SHELLBOXES

Morphswap

Smart Contract Security Audit

Prepared by ShellBoxes

Nov 22nd, 2022 - Dec 2nd, 2022

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	Morphswap
Version	1.0
Classification	Public

Scope

Repository	Commit Hash
https://github.com/morphswap/MS_Audit/tree/main/MS_Audit	2ec048e40c80f766c4870606ffa508cc19f72c88

Re-Audit

Repository	Commit Hash
https://github.com/morphswap/MS_Audit/tree/main/MS_Audit	f64bcdde99e56256f81535c84d0e867e78cd977a

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction

Morphswap engaged ShellBoxes to conduct a security assessment on the Morphswap beginning on Nov 22nd, 2022 and ending Dec 2nd, 2022. 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 Morphswap

Morphswap is a fully decentralized, cross-chain automated market maker.

Issuer	Morphswap
Website	`https://morphswap.io`
Type	Solidity Smart Contract
Whitepaper	`https://morphswap.io/whitepaper`
Documentation	`https://docs.morphswap.io`
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	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

Likelihood

2 Findings Overview

2.1 Disclaimer

Aside from the issues listed in the findings section, the audit team has encountered Stack too deep compilation errors in the contracts during the audit. Furthermore, the project lacks any unit, integration, or end-to-end testing methodologies that ensure the correctness of the contracts' functionalities, these tests are extremely critical and can help discover multiple bugs before deployment which can save potentially lost funds.

Additionally, the majority of smart contracts contain commented code, and the names of the variables and functions are not always obvious or well-documented, which could have helped in the discovery of further concerns.

Many functions from the OverallContract and PingContract contracts delegate calls to static addresses, which are not verified contracts, our auditors assume that those contracts are the same as the contracts in the project.

The Re-Audit phase resulted in the remediation of eleven issues after the team of auditors accompanied the Morphswap team in implementing the recommendations and validating the code's correctness. However, the Stack too deep compilation error still exists, we recommend keeping this issue in mind in order to avoid any future complications.

2.2 Summary

The following is a synopsis of our conclusions from our analysis of the Morphswap 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.

2.3 Key Findings

The smart contracts' implementation might be improved by addressing the discovered flaws, which include , 5 high-severity, 7 medium-severity, 3 low-severity, 1 informational-severity vulnerabilities.

Vulnerabilities	Severity	Status
SHB.1. The ChainId Can Be Manipulated	HIGH	Fixed
SHB.2. Division Before Multiplication Can Cause Loss of Precision	HIGH	Fixed
SHB.3. All Users Can Have A Referrer	HIGH	Fixed
SHB.4. The tip multiplier verification can result in DoS	HIGH	Fixed
SHB.5. The Architecture Can Have Multiple Central Nodes	HIGH	Acknowledged
SHB.6. deployNewPoolPair Does Not Deploy New Pairs	MEDIUM	Fixed
SHB.7. The liquidity provider's funds may get locked	MEDIUM	Fixed
SHB.8. Centralization Risk	MEDIUM	Acknowledged
SHB.9. Race Condition	MEDIUM	Acknowledged
SHB.10. The _admin Address Can Be Set Wrong	MEDIUM	Acknowledged
SHB.11. The Testing Contract Address Should Be Dynamic	MEDIUM	Fixed
SHB.12. Changing The _swapminingfee Can Desynchronize The clfee	MEDIUM	Fixed
SHB.13. Approve Race Condition	LOW	Fixed
SHB.14. Missing Address Verification	LOW	Acknowledged
SHB.15. Floating Pragma	LOW	Fixed
SHB.16. Too Many Digits	INFORMATIONAL	Fixed

3 Finding Details

SHB.1 The ChainId Can Be Manipulated

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

Description:

The chainid variable is initialized in the OverallContract's constructor based on the value of the chain_id argument, therefore this variable can be manipulated by the owner.

Exploit Scenario:

The owner incorrectly initializes the chainid variable, causing all functionality that relies on this variable to be executed with an incorrect value.

Files Affected:

SHB.1.1: OverallContract.sol

```
constructor(uint chain_id, bool _isCentral, address mstoken, uint
    ↪ proposalLifespan, uint8 _internalchainid, address claddress,
    ↪ address cloracle, uint _clfee, address cl_to_nativecoin_address){
_admin = msg.sender;
txnum = 0;
pairTracker = 0;
 chainid = chain_id;
defaultTipMultiplier = 2;
```

Recommendation:

Consider extracting the chain ID based on the following code:

SHB.1.2: getChainID

```
function getChainID() external view returns (uint256) {
    uint256 id;
    assembly {
        id := chainid()
    }
    return id;
}
```

Updates

The Morphswap team has fixed this issue by using the chainid() opcode instruction in the inline-assembly code to initialize the chainid variable with id of the current chain. This opcode can be used to prevent replay attacks between different chains.

SHB.1.3: OverallContract.sol

```
uint id;
assembly {
    id := chainid()
}
chainid = id;
```

SHB.2 Division Before Multiplication Can Cause Loss of Precision

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

Description:

The referral bonus is an amount that is taken from saleamount and sent to the admin or the referral, the refbonus variable is divided by 10000 before getting multiplied by _refbonusmultiplier or _refbonusmultiplier*2. This can result in a signification loss of precision in the division operation.

The same issue exists in the BuyWithNativeCoinContract.

Exploit Scenario:

- The saleamount is lower than 10000, the refbonus value will be rounded to zero.
- In the case where saleamount = k*10000 + p where k is an integer, p is an integer and 0 < p < 10000, the result of saleamount will be inaccurate, and it will result in a significant loss of precision.

Files Affected:

SHB.2.1: BuyContract.sol

```
uint refbonus = (saleamount)/10000;
uint endsaleamount = saleamount - (refbonus * _refbonusmultiplier);
if (referred_to_referrer[msg.sender] == address(0)) {
    endsaleamount = saleamount - (refbonus * (_refbonusmultiplier*2));
    require(IERC20(cvp.thischainasset).transferFrom(msg.sender, _admin,
        ↪ refbonus * (_refbonusmultiplier*2) ), "Error transferring
        ↪ tokens; make sure contract has allowance");
} else {
    require(IERC20(cvp.thischainasset).transferFrom(msg.sender,
        ↪ referred_to_referrer[msg.sender], refbonus *
        ↪ _refbonusmultiplier ), "Error transferring tokens; make sure
        ↪ contract has allowance");
}
```

SHB.2.2: BuyWithNativeCoinContract.sol

```
uint onetenthousandth = (posttip_value)/10000;
uint endsaleamount;
if (referred_to_referrer[msg.sender] == address(0)) {
    endsaleamount = posttip_value - (onetenthousandth * (
        ↪ _refbonusmultiplier*2));
    (bool refbonusresult, ) = _admin.call{value: onetenthousandth * (
       ↪ _refbonusmultiplier*2)}("");
    require (refbonusresult);
} else {
     endsaleamount = posttip_value - (onetenthousandth *
        ↪ _refbonusmultiplier);
    (bool refbonusresult, ) = referred_to_referrer[msg.sender].call{
       ↪ value: onetenthousandth * _refbonusmultiplier}("");
    require (refbonusresult);
}
```

Recommendation:

Before performing the division operation, consider multiplying the refbonus variable by _refbonusmultiplier if the referred_to_referrer[msg.sender] equals address(0), otherwise multiply it by _refbonusmultiplier*2 if it does not.

Updates

The Morphswap team resolved this issue by performing multiplication operations before division.

SHB.2.3: BuyContract.sol

```
        uint refbonus = (saleamount); // /10000;
        uint endsaleamount = saleamount - ((refbonus *
           ↪ _refbonusmultiplier)/10000);
        if (referred_to_referrer[msg.sender] == address(0)) {
            endsaleamount = saleamount - ((refbonus * (
               ↪ _refbonusmultiplier*2))/10000);
            require(IERC20(cvp.thischainasset).transferFrom(msg.sender,
               ↪ _admin, ((refbonus * (_refbonusmultiplier*2))/10000) ),
               ↪  "Error transferring tokens; make sure contract has
               ↪ allowance");
        } else {
            require(IERC20(cvp.thischainasset).transferFrom(msg.sender,
               ↪ referred_to_referrer[msg.sender], ((refbonus *
               ↪ _refbonusmultiplier)/10000) ), "Error transferring
               ↪ tokens; make sure contract has allowance");
        }
```

SHB.2.4: BuyWithNativeCoinContract.sol

```
        uint onetenthousandth = (posttip_value);
        uint endsaleamount;
        if (referred_to_referrer[msg.sender] == address(0)) {
            endsaleamount = posttip_value - ((onetenthousandth * (
               ↪ _refbonusmultiplier*2))/10000);
            (bool refbonusresult, ) = _admin.call{value: (
               ↪ onetenthousandth * (_refbonusmultiplier*2)/10000)}("");
            require (refbonusresult);
        } else {
             endsaleamount = posttip_value - ((onetenthousandth *
                ↪ _refbonusmultiplier)/10000);
            (bool refbonusresult, ) = referred_to_referrer[msg.sender].
               ↪ call{value: (onetenthousandth * _refbonusmultiplier)
               ↪ /10000}("");
            require (refbonusresult);
        }
```

SHB.3 All Users Can Have A Referrer

- Severity: HIGH
- Status: Fixed

- Likelihood: 3
- Impact: 2

Description:

The setReferrer function allows a user to choose a referrer in order to get a reduction on his first transaction. However, every user will be able to get a referrer by looking for a user that has already made a transaction on the protocol from the transaction history.

Exploit Scenario:

1. The user uses the transaction history in order to extract an address of a user that has already performed a transaction in the MorphSwap contracts.
2. The user sets this address as his referrer using the setReferrer function.
3. The user gets a reduction on his first transaction in the MorphSwap protocol.

Files Affected:

SHB.3.1: OverallContract.sol

```solidity
function setReferrer(address _referrer) public returns (bool) {
    require(referred_to_referrer[msg.sender] == address(0));
    require(old_user[_referrer]);
    referred_to_referrer[msg.sender] = _referrer;
    referrer_to_referred[_referrer].push(msg.sender);
    return true;
}
```

Recommendation:

Consider documenting this behavior in the referral functionality.

Updates

The Morphswap team resolved the issue by disabling the setReferrer function, and documenting this behavior in the project's documentation: Referrals | Morphswap.

SHB.3.2: OverallContract.sol

```
function setReferrer(address _referrer) public returns (bool) {
        require(referred_to_referrer[msg.sender] == address(0));
        require(old_user[_referrer]);
        require(false);
        referred_to_referrer[msg.sender] = _referrer;
        referrer_to_referred[_referrer].push(msg.sender);
        return true;
    }
```

SHB.4 The tip multiplier verification can result in DoS

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

Description:

In the buy and buyWithNativeCoin functions, the user can manipulate the c2 argument in order to pay a lower tip amount, while buying from a different chain using the pairID argument. In the PingContract, there is a check in place to prevent this action upon receipt of the ping; however, this check can result in a Denial of Service if the tip multiplier is greater than 255 due to a rounding error that occurs when casting the tip multiplier to an uint8.

Files Affected:

SHB.4.1: BuyContract.sol

```
require(tipamarg >= (multichainhop ? (ecid_to_tipmul[c2]*defaultTip*3)/2
   ↪  : ecid_to_tipmul[idToPair[pairID].otherchain]*defaultTip)  msg.
   ↪ sender == address(this), "Declared tip amount must be greater
   ↪ than default tip");
```

SHB.4.2: BuyWithNativeCoin.sol

```
require(tipamarg >= (multichainhop ? (ecid_to_tipmul[c2]*defaultTip*3)/2
    ↪ : ecid_to_tipmul[idToPair[pairID].otherchain]*defaultTip)  msg.
    ↪ sender == address(this), "Declared tip amount must be greater
    ↪ than default tip");
```

SHB.4.3: PingContract.sol

```
} else if (comper.method_id == 10 && ecid_to_tipmul[idToPair[comper.
    ↪ secondpair_id].otherchain] == comper.internal_end_chainid) {
  uint128 defaulttipmult = uint128(ecid_to_tipmul[idToPair[comper.
      ↪ secondpair_id].otherchain]);
```

Recommendation:

Consider verifying the tip multiplier to be less than 256 in order to avoid type conversion errors.

Updates

The Morphswap team fixed the issue by verifying that the tip multiplier transmitted in the txobj is less than 256, allowing it to be stored in a uint8 without type conversion errors.

SHB.4.4: BuyContract.sol

```
if (multichainhop) {
                //Multi-chain swaps cannot start on central chain
            require(!centralContract, "Cannot do multi-chain swap with
                ↪ the central chain as starting point");
            require(ecid_to_tipmul[c2] < 256);
           gtxnumber_to_txobj[txnum - 1] = txobj(10, internalchainid,
               ↪ uint8(ecid_to_tipmul[c2]), container.pairID, container.
               ↪ c2w, secondpairID, address(0), address(0), uint64(((
               ↪ endsaleamount - ((refbonus * _fee)/10000))*
               ↪ one_quadrillion)/(container.prexferbal + (endsaleamount
               ↪ - ((refbonus * _fee)/10000)))), tipratiosend/3,
               ↪ icid_to_lastrtxnum[_icid] - 1, altfee);
```

SHB.5 The Architecture Can Have Multiple Central Nodes

- Severity : HIGH
- Status : Acknowledged

- Likelihood : 2
- Impact : 3

Description:

As mentioned in the documentation, the protocol's architecture has a single central node and many peripheral nodes. The centralContract is a boolean variable that tells if a chain is a central chain or a peripheral chain, this variable can be manipulated by the admin, this can result in having multiple central chains which can introduce unexpected behaviors.

Files Affected:

SHB.5.1: OverallContract.sol

```
constructor(uint chain_id, bool _isCentral, address mstoken, uint
    ↪ proposalLifespan, uint8 _internalchainid, address claddress,
    ↪ address cloracle, uint _clfee, address cl_to_nativecoin_address){
_admin = msg.sender;
txnum = 0;
pairTracker = 0;
chainid = chain_id;
defaultTipMultiplier = 2;
//defaultTipAlternate should be set with the (updateAlternateTipDefault
    ↪ -> fulfillAltPrice) function sequence before using/activating
    ↪ alternate tip payment
defaultTipAlternate = 100000 ether;
//atlernatetip is divided by 2, so a value of 3 is effectively 150%
alternateTipMult = 3;
 centralContract = _isCentral;
```

Recommendation:

Consider setting the centralContract variable to true, only if the chainID is equal to the polygon's chain ID 137.

Updates

The Morphswap team acknowledged the issue, stating that it is intended as there will be many instances in the future where they may want to have multiple deployments on the same chain.

SHB.6 deployNewPoolPair Does Not Deploy New Pairs

- Severity : MEDIUM
- Status : Fixed
- Likelihood : 1
- Impact : 3

Description:

The deployNewPoolPair function is supposed to create new pool pairs. However, this function does not perform any pair creations, rendering the functionality unusable.

Files Affected:

SHB.6.1: DeployNewPoolPairContract.sol

```
function deployNewPoolPair(uint c1a_amount, address c1a, uint c2,
    ↪ address c2a, address c2w, uint128 tipamarg) public payable
    ↪ returns (address, uint) {
    require (tipamarg >= defaultTip*ecid_to_tipmul[c2]);
    require(msg.value >= tipamarg);
    require (supportedChains[c2]);
    require (cid_c1a_c2a[c2][c1a][c2a].isValid != true);
    /*
    stup memory fillr;
    fillr.c1a = c1a;
    fillr.hel = c1a_amount;
    fillr.wlt = c2w;
    fillr.c2a = c2a;
    fillr.c2 = c2;*/
    if (tipamarg > 0){
        return (c2w, c1a_amount);
    }
    return (c1a, c2);
```

Recommendation:

Consider implementing the required logic of the deployNewPoolPair function and deploying a new AssetPool within it.

Updates

The Morphswap team resolved the issue by removing the return statement and requiring the node to be non-central in order to enable the deployNewPoolPair function to avoid the desynchronization issues.

SHB.6.2: DeployNewPoolPairContract.sol

```
if (tipamarg > 0){
    return (c2w, c1a_amount);
}
 //return (c1a, c2);
/**/
stackTooDeep_avoider3 memory container;
container.c2w = c2w;
container.c1a = c1a;
container.c2a = c2a;
container.c2 = c2;
container.c1a_amount = c1a_amount;
//ADDED (7/30): require(tcw_firstca_secondca_txo[c2w][c1a][c2a].alt_fee
    ↪ == false); tcw_firstca_secondca_txo[c2w][c1a][c2a].alt_fee = true
    ↪ ;
require(tcw_firstca_secondca_txo[c2w][c1a][c2a].alt_fee == false);
tcw_firstca_secondca_txo[c2w][c1a][c2a].alt_fee = true;
uint64 _icid = chainid_to_internalchainid[c2];
 require(!centralContract);
```

SHB.7 The liquidity provider's funds may get locked

- Severity: MEDIUM
- Status: Fixed
- Likelihood: 1
- Impact: 3

Description:

The singleSidedLiquidity function allows a liquidity provider to deposit an amount of native tokens or ERC20 tokens into an asset pool, there is a scenario where the user's funds can get locked in the contract without being used in any use-case.

Exploit Scenario:

The caller will send a value of the native asset and thec1a is different from the address(0), therefore, the native token funds will be lost.

Files Affected:

SHB.7.1: SingleSidedLiquidityContract.sol

```
function singleSidedLiquidity(uint64 pairID, uint c1a_amount, address
    ↪ c1a) public payable returns (bool){
 //something is wrong with liquidity providing maybe? idk
    require(idToPair[pairID].isValid);
    require(idToPair[pairID].thischainasset == c1a);
    if (c1a == address(0)) {
        c1a_amount = msg.value;
```

```
            address payable tempad = payable(idToPair[pairID].thischainpool);
            (bool sentresult, ) = tempad.call{value: msg.value}("");
            require(sentresult);
        } else {
            require(IERC20(c1a).transferFrom(msg.sender, idToPair[pairID].
               ↪ thischainpool, c1a_amount));
        }

        (bool sent, uint addedlp, uint oldlpts) = AssetPool(payable(idToPair
           ↪ [pairID].thischainpool)).addLiquidity(msg.sender, c1a_amount);
        require(sent);
        if (old_user[msg.sender] == false) {
            old_user[msg.sender] = true;
        }

        emit SingleLiq(pairID, c1a, msg.sender, c1a_amount, addedlp, oldlpts
           ↪ , block.number, 4);
        return true;
    }
```

Recommendation:

Consider verifying the msg.value to be equal to zero when the c1a is different from the address(0).

Updates

The Morphswap team resolved the issue by verifying the msg.value to be equal to zero when the c1a is different from the address(0).

SHB.7.2: SingleSidedLiquidityContract.sol

```
        if (c1a == address(0)) {
            c1a_amount = msg.value;
            address payable tempad = payable(idToPair[pairID].
               ↪ thischainpool);
```

```
          (bool sentresult, ) = tempad.call{value: msg.value}("");
          require(sentresult);
      } else {
       require(msg.value == 0);
          require(IERC20(c1a).transferFrom(msg.sender, idToPair[pairID
             ↪ ].thischainpool, c1a_amount));
      }
```

SHB.8 Centralization Risk

- Severity : MEDIUM
- Status : Acknowledged

- Likelihood : 1
- Impact : 3

Description:

The withdrawC function allows the admin to withdraw any amount of tokens or native funds from the contract, this represents a significant centralization risk where the owner has too much control over the contract's funds.

Files Affected:

SHB.8.1: OverallContract.sol

```
function withdrawC(bool opt, address erct, uint amtw) public returns (
    ↪ bool){
   require(msg.sender == _admin);

   if (opt) {
          (bool sent, ) = msg.sender.call{value: address(this).balance
             ↪ }("");
          require(sent);
      } else {
          IERC20 erctoken = IERC20(erct);
```

```
            require(erctoken.transfer(msg.sender, amtw), "Failed to send asset");
        }
    return true;
}
```

Recommendation:

Consider limiting this functionality to reduce the power of the owner, and using a multisig wallet as the owner, to include multiple parties in decision-making in the contracts.

Updates

The Morphswap team acknowledged the issue, stating that the contract will not have funds even if it contains a receive and a fallback functions.

SHB.9 Race Condition

- Severity : MEDIUM
- Status : Acknowledged
- Likelihood : 1
- Impact : 3

Description:

A race condition vulnerability occurs when the code depends on the order of the transactions submitted to it. The project contains some modifiable variables that might be impacted by the execution order of the transaction.

Exploit Scenario:

The swap miner calls the oraclePing function from the PingContract contract using a specific value of the _swapminingfee, then the admin changes the _swapminingfee. If the admin's transaction gets mined first, the swap miner's transaction will be executed using the new value of _swapminingfee generating an unexpected output.

Files Affected:

SHB.9.1: OverallContract.sol

```
function changeSMfee(uint newfee) public returns (bool) {
    require(msg.sender == _admin);
    _swapminingfee = newfee;
    return true;
}
```

Recommendation:

It is recommended to add the swap mining fee as an argument to the oraclePing function, then verify that it is the same as the one that is stored in the contract. Also, consider notifying the community with any change in the fee structure.

Updates

The Morphswap team acknowledged the risk, stating that the issue has a low probability of occurrence knowing that only the admin can modify the _swapminingfee.

SHB.10 The _admin Address Can Be Set Wrong

- Severity : MEDIUM
- Status : Acknowledged
- Likelihood : 1
- Impact : 3

Description:

The _admin address can be set to a wrong address or to the address(0) which can render all the privileged action nonfunctional.

Files Affected:

SHB.10.1: OverallContract.sol

```solidity
function setAdmin(address newadmin) public returns (bool){
    require(msg.sender == _admin);
    _admin = newadmin;
    return true;
}
```

Recommendation:

Consider changing the admin in two steps, the first step is to set an address as a requested admin, then the second step requires the temporary admin to accept the request and get promoted to an admin.

Updates

The Morphswap team acknowledged the issue, stating that the _admin address will be set to the address(0) once development is complete.

SHB.11 The Testing Contract Address Should Be Dynamic

- Severity : MEDIUM
- Status : Fixed
- Likelihood : 1
- Impact : 3

Description:

The OverallContract makes use of the TestingContract to change some of its variables, mainly the addresses of the contracts used in the context of the OverallContract. However, the address of the TestingContract is hard-coded in the OverallContract, this address should be updated depending on the chainid.

Files Affected:

SHB.11.1: OverallContract.sol

```
    //TODO remove function for launch
    function changeContractAddress(uint cn, address ca) public {
        //CHANGE TO ADDRESS OF TestingContract
        require(msg.sender == _admin);
        address(0x476718Ea98525f6EEBa3689b321E709522aE0930).delegatecall(msg.data);
    }
```

Recommendation:

Consider storing the TestingContract address in a variable and initializing it in the constructor.

Updates

The Morphswap team resolved the issue by storing the TestingContract address in the testingContract variable and initializing it in the OverallContract's constructor.

SHB.11.2: OverallContract.sol

```
testingContract = 0xe9C8faCB383a10a2F2d20EDB25Ce270F37F0352d;
```

SHB.12 Changing The _swapminingfee Can Desynchronize The clfee

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

Description:

The _swapminingfee and the clfee are interrelated, the _swapminingfee equals clfee*11/10. However, the changeSMfee function changes the _swapminingfee without any change in the clfee function, which will result in a desynchronization between the two values.

Files Affected:

SHB.12.1: OverallContract.sol

```
function changeCLfee(uint newfee) public returns (bool) {
    require(msg.sender == _admin);
    clfee = newfee;
    _swapminingfee = (newfee*11)/10;
    return true;
}
```

SHB.12.2: OverallContract.sol

```
function changeSMfee(uint newfee) public returns (bool) {
    require(msg.sender == _admin);
    _swapminingfee = newfee;
    return true;
}
```

Recommendation:

Consider updating the clfee when modifying the _swapminingfee.

Updates

The Morphswap team resolved the issue by modifying the changeSMfee function to update the clfee when modifying the _swapminingfee.

SHB.12.3: OverallContract.sol

```
function changeSMfee(uint newfee) public returns (bool) {
        require(msg.sender == _admin);
        _swapminingfee = newfee;
        clfee = (newfee*10)/11;
        return true;
    }
```

SHB.13 Approve Race Condition

- Severity : LOW
- Status : Fixed

- Likelihood : 1
- Impact : 2

Description:

The standard ERC20 implementation contains a widely known racing condition in its approve function.

Exploit Scenario:

A spender can witness the token owner broadcast a transaction altering their approval and quickly sign and broadcast a transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the spender's transaction is validated before the owner's, the spender will be able to get both approval amounts of both transactions.

Files Affected:

SHB.13.1: AssetPool.sol

```
function approve(address spender, uint256 amount) public virtual
    ↪ override returns (bool) {
    _approve(_msgSender(), spender, amount);
    return true;
}
```

Recommendation:

We recommend using increaseAllowance and decreaseAllowance functions to modify the approval amount instead of using the approve function to modify it.

Updates

The Morphswap team resolved the issue by disabling the approve function.

SHB.13.2: AssetPool.sol

```
function approve(address spender, uint256 amount) public virtual override returns (bool) {
        _approve(_msgSender(), spender, amount);
        require(false);
        return true;
    }
```

SHB.14 Missing Address Verification

- Severity : LOW
- Status : Acknowledged
- 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.

Exploit Scenario:

- The caller sets the ca argument to the address(0), one of the contract addresses can then be set to the address(0) depending on the cn argument value, this will result in a denial of service in one or multiple functionalities of the contract. 2
- The admin sets mstoken, claddress, cloracle or the cl_to_nativecoin_address argument to the address(0), which will result in a denial of service in one or multiple functionalities of the contract and generate unexpected behaviors.

- The admin sets the neworacle argument to the address(0), which will result in a denial of service in one or multiple functionalities of the contract and generate unexpected behaviors.

Files Affected:

SHB.14.1: TestingContract.sol

```
function changeContractAddress(uint cn, address ca) public {
    if (cn == 1){
        buyContract = ca;
    } else if (cn == 2){
         buyWithNativeCoinContract = ca;
        } else if (cn == 3){
           deployNewPoolPairContract = ca;
        } else if (cn == 4){
           finishPoolPairContract = ca;
        } else if (cn == 5){
            autoTwoSidedLiquidityContract = ca;
        } else if (cn == 6){
manualTwoSidedLiquidityContract = ca;
        } else if (cn == 7){
         finishLiquidityContract = ca;
        } else if (cn == 8){
           confirmRemoveBothSidesLiqContract = ca;
        } else if (cn == 9){
           addSupportedChainsContract = ca;
        } else if (cn == 10){
            acknowledgeFinishLiquidityContract = ca;
        } else if (cn == 11){
governanceContract = ca;
        } else if (cn == 12){
           singleSidedLiquidityContract = ca;
        } else if (cn == 13){
            cancelManualEscrowContract = ca;
        } else if (cn == 14){
            pingContract = ca;
        }
}
```

SHB.14.2: OverallContract.sol

```
constructor(uint chain_id, bool _isCentral, address mstoken, uint proposalLifespan, uint8 _internalchainid, address claddress, address cloracle, uint _clfee, address cl_to_nativecoin_address){
_admin = msg.sender;
txnum = 0;
pairTracker = 0;
chainid = chain_id;
defaultTipMultiplier = 2;
//defaultTipAlternate should be set with the (updateAlternateTipDefault -> fulfillAltPrice) function sequence before using/activating alternate tip payment
defaultTipAlternate = 100000 ether;
//atlernatetip is divided by 2, so a value of 3 is effectively 150%
alternateTipMult = 3;
centralContract = _isCentral;
_fee = 30;
_refbonusmultiplier = 10;
_morphswaptoken = IERC20(mstoken);
_morphswaptoken.approve(address(this), type(uint256).max);
_morphswaptokenaddress = mstoken;
_proposalLifespan = proposalLifespan;
internalchainid = _internalchainid;
_claddress = claddress;
setChainlinkToken(_claddress);
setChainlinkOracle(cloracle);
//clfee should be in the form of no decimals (eg 100000000000000000 instead of 0.1)
clfee = _clfee;
```

```
//FIX
//make sure each jobid has the requesting chain's internal chain id
internalchainid_to_chainid[internalchainid] = chain_id;
//FIX
//FIX
chainid_to_internalchainid[chain_id] = internalchainid;
_swapminingfee = (_clfee*11)/10;
one_quadrillion = 1000000000000000;
priceFeed = AggregatorV3Interface(cl_to_nativecoin_address);
```

SHB.14.3: OverallContract.sol

```
function setOracleAddress(address neworacle) public returns (bool) {
    require(msg.sender == _admin);
    _oracle = neworacle;
    return true;
 }
```

Recommendation:

We recommend that you make sure the addresses provided in the arguments are different from the address(0).

Updates

The Morphswap team acknowledged the risk, stating that there are instances in which functionalities must be disabled for security reasons.

SHB.15 Floating Pragma

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

Description:

The contract makes use of the floating-point pragma 0.8. Contracts should be deployed using the same compiler version. Locking the pragma helps ensure that contracts will not unintentionally be deployed using another pragma, which in some cases may be an obsolete version, that may introduce issues to the contract system.

Files Affected:

SHB.15.1: AssetPool.sol

```
pragma solidity ^0.8.0;
```

SHB.15.2: Other Contracts

```
pragma solidity ^0.8.12;
```

Recommendation:

Consider locking the pragma version. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Updates

The Morphswap team resolved the issue by fixing the pragma version to 0.8.12.

SHB.16 Too Many Digits

- Severity : INFORMATIONAL
- Status : Fixed

- Likelihood : 1
- Impact : 0

Description:

There are several places with literals with too many digits. Consider the usage of constants with exponential notation. It will increase the readability of the code and decrease the chance of the typo error in the number of digits.

Files Affected:

SHB.16.1: AssetPool.sol

```
lptosend = 10000000000000000000;
```

SHB.16.2: AssetPool.sol

```
one_quadrillion = 1000000000000000;
```

SHB.16.3: OverallContract.sol

```
one_quadrillion = 1000000000000000;
```

Recommendation:

Consider using the scientific notation to improve readability.

Updates

The Morphswap team resolved the issue by updating the one_quadrillion variable to be equal to 10^{15} and the lptosend to be equal to 10^{19}.

SHB.16.4: AssetPool.sol

```
lptosend = 10**19;
```

SHB.16.5: AssetPool.sol

```
one_quadrillion = 10**15;
```

SHB.16.6: OverallContract.sol

```
one_quadrillion = 10**15;
```

4 Best Practices

BP.1 Remove Duplicated Function Code

Description:

The OverallContract and the PingContract contain an implementation to the same function code, it is recommended to remove the markNewPoolPairComplete function from the PingContract.

Files Affected:

BP.1.1: OverallContract.sol

```
function markNewPoolPairComplete(uint64 _pid) external returns (bool){
   require(msg.sender == address(this));
   require(idToPair[_pid].isValid != true);
   require(cid_c1a_c2a[idToPair[_pid].otherchain][idToPair[_pid].
       ↪ thischainasset][idToPair[_pid].otherchainasset].isValid !=
       ↪ true);
   idToPair[_pid].isValid = true;
   cid_c1a_c2a[idToPair[_pid].otherchain][idToPair[_pid].thischainasset
       ↪ ][idToPair[_pid].otherchainasset] = idToPair[_pid];

   emit AcknowledgedFinishedPair(_pid, idToPair[_pid].icid, idToPair[
       ↪ _pid].thischainasset, idToPair[_pid].otherchainasset);
   return true;
}
```

BP.1.2: PingContract.sol

```
function markNewPoolPairComplete(uint64 _pid) external returns (bool){
   require(msg.sender == address(this));
   require(idToPair[_pid].isValid != true);
   require(cid_c1a_c2a[idToPair[_pid].otherchain][idToPair[_pid].
       ↪ thischainasset][idToPair[_pid].otherchainasset].isValid !=
        ↪ true);
    idToPair[_pid].isValid = true;
    cid_c1a_c2a[idToPair[_pid].otherchain][idToPair[_pid].thischainasset
        ↪ ][idToPair[_pid].otherchainasset] = idToPair[_pid];

    emit AcknowledgedFinishedPair(_pid, idToPair[_pid].icid, idToPair[
        ↪ _pid].thischainasset, idToPair[_pid].otherchainasset);
    return true;
}
```

Status - Not Fixed

BP.2 Write error messages in require statements

Description:

The code contains multiple require statements that revert the transaction when the condition is not met, and throws an error, however most of the require statements do not have error messages, it is recommended to add custom error messages in all the cases in order to make the debugging easier and the code more understandable.

BP.2.1: Example

```
    require(msg.sender == _admin,"Only the Admin can call this function");
```

Files Affected:

All Contracts.

Status - Not Fixed

BP.3 Remove Zero Initialization

Description:

In solidity, there is no need to initialize a variable with its default value, this is done automatically after the variable declaration.

Files Affected:

BP.3.1: OverallContract.sol

```
    constructor(uint chain_id, bool _isCentral, address mstoken, uint
        ↪ proposalLifespan, uint8 _internalchainid, address claddress,
        ↪ address cloracle, uint _clfee, address
        ↪ cl_to_nativecoin_address){
    _admin = msg.sender;
    txnum = 0;
    pairTracker = 0;
```

BP.3.2: AssetPool.sol

```
    constructor(address c1a, uint pid, bool istippool) {
        _poolAsset = c1a;
        _name = 'MorphSwap LP';
        _symbol = 'MSLP';
        _totalSupply = 0;
```

Status - Not Fixed

BP.4 Rename Variables And Functions

Description:

When you are naming a function, variable or a contract, You should think of a name as a tiny comment you put in your code. The key idea when naming something is to convey as much information as possible.

- Choose a word with meaning (provide some context)
- Avoid generic names (like tmp)
- Attach additional information to a name (use suffix or prefix)
- Don't make your names too long or too short
- Use consistent formatting

Status - Not Fixed

BP.5 Remove Commented/Dead code

Description:

The project's codebase contains a lot of commented code, it is recommended that you either uncomment it to utilize it or remove it.

Status - Not Fixed

BP.6 Optimize the order of struct variables declaration

Description:

Variables in solidity are persisted in storage slots each measuring 256bits or 32bytes. When using a struct, it's recommended to declare small sized data types close to each other in order to reduce the contract size.
Refers to : Storing Structs is costing you gas

Files Affected:

BP.6.1: stackTooDeep_avoider3

```
    struct stackTooDeep_avoider3{
           uint64 pairID;
           uint prexferbal;
           uint pretip_amount;
           uint tipamarg;
           address c2w;
           uint64 secondpairID;
           uint _icid;
           bool altfee;
           bool multichainhop;
           uint c1a_amount;
           address c1a;
           uint c2;
           address c2a;
           uint128 rtxnum;
           uint64 convPairId;
       }
```

BP.6.2: AstackTooDeep_avoider4

```
    struct stackTooDeep_avoider4{
           uint64 pairID;
           address otherchainwallet;
           address thischainpool;
           uint otherchain;
           uint icid;
           uint totalval;
           uint128 sent_tipam;
           uint64 tipratiosend;
           uint128 cur_rtxnum;
           uint64 ratiosend;

       }
```

BP.6.3: poolPair

```
     struct poolPair {
           address thischainasset;
           address thischainpool;
           uint otherchain;
           uint8 icid;
           address otherchainasset;
           uint64 pairid;
           bool isValid;
        }
```

BP.6.4: txobj

```
    struct txobj {
           uint8 method_id;
           uint8 internal_start_chainid;
           uint8 internal_end_chainid;
           uint64 pair_id;
           address finalchain_wallet;
           uint64 secondpair_id;
           address firstchain_asset;
           address finalchain_asset;
           uint64 quadrillionratio;
           uint64 quadrilliontipratio;
           uint128 rtxnum;
           bool alt_fee;
        }
```

BP.6.5: containerone

```
    struct containerone {
           bytes32 _requestId;
           uint8 method_id;
           uint8 internal_start_chainid;
           uint8 internal_end_chainid;
           uint64 pair_id;
```

```
        address finalchain_wallet;
        uint64 secondpair_id;
        address firstchain_asset;
        address finalchain_asset;
        uint64 sentratio;
        uint64 tipratio;
        uint128 rtxnum;
        bool paidwithalt;
        bytes20 swapminer;
    }
```

Status - Not Fixed

BP.7 Make use of the Diamond Proxy Pattern

Description:

Because most of the contracts contain the same events and variables declared each time, causing some confusion and code duplication. We recommend making use of the Diamond Proxy Pattern in order to have unlimited functionalities without needing to worry about the contract size while following the standard and having structured, and well organized code. Please refer to the following Ethereum Improvement Proposal for more information.

Status - Not Fixed

Conclusion:

The morphswap team will implement all of the above mentioned best practices in their up-coming versions.

5 Tests

Because the project lacks unit, integration, and end-to-end tests, we recommend establishing numerous testing methods covering multiple scenarios for all features in order to ensure the correctness of the smart contracts.

6 Conclusion

In this audit, we examined the design and implementation of Morphswap contract and discovered several issues of varying severity. Morphswap team addressed 11 issues raised in the initial report and implemented the necessary fixes, while classifying the rest as a risk with low-probability of occurrence. Shellboxes' auditors advised Morphswap Team to maintain a high level of vigilance and to keep those findings in mind in order to avoid any future complications.

7 Scope Files

7.1 Audit

Files	MD5 Hash
MS_Audit/SingleSidedLiquidityContract.sol	41dfaedcb1b1e10626da30d48944ca77
MS_Audit/PingContract.sol	f51027da78b4ca833e605f0c0f35f487
MS_Audit/BuyWithNativeCoinContract.sol	4ffafce0e239d00d0eb8f0ef9bc16612
MS_Audit/BuyContract.sol	8c26438df01b26f2c291d64e49e03df8
MS_Audit/TestingContract.sol	8d9692e870364226eaa8eec68c5f70bb
MS_Audit/FinishPoolPairContract.sol	210c4756ee8a1a597409e1bdf303eb02
MS_Audit/AssetPool.sol	332c0982d7eb89d5dce9361cc4a33a8a
MS_Audit/AddSupportedChainsContract.sol	72f736ebe7f95700fc238b93f0e0d369
MS_Audit/IERC20.sol	2a13ba773d9de22d48b11e5d8594b7a8
MS_Audit/DeployNewPoolPairContract.sol	5fa554d1989752a812838c90dc31a71a
MS_Audit/OverallContract.sol	607d0d269a84c2e5e7da54b9bf3d1bb6
MS_Audit/extensions/IERC20Metadata.sol	193e175856c30259e7b08fd15745819f
MS_Audit/utils/Context.sol	c4b296fb9a98a645ca52cc72c3fbae06

7.2 Re-Audit

Files	MD5 Hash
MS_Audit/SingleSidedLiquidityContract.sol	b023c4e9c5c336e7c0c1f6e67c213513
MS_Audit/PingContract.sol	4a961aa58b22cbd2e2f0144624712909
MS_Audit/BuyWithNativeCoinContract.sol	21a1a31c18f22f23e913dcc3d7ee31f8
MS_Audit/BuyContract.sol	dffa54a25767b514aeda3eb9f34179bd
MS_Audit/TestingContract.sol	390ac46c3e720f58929688f15da631f3
MS_Audit/FinishPoolPairContract.sol	7d85d2398145178865616cceca99e399
MS_Audit/AssetPool.sol	3432d3ebef269bbc0c2a074451f0f919
MS_Audit/AddSupportedChainsContract.sol	f534aee47fc1e0684bf08fa83bb28eeb
MS_Audit/IERC20.sol	7b8d074bd31c18cc10b2680bd77db24a
MS_Audit/DeployNewPoolPairContract.sol	7fe23148669f1a3d75ee286e880cba92
MS_Audit/OverallContract.sol	f2535d3463c62a80dc7d65189e5b0881
MS_Audit/extensions/IERC20Metadata.sol	be3e852a27fc410a51da4e5672c620be
MS_Audit/utils/Context.sol	56a1c7f1985e1ed5557f05387854d9fb

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