors' Native Tokens Can Get Locked	LOW	Fixed

SHB.8. Missing Value Verification	LOW	Fixed
SHB.9. Missing Address Verification	LOW	Fixed
SHB.10. Potential Denial of Service (DoS) and Compatibility Issues Due to Use of transfer For Sending Ether	LOW	Fixed
SHB.11. Missing User Address Validation in Investment Functions	LOW	Fixed

3 Finding Details

SHB.1 The Investor Can Lose a Part Of His Provided Liquidity

- Severity : CRITICAL
- Status : Fixed
- Likelihood : 3
- Impact : 3

Description:

The opportunity contracts allow an investor to provide liquidity for a liquidity pool either by using tokenA and tokenB by calling the investByTokenATokenB function, or by using one of the two tokens by calling the investByTokenAOrTokenB function, or by using a third token by calling the investByToken function. The investor specifies the amounts desired to be invested in the _addLiqDescriptor argument of type AddLiqDescriptor, specifically in the amountADesired and amountBDesired fields. These functions add these deposited amounts as liquidity to an AMM router after taking the addLiquidityFee and the stakeFee. However, the AMM router does not take as liquidity the amountADesired and amountBDesired, instead it takes one amount of those and it calculates the optimal amount to be added to the other side of the liquidity pool. Therefore, an amount that is equal to amountXDesired - amountX is not added to liquidity pool and not refunded to the investor, then it can be considered as lost from the investor's end.

Exploit Scenario:

Let's consider, for example, a liquidity pool in UniswapV2 that contains 100 A tokens and 100 B tokens:

- The user specifies 10 as amountADesired and 50 as amountBDesired.
- Using these values and considering the logics of the UniswapV2Router02, the addLiquidity call will return 10 as amountA and 10 as amountB, in addition to the minted LP tokens.

- The investor spent 10 A tokens and 50 B tokens, and he only provided 10 A tokens and 10 B tokens to the liquidity pool, which is also what he will be able to withdraw when he calls the leave function in order to remove the liquidity and get back his funds.

As we can see, in this scenario, the investor lost 40 B tokens, and this amount can change based on the desired amounts specified by the investor and also the amount existing in each side of the liquidity pool.

Files Affected:

SHB.1.1: Opportunity.sol

```
function _addLiquidity(
    AddLiqDescriptor memory _addLiqDescriptor
) private returns (uint256) {
    uint256 balanceTokenA = tokenA.uniBalanceOf(address(this));
    require(balanceTokenA >= _addLiqDescriptor.amountADesired, "oe13");
    uint256 balanceTokenB = tokenB.uniBalanceOf(address(this));
    require(balanceTokenB >= _addLiqDescriptor.amountBDesired, "oe14");

    uint256 _beforeBalance = pair.uniBalanceOf(address(this));
    (uint256 amountA, uint256 amountB, uint256 liquidity) = addLiquidity
        ↪ (_addLiqDescriptor);
    require(liquidity > 0, "oe10");
    uint256 _afterBalance = pair.uniBalanceOf(address(this));
    require(_afterBalance - _beforeBalance == liquidity, "oe06");
    emit AddedLiquidity(msg.sender, amountA, amountB, liquidity);
    return liquidity;
}
```

SHB.1.2: BeefyMimaticUsdcOpportunity.sol

```
function addLiquidity(
    AddLiqDescriptor memory _addLiqDescriptor
) internal override returns (uint256, uint256, uint256) {
    // gas savings
    IUniswapV2Router02 _router = router;
    tokenA.uniApprove(address(_router), _addLiqDescriptor.amountADesired
       ↪ );
    tokenB.uniApprove(address(_router), _addLiqDescriptor.amountBDesired
       ↪ );
    return _router.addLiquidity(
        address(tokenA),
        address(tokenB),
        _addLiqDescriptor.amountADesired,
        _addLiqDescriptor.amountBDesired,
        _addLiqDescriptor.amountAMin,
        _addLiqDescriptor.amountBMin,
        address(this),
        _addLiqDescriptor.deadline
    );
}
```

SHB.1.3: CrowdUsdtLpStakeOpportunity.sol

```
function addLiquidity(
    AddLiqDescriptor memory _addLiqDescriptor
) internal override returns (uint256, uint256, uint256) {
    IUniswapV2Router02 _router = router; // gas savings
    tokenA.uniApprove(address(_router), _addLiqDescriptor.amountADesired
       ↪ );
    tokenB.uniApprove(address(_router), _addLiqDescriptor.amountBDesired
       ↪ );
    return _router.addLiquidity(
        address(tokenA),
        address(tokenB),
        _addLiqDescriptor.amountADesired,
        _addLiqDescriptor.amountBDesired,
        _addLiqDescriptor.amountAMin,
        _addLiqDescriptor.amountBMin,
        address(this),
        _addLiqDescriptor.deadline
    );
}
```

SHB.1.4: PancakeOpportunity.sol

```
function addLiquidity(AddLiqDescriptor memory _addLiqDescriptor)
    internal
    override
    returns (
        uint256,
        uint256,
        uint256
    )
{
    IUniswapV2Router02 _router = router; // gas savings
    tokenA.uniApprove(address(_router), _addLiqDescriptor.amountADesired
        ↪ );
    tokenB.uniApprove(address(_router), _addLiqDescriptor.amountBDesired
        ↪ );
    return
        _router.addLiquidity(
            address(tokenA),
            address(tokenB),
            _addLiqDescriptor.amountADesired,
            _addLiqDescriptor.amountBDesired,
            _addLiqDescriptor.amountAMin,
            _addLiqDescriptor.amountBMin,
            address(this),
            _addLiqDescriptor.deadline
        );
}
```

SHB.1.5: UniswapV2Router02.sol

```
function _addLiquidity(
    address tokenA,
    address tokenB,
    uint amountADesired,
    uint amountBDesired,
    uint amountAMin,
    uint amountBMin
) internal virtual returns (uint amountA, uint amountB) {
    // create the pair if it doesn't exist yet
    if (IUniswapV2Factory(factory).getPair(tokenA, tokenB) == address(0) ) {
        IUniswapV2Factory(factory).createPair(tokenA, tokenB);
    }
    (uint reserveA, uint reserveB) = UniswapV2Library.getReserves( factory, tokenA, tokenB);
    if (reserveA == 0 && reserveB == 0) {
        (amountA, amountB) = (amountADesired, amountBDesired);
    } else {
        uint amountBOptimal = UniswapV2Library.quote(amountADesired, reserveA, reserveB);
        if (amountBOptimal <= amountBDesired) {
            require(amountBOptimal >= amountBMin, 'UniswapV2Router: INSUFFICIENT_B_AMOUNT');
            (amountA, amountB) = (amountADesired, amountBOptimal);
        } else {
            uint amountAOptimal = UniswapV2Library.quote(amountBDesired, reserveB, reserveA);
            assert(amountAOptimal <= amountADesired);
            require(amountAOptimal >= amountAMin, 'UniswapV2Router: INSUFFICIENT_A_AMOUNT');
            (amountA, amountB) = (amountAOptimal, amountBDesired);
        }
    }
}
function addLiquidity(
    address tokenA,
    address tokenB,
    uint amountADesired,
    uint amountBDesired,
    uint amountAMin,
    uint amountBMin,
    address to,
    uint deadline
) external virtual override ensure(deadline) returns (uint amountA, uint amountB, uint liquidity) {
    (amountA, amountB) = _addLiquidity(tokenA, tokenB, amountADesired, amountBDesired, amountAMin, amountBMin);
    address pair = UniswapV2Library.pairFor(factory, tokenA, tokenB);
    TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA);
    TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB);
    liquidity = IUniswapV2Pair(pair).mint(to);
}
```

Recommendation:

Consider sending back the non-invested funds to the investor, this amount will be equal to amountBDesired - amountB when amountADesired equals to amountA, and amountADesired - amountA when amountBDesired equals to amountB.

Updates

The Crowdswap team resolved the issue by returning the left tokens to the investor using the refund modifier.

SHB.2 The Investor Can Lose a Part Of his Swapped Tokens

- Severity : HIGH
- Status : Fixed
- Likelihood : 2
- Impact : 3

Description:

The opportunity contracts allow an investor to provide liquidity for a liquidity pool either using tokenA and tokenB by calling the investByTokenATokenB function, or using one of the two tokens by calling the investByTokenAOrTokenB function, or using a third token by calling the investByToken function. In the case of the investByTokenAOrTokenB and the investByToken functions, there is a need to swap a part of the tokens to either tokenA or tokenB, then the amountOut is verified to be higher than the desired amount. However, the liquidity will be added only based on the desired amounts specified in the arguments, which means the amountOut - amountXDesired where X is either A or B will be lost from the investor's end.

Files Affected:

SHB.2.1: Opportunity.sol

```
function investByTokenAOrTokenB(
    address _userAddress,
    IERC20Upgradeable _token,
    uint256 _amount,
    uint256 _secondAmount,
    AddLiqDescriptor memory _addLiqDescriptor,
    bytes calldata _swapData
) external whenNotPaused {
    IERC20Upgradeable _tokenA = tokenA; // gas savings
    IERC20Upgradeable _tokenB = tokenB; // gas savings
    require(_token == _tokenA  _token == _tokenB, "oe04");

    _transferFrom(_token, _amount);

    emit InvestedByTokenAOrTokenB(_userAddress, address(_token), _amount
        ↪ );

    uint256[] memory _fees = new uint256[](2);
    _fees[0] = addLiquidityFee;
    _fees[1] = stakeFee;
    uint256 _totalFee = _deductFee(_fees, _token, _amount);
    _amount = _amount - _totalFee;

    uint256 _amountOut = _swap(_token, _token == _tokenA ? _tokenB :
        ↪ _tokenA, _secondAmount, _swapData); // [+] to verify how much
        ↪ to swap ?
    require(_amountOut >= (_token == _tokenA ? _addLiqDescriptor.
        ↪ amountBDesired : _addLiqDescriptor.amountADesired), "oe01");

    uint256 _liquidity = _addLiquidity(_addLiqDescriptor);
    _stake(_userAddress, _liquidity);
}
```

SHB.2.2: Opportunity.sol

```
function investByToken(
    address _userAddress,
    IERC20Upgradeable _token,
    uint256 _amount,
    uint256 _secondAmount,
    AddLiqDescriptor memory _addLiqDescriptor,
    bytes calldata _swapDataToB,
    bytes calldata _swapDataToA
) external payable whenNotPaused {
    if (_token.isETH()) {
        require(msg.value >= _amount, "oe03"); /
    } else {
        _transferFrom(_token, _amount);
    }

    emit InvestedByToken(_userAddress, address(_token), _amount);

    uint256[] memory _fees = new uint256[](2);
    _fees[0] = addLiquidityFee;
    _fees[1] = stakeFee;
    uint256 _totalFee = _deductFee(_fees, _token, _amount);
    _amount = _amount - _totalFee;

    uint256 _amountOut = _swap(_token, tokenB, _amount, _swapDataToB);
    require(_amountOut >= _addLiqDescriptor.amountBDesired +
        ↪ _secondAmount, "oe01");

    _amountOut = _swap(tokenB, tokenA, _secondAmount, _swapDataToA);
    require(_amountOut >= _addLiqDescriptor.amountADesired, "oe02");

    uint256 _liquidity = _addLiquidity(_addLiqDescriptor);
    _stake(_userAddress, _liquidity);
}
```

Recommendation:

Consider updating the amountXDesired attribute to _amountOut where X is the swapped to token.

Updates

The Crowdswap team resolved the issue by returning the left tokens to the investor using the refund modifier.

SHB.3 Rounding Error Can Prevent the Stakers From Getting the Rewards

- Severity : HIGH
- Status : Fixed
- Likelihood : 2
- Impact : 3

Description:

The notifyRewardAmount function is used by the owner in order to set the rewardRate that will be used to calculate the users' rewards. However, if the reward argument is lower than the rewardsDuration, the rewardRate variable will round to zero. Therefore, the users will not be able to get their rewards even if the contract is funded, the likelihood of the issue gets higher for long staking periods.

Files Affected:

SHB.3.1: StakingLP.sol

```
function notifyRewardAmount(uint256 reward) external onlyOwner updateReward(address(0)) {
    if (block.timestamp >= periodFinish) {
        rewardRate = reward / rewardsDuration;
        _rewards = reward;
    } else {
        uint128 currentTime =
            startTime >= block.timestamp ? startTime : uint128(block.timestamp);
        uint128 remaining = periodFinish - currentTime;
        uint256 leftover = remaining * rewardRate;
        rewardRate = (reward + leftover) / rewardsDuration;
        _rewards = reward + leftover;
    }
    // Ensure the provided reward amount is not more than the balance in the contract.
    // This keeps the reward rate in the right range, preventing overflows due to
    // very high values of rewardRate in the earned and rewardsPerToken functions;
    // Reward + leftover must be less than 2^256 / 10^18 to avoid overflow.
    uint balance = rewardToken.balanceOf(address(this));
    require(rewardRate <= balance / rewardsDuration, "LPStaking:
        ↪ Provided reward too high");
    lastUpdatedTime = startTime >= block.timestamp ? startTime : uint128
        ↪ (block.timestamp);
    periodFinish = startTime + rewardsDuration;
    emit RewardAdded(reward);
}
```

Recommendation:

It is recommended to use a multiplier to increase the precision of the rewardRate calculation and to avoid rounding errors.

Updates

The Crowdswap team resolved the issue by using a multiplier to reduce the risk of rounding errors.

SHB.4 The Fees Can Be Bypassed

- Severity : HIGH
- Status : Acknowledged
- Likelihood : 2
- Impact : 3

Description:

The opportunity contracts implement two types of fees: liquidity providing fees (addLiquidityFee, removeLiquidityFee), staking fees (stakingFee, unstakingFee). These fees are calculated using the _deductFee function which calls the _calculateFee function, due to a rounding error that can occur in the _calculateFee function, the fees can be bypassed for any _amount that makes _percentage * _amount lower than 10^{20}.

Files Affected:

SHB.4.1: Opportunity.sol

```
function _deductFee(
    uint256[] memory _fees,
    IERC20Upgradeable _token,
    uint256 _amount
) private returns (uint256 _totalFee) {
    for(uint256 i = 0; i < _fees.length; i++) {
        _totalFee += _calculateFee(_amount, _fees[i]);
    }
    _token.uniTransfer(feeTo, _totalFee);
    emit FeeDeducted(msg.sender, address(_token), _amount, _totalFee);
}

function _calculateFee(
    uint256 _amount,
    uint256 _percentage
) private pure returns (uint256) {
    return _percentage * _amount / 1 ether / 100;
}
```

Recommendation:

Consider using a multiplier to increase the precision of the _totalFee calculation and to avoid rounding errors.

Updates

The Crowdswap team acknowledged the issue, stating that the protocol accepts not receiving fees for the small amounts.

SHB.5 Possible Desynchronization Between tokenA, tokenB and pair

- Severity : MEDIUM
- Status : Fixed
- Likelihood : 2
- Impact : 2

Description:

The setTokenA, setTokenB and setPair functions are used to modify the addresses of tokenA, tokenB and pair. These addresses are interrelated, so any change in any of these addresses requires a change in one of the other ones, the current implementation can cause a desynchronization between these addresses.

Files Affected:

SHB.5.1: Opportunity.sol

```
function setTokenA(address _tokenA) external onlyOwner {
    require(_tokenA != address(0), "oe12");
    tokenA = IERC20Upgradeable(_tokenA);
}

function setTokenB(address _tokenB) external onlyOwner {
    require(_tokenB != address(0), "oe12");
    tokenB = IERC20Upgradeable(_tokenB);
}

function setPair(address _pair) external onlyOwner {
    require(_pair != address(0), "oe12");
    pair = IERC20Upgradeable(_pair);
}
```

SHB.5.2: Opportunity.sol

```
function _initializeContracts(
    address _tokenA,
    address _tokenB,
    address _pair
) internal onlyInitializing {
    OwnableUpgradeable.initialize();
    PausableUpgradeable.__Pausable_init();
    tokenA = IERC20Upgradeable(_tokenA);
    tokenB = IERC20Upgradeable(_tokenB);
    pair = IERC20Upgradeable(_pair);
}
```

Recommendation:

Consider using one setter that changes the tokenA and tokenB, and change the pair value based on the new tokens by calling the getPair function of the UniswapV2Factory.

Updates

The Crowdswap team resolved the issue by using setTokenAandTokenB as a setter and changing the pair accordingly.

SHB.5.3: Opportunity.sol

```
function setTokenAandTokenB(
    address _tokenA,
    address _tokenB
) public onlyOwner {
    require(_tokenA != address(0), "oe12");
    require(_tokenB != address(0), "oe12");
    _tokenA = IERC20Upgradeable(_tokenA).isETH()
        ? address(coinWrapper)
        : _tokenA;
    _tokenB = IERC20Upgradeable(_tokenB).isETH()
        ? address(coinWrapper)
        : _tokenB;

    address _pair = IUniswapV2Factory(pairFactoryContract).getPair(
        _tokenA,
        _tokenB
    );
    require(_pair != address(0), "pair is not valid");
    pair = IERC20Upgradeable(_pair);
    tokenA = IERC20Upgradeable(_tokenA);
    tokenB = IERC20Upgradeable(_tokenB);
    emit SetTokens(msg.sender, _tokenA, _tokenB);
}
```

SHB.6 Front-run In The Contract's Initialization

- Severity : MEDIUM
- Status : Fixed

- Likelihood : 1
- Impact : 3

Description:

The opportunity contracts and staking contract initialize their state with an initialize function instead of a constructor to implement upgradability, leaving the initialization vulnerable to being front-run by an attacker.

Exploit Scenario:

The owner deploys the contract and performs the initialize function, then the attacker front-runs the transaction by paying a higher gas price and inputting malicious values into the contract.

Files Affected:

SHB.6.1: BeefyMimaticUsdcOpportunity.sol

```
function initialize(
    address _tokenMimatic,
    address _tokenUsdc,
    address _pairMimaticUsdc,
    address payable _feeTo,
    uint256 _addLiquidityFee,
    uint256 _removeLiquidityFee,
    uint256 _stakeFee,
    uint256 _unstakeFee,
    address _swapContract,
    address _router,
    address _vault
) public initializer {
```

SHB.6.2: CrowdUsdtLpStakeOpportunity.sol

```
function initialize(
    address _tokenCrowd,
    address _tokenUsdt,
    address _pairCrowdUsdt,
    address payable _feeTo,
    uint256 _addLiquidityFee,
    uint256 _removeLiquidityFee,
    uint256 _stakeFee,
    uint256 _unstakeFee,
    address _swapContract,
    address _router,
    address _stakingLP
) public initializer {
```

SHB.6.3: PancakeOpportunity.sol

```
function initialize(
    address _tokenA,
    address _tokenB,
    address _rewardToken,
    address _pair,
    FeeStruct memory feeStruct,
    address _swapContract,
    address _router,
    address _pancakeMasterChefV2,
    uint256 _pId
) public initializer {
```

SHB.6.4: StakingLP.sol

```
function initialize(
    address _lpStakingToken,
    address _rewardToken,
    uint128 _rewardsDuration,
    uint128 _startTime
) public initializer {
```

Recommendation:

Consider deploying the contract and initializing it in the same transaction or adding access control to the initialize function.

Updates

The Crowdswap team resolved the issue by making use of the hardhat upgrades library which uses upgradeToAndCall to initialize the implementation in the same transaction.

SHB.7 The Investors' Native Tokens Can Get Locked

- Severity : LOW
- Status : Fixed
- Likelihood : 1
- Impact : 2

Description:

The investByToken is payable and accepts the native token as an input token in addition to ERC20 tokens, in the case where the _token argument is not the native token, and the user sends native tokens to the contracts, these funds will get locked in the contract.

Files Affected:

SHB.7.1: Opportunity.sol

```
function investByToken(
    address _userAddress,
    IERC20Upgradeable _token,
    uint256 _amount,
    uint256 _secondAmount,
    AddLiqDescriptor memory _addLiqDescriptor,
    bytes calldata _swapDataToB,
    bytes calldata _swapDataToA
) external payable whenNotPaused {
    if (_token.isETH()) {
        require(msg.value >= _amount, "oe03");
    } else {
        _transferFrom(_token, _amount);
    }
```

Recommendation:

Consider verifying the msg.value to be equal to zero if the _token argument is different than the native token.

Updates

The Crowdswap team resolved the issue by adding a require statement to make sure the msg.value is equal to zero when the invested token is an ERC20.

SHB.8 Missing Value Verification

- Severity : LOW
- Status : Fixed
- Likelihood : 1
- Impact : 2

Description:

Certain functions lack a value safety check, the values of the arguments should be verified to allow only the ones that comply with the contract's logic. In the Opportunity contract, the investByTokenAOrTokenB and investByToken functions should verify the _secondAmount to be lower than _amount. Also all the fees initializers and setters should implement a limitation that verifies that the fees cannot surpass a reasonable value. Finally, the initialize function of the StakingLP contract should verify the _rewardsDuration to be different from zero, and the _startTime to be greater than block.timestamp.

Files Affected:

SHB.8.1: Opportunity.sol

```
function investByTokenAOrTokenB(
    address _userAddress,
    IERC20Upgradeable _token,
    uint256 _amount,
    uint256 _secondAmount,
    AddLiqDescriptor memory _addLiqDescriptor,
    bytes calldata _swapData
) external whenNotPaused {
```

SHB.8.2: Opportunity.sol

```
function investByToken(
    address _userAddress,
    IERC20Upgradeable _token,
    uint256 _amount,
    uint256 _secondAmount,
    AddLiqDescriptor memory _addLiqDescriptor,
    bytes calldata _swapDataToB,
    bytes calldata _swapDataToA
) external payable whenNotPaused {
```

SHB.8.3: Opportunity.sol

```
function setAddLiquidityFee(uint256 _feePercentage) external onlyOwner {
    require(_feePercentage >= 0, "oe11");
    addLiquidityFee = _feePercentage;
}

function setRemoveLiquidityFee(uint256 _feePercentage) external onlyOwner {
    require(_feePercentage >= 0, "oe11");
    removeLiquidityFee = _feePercentage;
}

function setStakeFee(uint256 _feePercentage) external onlyOwner {
    require(_feePercentage >= 0, "oe11");
    stakeFee = _feePercentage;
}

function setUnstakeFee(uint256 _feePercentage) external onlyOwner {
    require(_feePercentage >= 0, "oe11");
    unstakeFee = _feePercentage;
}
```

SHB.8.4: Opportunity.sol

```
function _initializeFees(
    address payable _feeTo,
    uint256 _addLiquidityFee,
    uint256 _removeLiquidityFee,
    uint256 _stakeFee,
    uint256 _unstakeFee
) internal onlyInitializing {
    feeTo = _feeTo;
    addLiquidityFee = _addLiquidityFee;
    removeLiquidityFee = _removeLiquidityFee;
    stakeFee = _stakeFee;
    unstakeFee = _unstakeFee;
}
```

SHB.8.5: StakingLP.sol

```
function initialize(
    address _lpStakingToken,
    address _rewardToken,
    uint128 _rewardsDuration,
    uint128 _startTime
) public initializer {
    OwnableUpgradeable.initialize();
    PausableUpgradeable.__Pausable_init();
    lpStakingToken = IERC20Upgradeable(_lpStakingToken);
    rewardToken = IERC20Upgradeable(_rewardToken);
    periodFinish = 0;
    rewardsDuration = _rewardsDuration;
    startTime = _startTime;
}
```

Recommendation:

We recommend that you verify the values provided in the arguments. The issue can be addressed by utilizing a require statement.

Updates

The Crowdswap team resolved the issue by verifying the values provided in the arguments to restrict invalid values.

SHB.9 Missing Address Verification

- Severity : LOW
- Status : Fixed
- Likelihood : 1
- Impact : 2

Description:

Certain functions lack a safety check in the address, the address-type arguments should include a zero-address test, otherwise, the contract's functionality may become inaccessible.

Files Affected:

SHB.9.1: Opportunity.sol

```
function _initializeContracts(
    address _tokenA,
    address _tokenB,
    address _pair
) internal onlyInitializing {
    OwnableUpgradeable.initialize();
    PausableUpgradeable.__Pausable_init();
    tokenA = IERC20Upgradeable(_tokenA);
    tokenB = IERC20Upgradeable(_tokenB);
    pair = IERC20Upgradeable(_pair);
}
```

SHB.9.2: StakingLP.sol

```
function setOpportunityContract(address _opportunityContract) external onlyOwner {
    opportunityContract = _opportunityContract;
    emit callerSet(_opportunityContract);
}
```

SHB.9.3: StakingLP.sol

```
function setResonateAdapter(address _resonateAdapter) external onlyOwner {
    resonateAdapter = _resonateAdapter;
    emit callerSet(_resonateAdapter);
}
```

Recommendation:

Consider verifying the following arguments to be different from the address(0): _tokenA, _tokenB, _pair, _opportunityContract and _resonateAdapter.

Updates

The Crowdswap team resolved the issue by verifying the address arguments to be different from the address(0).

SHB.10 Potential Denial of Service (DoS) and Compatibility Issues Due to Use of transfer For Sending Ether

- Severity : LOW
- Status : Fixed

- Likelihood : 1
- Impact : 2

Description:

The smart contract uses the transfer function to send Ether. While transfer is a commonly used function, it has a gas limit of 2300, which can potentially lead to a Denial of Service (DoS) attack if the opcode costs change such that 2300 gas is insufficient. This could cause the function to fail, preventing the contract from sending Ether. Additionally, the transfer function is not supported by some Layer 2 blockchains, which could limit the contract's compatibility and usability across different blockchain networks.

Files Affected:

SHB.10.1: UniERC20.sol

```
function uniTransfer(IERC20 token, address payable to, uint256 amount)
    ↪ internal {
    if (amount > 0) {
        if (isETH(token)) {
            to.transfer(amount);
        } else {
            token.safeTransfer(to, amount);
        }
    }
}
```

SHB.10.2: UniERC20Upgradeable.sol

```
function uniTransfer(IERC20Upgradeable token, address payable to,
    ↪ uint256 amount) internal {
    if (amount > 0) {
        if (isETH(token)) {
            to.transfer(amount);
        } else {
            token.safeTransfer(to, amount);
        }
    }
}
```

SHB.10.3: OpportunityV2.sol

```solidity
function _returnRemainedTokens(
    IERC20Upgradeable _token,
    uint256 _amount,
    address _userAddress
) private {
    if (_amount <= 0) return;
    if (address(_token) == address(coinWrapper)) {
        coinWrapper.withdraw(_amount);
        payable(_userAddress).transfer(_amount);
    } else {
        _token.uniTransfer(payable(_userAddress), _amount);
    }
    emit Refund(_userAddress, address(_token), _amount);
}
```

Recommendation:

Consider replacing the transfer function with a safer alternative, such as the callvalue:...("") function. This function does not have a gas limit, reducing the risk of a DoS attack due to insufficient gas. It is also more widely supported across different blockchain networks, improving the contract's compatibility. Note that you need to make sure to eliminate the attack vectors related to reentrancy when using callvalue:...("") by making use of the Check Effect Interactions pattern or using the nonReenrant modifier.

Updates

The Crowdswap team resolved the issue by using callvalue:...("") to transfer ether. Furthermore, they implemented the use of the nonReenrant modifier to prevent reentrancy attacks.

SHB.11 Missing User Address Validation in Investment Functions

- Severity : LOW
- Status : Fixed
- Likelihood : 1
- Impact : 2

Description:

The investByTokenATokenB, investByTokenA, investByTokenB, investByToken, and investByLP functions in the smart contract do not validate that the _userAddress parameter is not the zero address (0x0). This omission can lead to a loss of funds if a user mistakenly enters the zero address when calling these functions, as any tokens sent to the zero address are irretrievable.

Files Affected:

SHB.11.1: OpportunityV2.sol

```
function investByTokenATokenB(
    address _userAddress,
    uint256 _amountA,
    uint256 _amountB,
    uint256 _addLiquidityDeadline
) external payable whenNotPaused refund(_userAddress) {
```

SHB.11.2: OpportunityV2.sol

```
function investByTokenB(
    address _userAddress,
    uint256 _amount,
    DexDescriptor memory _dexDescriptor,
    uint256 _addLiquidityDeadline
) external payable whenNotPaused refund(_userAddress) {
```

SHB.11.3: OpportunityV2.sol

```
function investByToken(
    address _userAddress,
    IERC20Upgradeable _token,
    uint256 _amount,
    DexDescriptor memory _dexDescriptorB,
    DexDescriptor memory _dexDescriptorA,
    uint256 _deadline
) external payable whenNotPaused refund(_userAddress) {
```

SHB.11.4: OpportunityV2.sol

```
function investByLP(
    address _userAddress,
    uint256 _amountLP
) external whenNotPaused {
```

Recommendation:

Implement checks in the investByTokenATokenB, investByTokenA, investByTokenB, investByToken, and investByLP functions to ensure that the _userAddress parameter is not the zero address. This will prevent funds from being accidentally sent to the zero address. It is also recommended to conduct further testing and auditing to ensure that the updated functions behave as expected.

Updates

The Crowdswap team resolved the issue by verifying the _userAddress argument to be different from the address(0).

4 Best Practices

BP.1 Utilize more expressive error messages

Description:

All the require statements utilize a format of errors that is not understandable, it is recommended to write more expressive error messages to make debugging easier and to improve the quality of the code.

Status - Acknowledged

BP.2 Remove unnecessary initializations

Description:

The variable periodFinish is initialized to zero in the initialize function. However, this initialization is unnecessary, as the value of periodFinish is set by default to zero. Therefore, it is recommended to remove the unnecessary initialization of periodFinish to zero. This will help simplify the code and reduce the gas cost of executing the function.

Files Affected:

BP.2.1: StakingLP.sol

```
function initialize(
    address _lpStakingToken,
    address _rewardToken,
    uint128 _rewardsDuration,
    uint128 _startTime
) public initializer {
    OwnableUpgradeable.initialize();
    PausableUpgradeable.__Pausable_init();
    lpStakingToken = IERC20Upgradeable(_lpStakingToken);
    rewardToken = IERC20Upgradeable(_rewardToken);
    periodFinish = 0;
    rewardsDuration = _rewardsDuration;
    startTime = _startTime;
}
```

Status - Fixed

BP.3 Usage of pre-increment

Description:

i++ is generally more expensive because it must increment a value and "return" the old value, so it may require holding two numbers in memory. ++i only ever uses one number in memory therfore, ++i consumes less Gas than i++.

Status - Acknowledged

BP.4 Public Function Can Be Called External

Description:

The functions with a public scope that are not called inside the contract should be declared external to optimize the gas cost.

Files Affected:

BP.4.1: BeefyMimaticUsdcOpportunity.sol

```
function initialize(
    address _tokenMimatic,
    address _tokenUsdc,
    address _pairMimaticUsdc,
    address payable _feeTo,
    uint256 _addLiquidityFee,
    uint256 _removeLiquidityFee,
    uint256 _stakeFee,
    uint256 _unstakeFee,
    address _swapContract,
    address _router,
    address _vault
) public initializer {
```

BP.4.2: CrowdUsdtLpStakeOpportunity.sol

```
function initialize(
    address _tokenCrowd,
    address _tokenUsdt,
    address _pairCrowdUsdt,
    address payable _feeTo,
    uint256 _addLiquidityFee,
    uint256 _removeLiquidityFee,
    uint256 _stakeFee,
    uint256 _unstakeFee,
    address _swapContract,
    address _router,
    address _stakingLP
) public initializer {
```

BP.4.3: PancakeOpportunity.sol

```
function initialize(
    address _tokenA,
    address _tokenB,
    address _rewardToken,
    address _pair,
    FeeStruct memory feeStruct,
    address _swapContract,
    address _router,
    address _pancakeMasterChefV2,
    uint256 _pId
) public initializer {
```

BP.4.4: StakingLP.sol

```
function initialize(
    address _lpStakingToken,
    address _rewardToken,
    uint128 _rewardsDuration,
    uint128 _startTime
) public initializer {
```

Status - Acknowledged

BP.5 Remove tautology

Description:

The fee setters contain a tautology, the functions verify the argument to be greater or equal to zero, which is always true since this check passes for any uint256, it is recommended to remove this verification.

Files Affected:

BP.5.1: Opportunity.sol

```
function setAddLiquidityFee(uint256 _feePercentage) external onlyOwner {
    require(_feePercentage >= 0, "oe11");
    addLiquidityFee = _feePercentage;
}

function setRemoveLiquidityFee(uint256 _feePercentage) external
    ↪ onlyOwner {
    require(_feePercentage >= 0, "oe11");
    removeLiquidityFee = _feePercentage;
}

function setStakeFee(uint256 _feePercentage) external onlyOwner {
    require(_feePercentage >= 0, "oe11");
    stakeFee = _feePercentage;
}

function setUnstakeFee(uint256 _feePercentage) external onlyOwner {
    require(_feePercentage >= 0, "oe11");
    unstakeFee = _feePercentage;
}
```

Status - Fixed

5 Tests

The tests included in this section were based on the commit hash : 393906dbc080e835204ce1fee56e95b5bc4717f2

→ BeefyMimaticUsdcOpportunity

→ invest

✓ User should be able to invest sending USDC and MIMATIC

✓ User should be able to invest sending MIMATIC and USDC

✓ User should be able to invest sending DAI

✓ User should be able to invest sending MATIC

✓ User should be able to invest sending MIMATIC

✓ User should be able to invest sending USDC

✓ User should be able to invest sending LP

✓ Should fail when the amountOut of the first swap is not equal or greater than the expected amountOut, sending MIMATIC

✓ Should fail when the amountOut of the first swap is not equal or greater than the expected amountOut, sending USDC

✓ Should fail when the amountOut of the first swap is not equal or greater than the expected amountOut, sending DAI

✓ Should fail when the amountOut of the first swap is not equal or greater than the expected amountOut, sending MATIC

✓ Should fail when the amountOut of the second swap is not equal or greater than the expected amountOut, sending DAI

✓ Should fail when the amountOut of the second swap is not equal or greater than the expected amountOut, sending MATIC

✓ Should fail when the msg.value is lower than the input amount

✓ Should fail when unknown token is sent to investByTokenATokenB function

✓ Should fail when unknown token is sent to investByTokenAOrTokenB function

✓ Should fail when wrong swap data is sent to investByToken function

→ leave

✓ User should be able to leave, unstaking all LP

✓ User should be able to leave, unstaking some LP

→ admin operations

✓ should change the fee recipient

✓ should change the add liquidity fee

✓ should change the remove liquidity fee

✓ should change the stake fee

✓ should change the unstake fee

✓ should change the tokenA

✓ should change the tokenB

✓ should change the pair contract

✓ should change the swap contract

✓ should change the router contract

✓ should change the stakingLP contract

✓ should fail using none owner address

✓ should fail to set addresses to zero

→ Pausable

✓ should pause the contract

✓ should fail to invest while the contract is paused

✓ should fail to leave while the contract is paused

✓ should unpause the contract

✓ should fail using none owner address

→ CrowdUsdtLpStakeOpportunity

→ invest

✓ User should be able to invest sending USDT and CROWD

✓ User should be able to invest sending CROWD and USDT

✓ User should be able to invest sending DAI

✓ User should be able to invest sending MATIC

✓ User should be able to invest sending CROWD

✓ User should be able to invest sending USDT

✓ User should be able to invest sending LP

✓ Should fail when the amountOut of the first swap is not equal or greater than the expected amountOut, sending CROWD

✓ Should fail when the amountOut of the first swap is not equal or greater than the expected amountOut, sending USDT

✓ Should fail when the amountOut of the first swap is not equal or greater than the expected amountOut, sending DAI

✓ Should fail when the amountOut of the first swap is not equal or greater than the expected amountOut, sending MATIC

✓ Should fail when the amountOut of the second swap is not equal or greater than the expected amountOut, sending DAI

✓ Should fail when the amountOut of the second swap is not equal or greater than the expected amountOut, sending MATIC

✓ Should fail when the msg.value is lower than the input amount

✓ Should fail when unknown token is sent to investByTokenATokenB function

✓ Should fail when unknown token is sent to investByTokenAOrTokenB function

✓ Should fail when wrong swap data is sent to investByToken function

→ leave

✓ User should be able to leave, unstaking all LP

✓ User should be able to leave, unstaking some LP

→ admin operations

✓ should change the fee recipient

✓ should change the add liquidity fee

✓ should change the remove liquidity fee

✓ should change the stake fee

✓ should change the unstake fee

✓ should change the tokenA

✓ should change the tokenB

✓ should change the pair contract

✓ should change the swap contract

```
      ✓ should change the router contract
      ✓ should change the stakingLP contract
      ✓ should fail using none owner address
      ✓ should fail to set addresses to zero

    → Pausable
      ✓ should pause the contract
      ✓ should fail to invest while the contract is paused
      ✓ should fail to leave while the contract is paused
      ✓ should unpause the contract
      ✓ should fail using none owner address

  → PancakeCakeBnbOpportunity

    → invest
      ✓ User should be able to invest sending CAKE and WBNB
      ✓ User should be able to invest sending WBNB and CAKE
      ✓ User should be able to invest sending DAI
      ✓ User should be able to invest sending BNB
      ✓ User should be able to invest sending CAKE
      ✓ User should be able to invest sending WBNB
      ✓ User should be able to invest sending LP
      ✓ Should fail when the amountOut of the first swap is not equal or
        greater than the expected amountOut, sending CAKE
      ✓ Should fail when the amountOut of the first swap is not equal or
        greater than the expected amountOut, sending WBNB
```

✓ Should fail when the amountOut of the first swap is not equal or greater than the expected amountOut, sending DAI

✓ Should fail when the amountOut of the first swap is not equal or greater than the expected amountOut, sending BNB

✓ Should fail when the amountOut of the second swap is not equal or greater than the expected amountOut, sending DAI

✓ Should fail when the amountOut of the second swap is not equal or greater than the expected amountOut, sending BNB

✓ Should fail when the msg.value is lower than the input amount

✓ Should fail when unknown token is sent to investByTokenATokenB function

✓ Should fail when unknown token is sent to investByTokenAOrTokenB function

✓ Should fail when wrong swap data is sent to investByToken function

→ leave

✓ User should be able to leave, unstaking all LP

✓ User should be able to leave, unstaking some LP

→ withdrawRewards

✓ User should be able to withdraw his/her rewards

✓ Users should be able to withdraw their rewards

✓ cannot withdraw 0

✓ should fail when the user does not exist

✓ should fail when trying to withdraw more rewards

→ splitting the rewards

✓ the sum of users' balance in the opportunity contract must be equal to the amount of the opportunity contract in the masterChef

✓ should correctly split rewards according to users' shares

→ admin operations

✓ should change the fee recipient

✓ should change the add liquidity fee

✓ should change the remove liquidity fee

✓ should change the stake fee

✓ should change the unstake fee

✓ should change the tokenA

✓ should change the tokenB

✓ should change the pair contract

✓ should change the swap contract

✓ should change the router contract

✓ should change the pancakeMasterChefV2 contract

✓ should fail using none owner address

✓ should fail to set addresses to zero

→ Pausable

✓ should pause the contract

✓ should fail to invest while the contract is paused

✓ should fail to leave while the contract is paused

✓ should unpause the contract

```
        ✓ should fail using none owner address

    → StakingLP

        → stakeLP
        ✓ should fail before setting the OpportunityContract
        ✓ should fail before setting the ResonateAdapter
        ✓ should fail before setting both contracts
        ✓ should fail if the caller is not the OpportunityContract
        ✓ should fail before the start of the opportunity
        ✓ should fail sending zero
        ✓ User should be able to stake if they are eligible
        ✓ User should be able to stake
        ✓ ResonateAdapter should be able to stake

        → withdrawRewards
        ✓ should fail sending zero
        ✓ should fail when stakeholder does not exist
        ✓ should fail when trying to withdraw more rewards
        ✓ User should be able to withdraw rewards

        → withdraw
        ✓ should fail before setting the OpportunityContract
        ✓ should fail before setting the ResonateAdapter
        ✓ should fail before setting both contracts
        ✓ should fail if the caller is not the OpportunityContract
```

✓ should fail sending zero

✓ should fail when stakeholder does not exist

✓ should fail when trying to withdraw more LP tokens

✓ should fail when trying to withdraw for an account other than resonateAdapter

✓ User should be able to withdraw some LP tokens

✓ User should be able to withdraw all LP tokens and receive all rewards

→ withdrawByOwner

✓ should fail using none owner address

✓ Owner should be able to withdraw some LP tokens

✓ Owner should be able to withdraw all LP tokens and receive all rewards

→ notifyRewardAmount

✓ rewards are changed during the opportunity

✓ rewards are changed before the start of the opportunity

→ setRewardsDuration

✓ should fail passing invalid duration

✓ rewards are changed during the opportunity

✓ rewards are changed before the start of the opportunity

✓ duration is changed during the opportunity

✓ duration is changed before the start of the opportunity

✓ opportunity has started with couple of users

→ earned

✓ The eligible user stakes and their rewards should be calculated after startTime

✓ The rewards calculation should stop when the opportunity ends

✓ Multiple users

→ startTime

✓ should fail sending passed timestamp

✓ should fail when the startTime has passed

✓ combination of change duration and start time

→ Pausable

✓ should pause the contract

✓ should fail to withdrawRewards while the contract is paused

✓ should unpause the contract

✓ should fail using none owner address

162 passing (7 min)

Coverage:

The code coverage results were obtained by running npx hardhat coverage in the Opportunities project. We found the following results :

- Statements Coverage : 94.63%
- Branches Coverage : 74.19%
- Functions Coverage : 88.16%
- Lines Coverage : 94.5%

6 Conclusion

We examined the design and implementation of Crowdswap in this audit and found several issues of various severities. We advise Crowdswap team to implement the recommendations contained in all 11 of our findings to further enhance the code's security. It is of utmost priority to start by addressing the most severe exploit discovered by the auditors then followed by the remaining exploits, and finally we will be conducting a re-audit following the implementation of the remediation plan contained in this report.

We would much appreciate any constructive feedback or suggestions regarding our methodology, audit findings, or potential scope gaps in this report.

7 Scope Files

7.1 Audit

Files	MD5 Hash
contracts/opportunity/BeefyMimaticUsdcOpportunity.sol	1f5682135b278995ac491985a7068cd8
contracts/opportunity/CrowdUsdtLpStakeOpportunity.sol	714eafe500552caddc67ff67a9b16115
contracts/opportunity/Opportunity.sol	70575785ebb3858c3b93f3e4f8f9e96f
contracts/opportunity/PancakeOpportunity.sol	2911191457fd9668bedb8592b355a4ae
contracts/opportunity/StakingLP.sol	f8802d2a4f5e4925197c6db541d38377
contracts/libraries/UniERC20Upgradeable.sol	83849bfaf3313ee4576b655d8b919849
contracts/interfaces/IBeefyVault.sol	db0532e52f5c8d8b193b1ffd30a426dd
contracts/interfaces/IPancakeMasterChefV2.sol	0bb246731b2b9040f34338b89f7609fa
contracts/interfaces/IStakingLP.sol	4e3547ed3e978558eed7db7fe234ad6e
contracts/interfaces/IUniswapV2Router02.sol	c15829eed107c99f43227936f222f4dc
contracts/helpers/OwnableUpgradeable.sol	d6f1fd5b81b5f8b826f6c04ee2dfdb00

Files	MD5 Hash
contracts/helpers/OwnableUpgradeable.sol	d6f1fd5b81b5f8b826f6c04ee2dfdb00
contracts/opportunity/StakingLP.sol	c00fd0f9dc449abf960b548a227c89a3

contracts/opportunity/v2/PancakeOpportunityV2.sol	3b94cbef4104bc950ebda1d0d3b7ef1e
contracts/opportunity/v2/OpportunityV2.sol	ac0ee6f6f2f349a66ee5724cecb30047
contracts/opportunity/v2/CrowdUsdtLpStakeOpportunityV2.sol	7f7b945add155dc3c918c44bdc14476e
contracts/opportunity/v2/BeefyMimaticUsdcOpportunityV2.sol	1e7623f5e93419a6ddfd85e6d7165011
contracts/interfaces/IUniswapV2Factory.sol	70691070ed218dd4f10a94020e7e9537
contracts/interfaces/IPancakeMasterChefV2.sol	0bb246731b2b9040f34338b89f7609fa
contracts/interfaces/IUniswapV2Pair.sol	53e81d91648652da9554789a410fc83e
contracts/interfaces/IBeefyVault.sol	db0532e52f5c8d8b193b1ffd30a426dd
contracts/interfaces/IStakingLP.sol	4e3547ed3e978558eed7db7fe234ad6e
contracts/interfaces/ICrowdswapAggregator.sol	1a4e4d3162d8489b8a206fec075b2218
contracts/interfaces/IWETH.sol	b4e4c3a8d6620db11f19952bdbdb44fd
contracts/interfaces/IUniswapV2Router02.sol	9451283ffa968e2342392ddb98dd530e
contracts/libraries/UniERC20.sol	a405c48c4ec526bc8f8fb1fbb5db23cb
contracts/libraries/Math.sol	b6387945b4dff0e94e518ebbc3e2409c
contracts/libraries/UniERC20Upgradeable.sol	83849bfaf3313ee4576b655d8b919849

7.2 Re-Audit

Files	MD5 Hash

contracts/opportunity/StakingLP.sol	c00fd0f9dc449abf960b548a227c89a3
contracts/opportunity/v2/BeefyMimaticUsdcOpportunityV2.sol	1e7623f5e93419a6ddfd85e6d7165011
contracts/opportunity/v2/CrowdUsdtLpStakeOpportunityV2.sol	7f7b945add155dc3c918c44bdc14476e
contracts/opportunity/v2/OpportunityV2.sol	b2c1d5e7515af1424977111dea91ab1f
contracts/opportunity/v2/PancakeOpportunityV2.sol	416c16cdff17e46106780e318bd8e144
contracts/libraries/Math.sol	b6387945b4dff0e94e518ebbc3e2409c
contracts/libraries/UniERC20.sol	e143e2732dfb6016a60c7a9d9705f9e8
contracts/libraries/UniERC20Upgradeable.sol	bff9952303518ac64a8ab09f431b2532
contracts/interfaces/IBeefyVault.sol	db0532e52f5c8d8b193b1ffd30a426dd
contracts/interfaces/ICrowdswapAggregator.sol	1a4e4d3162d8489b8a206fec075b2218
contracts/interfaces/IPancakeMasterChefV2.sol	0bb246731b2b9040f34338b89f7609fa
contracts/interfaces/IStakingLP.sol	4e3547ed3e978558eed7db7fe234ad6e
contracts/interfaces/IUniswapV2Factory.sol	70691070ed218dd4f10a94020e7e9537
contracts/interfaces/IUniswapV2Pair.sol	53e81d91648652da9554789a410fc83e
contracts/interfaces/IUniswapV2Router02.sol	9451283ffa968e2342392ddb98dd530e
contracts/interfaces/IWETH.sol	b4e4c3a8d6620db11f19952bdbdb44fd
contracts/helpers/OwnableUpgradeable.sol	d6f1fd5b81b5f8b826f6c04ee2dfdb00

8 Disclaimer

Shellboxes reports should not be construed as ”endorsements” or ”disapprovals” of particular teams or projects. These reports do not reflect the economics or value of any ”product” or ”asset” produced by any team or project that engages Shellboxes to do a security evaluation, nor should they be regarded as such. Shellboxes Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the examined technology, nor do they provide any indication of the technology’s proprietors, business model, business or legal compliance. Shellboxes Reports should not be used in any way to decide whether to invest in or take part in a certain project. These reports don’t offer any kind of investing advice and shouldn’t be used that way. Shellboxes Reports are the result of a thorough auditing process designed to assist our clients in improving the quality of their code while lowering the significant risk posed by blockchain technology. According to Shellboxes, each business and person is in charge of their own due diligence and ongoing security. Shellboxes does not guarantee the security or functionality of the technology we agree to research; instead, our purpose is to assist in limiting the attack vectors and the high degree of variation associated with using new and evolving technologies.

SHELLBOXES

For a Contract Audit, contact us at contact@shellboxes.com