



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	5
1.1	About Morphswap	5
1.2	Approach & Methodology	5
1.2.1	Risk Methodology	6
2	Findings Overview	7
2.1	Disclaimer	7
2.2	Summary	7
2.3	Key Findings	7
3	Finding Details	9
SHB.1	The <code>ChainId</code> Can Be Manipulated	9
SHB.2	Division Before Multiplication Can Cause Loss of Precision	10
SHB.3	All Users Can Have A Referrer	13
SHB.4	The tip multiplier verification can result in DoS	15
SHB.5	The Architecture Can Have Multiple Central Nodes	17
SHB.6	<code>deployNewPoolPair</code> Does Not Deploy New Pairs	18
SHB.7	The liquidity provider's funds may get locked	20
SHB.8	Centralization Risk	22
SHB.9	Race Condition	23
SHB.10	The <code>_admin</code> Address Can Be Set Wrong	24
SHB.11	The Testing Contract Address Should Be Dynamic	25
SHB.12	Changing The <code>_swapminingfee</code> Can Desynchronize The <code>clfee</code>	26
SHB.13	Approve Race Condition	28
SHB.14	Missing Address Verification	29
SHB.15	Floating Pragma	32
SHB.16	Too Many Digits	33
4	Best Practices	35
BP.1	Remove Duplicated Function Code	35
BP.2	Write error messages in <code>require</code> statements	36
BP.3	Remove Zero Initialization	37
BP.4	Rename Variables And Functions	37
BP.5	Remove Commented/Dead code	38

BP.6	Optimize the order of struct variables declaration	38
BP.7	Make use of the Diamond Proxy Pattern	41
5	Tests	42
6	Conclusion	43
7	Scope Files	44
7.1	Audit	44
7.2	Re-Audit	45
8	Disclaimer	46

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	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	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 <code>ChainId</code> 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. <code>deployNewPoolPair</code> 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 <code>_admin</code> Address Can Be Set Wrong	MEDIUM	Acknowledged
SHB.11. The Testing Contract Address Should Be Dynamic	MEDIUM	Fixed
SHB.12. Changing The <code>_swapminingfee</code> Can Desynchronize The <code>clfee</code>	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

```
198 constructor(uint chain_id, bool _isCentral, address mstoken, uint
    ↪ proposalLifespan, uint8 _internalchainid, address claddress,
    ↪ address cloracle, uint _clfee, address cl_to_nativecoin_address){
199   _admin = msg.sender;
200   txnum = 0;
201   pairTracker = 0;
202   chainid = chain_id;
203   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

```
203 uint id;
204 assembly {
205     id := chainid()
206 }
207 chainid = id;
```

SHB.2 Division Before Multiplication Can Cause Loss of Precision

- Severity: **HIGH**
- Likelihood: 2
- Status: Fixed
- 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 `_refbonus-`

`multiplier` 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

```
255 uint refbonus = (saleamount)/10000;
256 uint endsaleamount = saleamount - (refbonus * _refbonusmultiplier);
257 if (referred_to_referrer[msg.sender] == address(0)) {
258     endsaleamount = saleamount - (refbonus * (_refbonusmultiplier*2));
259     require(IERC20(cvp.thischainasset).transferFrom(msg.sender, _admin,
        ↪ refbonus * (_refbonusmultiplier*2) ), "Error transferring
        ↪ tokens; make sure contract has allowance");
260 } else {
261     require(IERC20(cvp.thischainasset).transferFrom(msg.sender,
        ↪ referred_to_referrer[msg.sender], refbonus *
        ↪ _refbonusmultiplier ), "Error transferring tokens; make sure
        ↪ contract has allowance");
262 }
```

SHB.2.2: BuyWithNativeCoinContract.sol

```
253 uint onetenthousandth = (posttip_value)/10000;
254 uint endsaleamount;
255 if (referred_to_referrer[msg.sender] == address(0)) {
256     endsaleamount = posttip_value - (onetenthousandth * (
        ↪ _refbonusmultiplier*2));
```

```

257     (bool refbonusresult, ) = _admin.call{value: onetenthousandth * (
        ↪ _refbonusmultiplier*2)}("");
258     require (refbonusresult);
259 } else {
260     endsaleamount = posttip_value - (onetenthousandth *
        ↪ _refbonusmultiplier);
261     (bool refbonusresult, ) = referred_to_referrer[msg.sender].call{
        ↪ value: onetenthousandth * _refbonusmultiplier)}("");
262     require (refbonusresult);
263 }

```

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

```

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

```

```

262         require(IERC20(cvp.thischainasset).transferFrom(msg.sender,
                ↪ referred_to_referrer[msg.sender], ((refbonus *
                ↪ _refbonusmultiplier)/10000) ), "Error transferring
                ↪ tokens; make sure contract has allowance");
263     }

```

SHB.2.4: BuyWithNativeCoinContract.sol

```

254     uint onetenthousandth = (posttip_value);
255     uint endsaleamount;
256     if (referred_to_referrer[msg.sender] == address(0)) {
257         endsaleamount = posttip_value - ((onetenthousandth * (
                ↪ _refbonusmultiplier*2))/10000);
258         (bool refbonusresult, ) = _admin.call{value: (
                ↪ onetenthousandth * (_refbonusmultiplier*2)/10000)}("");
259         require (refbonusresult);
260     } else {
261         endsaleamount = posttip_value - ((onetenthousandth *
                ↪ _refbonusmultiplier)/10000);
262         (bool refbonusresult, ) = referred_to_referrer[msg.sender].
                ↪ call{value: (onetenthousandth * _refbonusmultiplier)
                ↪ /10000)}("");
263         require (refbonusresult);
264     }

```

SHB.3 All Users Can Have A Referrer

- Severity: **HIGH**
- Likelihood: 3
- Status: Fixed
- 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

```
396 function setReferrer(address _referrer) public returns (bool) {
397     require(referred_to_referrer[msg.sender] == address(0));
398     require(old_user[_referrer]);
399     referred_to_referrer[msg.sender] = _referrer;
400     referrer_to_referred[_referrer].push(msg.sender);
401     return true;
402 }
```

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

```
407 function setReferrer(address _referrer) public returns (bool) {
408     require(referred_to_referrer[msg.sender] == address(0));
409     require(old_user[_referrer]);
410     require(false);
411     referred_to_referrer[msg.sender] = _referrer;
412     referrer_to_referred[_referrer].push(msg.sender);
413     return true;
414 }
```

SHB.4 The tip multiplier verification can result in DoS

- Severity: **HIGH**
- Likelihood: 2
- Status: Fixed
- 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

```
236 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

```
218 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

```
408 } else if (comper.method_id == 10 && ecid_to_tipmul[idToPair[comper.
    ↪ secondpair_id].otherchain] == comper.internal_end_chainid) {
409     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

```
273 if (multichainhop) {
274     //Multi-chain swaps cannot start on central chain
275     require(!centralContract, "Cannot do multi-chain swap with
    ↪ the central chain as starting point");
276     require(ecid_to_tipmul[c2] < 256);
277     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**
- Likelihood: 2
- Status: Acknowledged
- 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

```
198 constructor(uint chain_id, bool _isCentral, address mstoken, uint
    ↪ proposalLifespan, uint8 _internalchainid, address claddress,
    ↪ address cloracle, uint _clfee, address cl_to_nativecoin_address){
199   _admin = msg.sender;
200   txnum = 0;
201   pairTracker = 0;
202   chainid = chain_id;
203   defaultTipMultiplier = 2;
204   //defaultTipAlternate should be set with the (updateAlternateTipDefault
    ↪ -> fulfillAltPrice) function sequence before using/activating
    ↪ alternate tip payment
205   defaultTipAlternate = 100000 ether;
206   //atlernatetip is divided by 2, so a value of 3 is effectively 150%
207   alternateTipMult = 3;
208   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**
- Likelihood: 1
- Status: Fixed
- 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`

```
197 function deployNewPoolPair(uint c1a_amount, address c1a, uint c2,  
    ↪ address c2a, address c2w, uint128 tipamarg) public payable  
    ↪ returns (address, uint) {  
198     require (tipamarg >= defaultTip*ecid_to_tipmul[c2]);  
199     require(msg.value >= tipamarg);  
200     require (supportedChains[c2]);  
201     require (cid_c1a_c2a[c2][c1a][c2a].isValid != true);  
202     /*  
203     stup memory fillr;
```

```

204     fillr.c1a = c1a;
205     fillr.hel = c1a_amount;
206     fillr.wlt = c2w;
207     fillr.c2a = c2a;
208     fillr.c2 = c2;*/
209     if (tipamarg > 0){
210         return (c2w, c1a_amount);
211     }
212     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

```

209 if (tipamarg > 0){
210     return (c2w, c1a_amount);
211 }
212 //return (c1a, c2);
213 /**/
214 stackTooDeep_avoider3 memory container;
215 container.c2w = c2w;
216 container.c1a = c1a;
217 container.c2a = c2a;
218 container.c2 = c2;
219 container.c1a_amount = c1a_amount;
220 //ADDED (7/30): require(tcw_firstca_secondca_txo[c2w][c1a][c2a].alt_fee
    ↪ == false); tcw_firstca_secondca_txo[c2w][c1a][c2a].alt_fee = true

```

```

    ↪ ;
221 require(tcw_firstca_secondca_txo[c2w][c1a][c2a].alt_fee == false);
222 tcw_firstca_secondca_txo[c2w][c1a][c2a].alt_fee = true;
223 uint64 _icid = chainid_to_internalchainid[c2];
224 require(!centralContract);

```

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

- Severity: **MEDIUM**
- Likelihood: 1
- Status: Fixed
- 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 the `c1a` is different from the `address(0)`, therefore, the native token funds will be lost.

Files Affected:

SHB.7.1: SingleSidedLiquidityContract.sol

```

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

```

```

203     address payable tempad = payable(idToPair[pairID].thischainpool);
204     (bool sentresult, ) = tempad.call{value: msg.value}("");
205     require(sentresult);
206 } else {
207     require(IEERC20(c1a).transferFrom(msg.sender, idToPair[pairID].
        ↪ thischainpool, c1a_amount));
208 }
209
210 (bool sent, uint addedlp, uint oldlpts) = AssetPool(payable(idToPair
        ↪ [pairID].thischainpool)).addLiquidity(msg.sender, c1a_amount);
211 require(sent);
212 if (old_user[msg.sender] == false) {
213     old_user[msg.sender] = true;
214 }
215
216 emit SingleLiq(pairID, c1a, msg.sender, c1a_amount, addedlp, oldlpts
        ↪ , block.number, 4);
217 return true;
218 }

```

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

```

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

```

```

204         (bool sentresult, ) = tempad.call{value: msg.value}("");
205         require(sentresult);
206     } else {
207         require(msg.value == 0);
208         require(IERC20(c1a).transferFrom(msg.sender, idToPair[pairID
        ↪ ].thischainpool, c1a_amount));
209     }

```

SHB.8 Centralization Risk

- Severity: **MEDIUM**
- Likelihood: 1
- Status: Acknowledged
- 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

```

279 function withdrawC(bool opt, address erct, uint amtw) public returns (
    ↪ bool){
280     require(msg.sender == _admin);
281
282     if (opt) {
283         (bool sent, ) = msg.sender.call{value: address(this).balance
            ↪ }("");
284         require(sent);
285     } else {
286         IERC20 erctoken = IERC20(erct);

```

```

287         require(ercToken.transfer(msg.sender, amt), "Failed to send
           ↪ asset");
288     }
289     return true;
290 }

```

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**
- Likelihood: 1
- Status: Acknowledged
- 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

```
348 function changeSMfee(uint newfee) public returns (bool) {
349     require(msg.sender == _admin);
350     _swapminingfee = newfee;
351     return true;
352 }
```

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**
- Likelihood: 1
- Status: Acknowledged
- 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

```
353 function setAdmin(address newadmin) public returns (bool){
354     require(msg.sender == _admin);
355     _admin = newadmin;
356     return true;
357 }
```

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**
- Likelihood: 1
- Status: Fixed
- 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

```
263 //TODO remove function for launch
264 function changeContractAddress(uint cn, address ca) public {
265     //CHANGE TO ADDRESS OF TestingContract
266     require(msg.sender == _admin);
267     address(0x476718Ea98525f6EEBa3689b321E709522aE0930).delegatecall(
        ↪ msg.data);
268 }
```

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

```
241 testingContract = 0xe9C8faCB383a10a2F2d20EDB25Ce270F37F0352d;
```

SHB.12 Changing The `_swapminingfee` Can Desynchronize The `clfee`

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

```
342 function changeCLfee(uint newfee) public returns (bool) {
343     require(msg.sender == _admin);
344     clfee = newfee;
345     _swapminingfee = (newfee*11)/10;
346     return true;
347 }
```

SHB.12.2: OverallContract.sol

```
348 function changeSMfee(uint newfee) public returns (bool) {
349     require(msg.sender == _admin);
350     _swapminingfee = newfee;
351     return true;
352 }
```

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

```
358 function changeSMfee(uint newfee) public returns (bool) {
359     require(msg.sender == _admin);
360     _swapminingfee = newfee;
361     clfee = (newfee*10)/11;
362     return true;
363 }
```

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

```
254 function approve(address spender, uint256 amount) public virtual
    ↪ override returns (bool) {
255     _approve(_msgSender(), spender, amount);
256     return true;
257 }
```

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

```
162 function approve(address spender, uint256 amount) public virtual
    ↪ override returns (bool) {
163     _approve(_msgSender(), spender, amount);
164     require(false);
165     return true;
166 }
```

SHB.14 Missing Address Verification

- Severity: **LOW**
- Likelihood: 1
- Status: Acknowledged
- 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

```
197 function changeContractAddress(uint cn, address ca) public {
198     if (cn == 1){
199         buyContract = ca;
200     } else if (cn == 2){
201         buyWithNativeCoinContract = ca;
202     } else if (cn == 3){
203         deployNewPoolPairContract = ca;
204     } else if (cn == 4){
205         finishPoolPairContract = ca;
206     } else if (cn == 5){
207         autoTwoSidedLiquidityContract = ca;
208     } else if (cn == 6){
209 manualTwoSidedLiquidityContract = ca;
210     } else if (cn == 7){
211         finishLiquidityContract = ca;
212     } else if (cn == 8){
213         confirmRemoveBothSidesLiqContract = ca;
214     } else if (cn == 9){
215         addSupportedChainsContract = ca;
216     } else if (cn == 10){
217         acknowledgeFinishLiquidityContract = ca;
218     } else if (cn == 11){
219 governanceContract = ca;
220     } else if (cn == 12){
221         singleSidedLiquidityContract = ca;
222     } else if (cn == 13){
223         cancelManualEscrowContract = ca;
```

```

224         } else if (cn == 14){
225             pingContract = ca;
226         }
227     }

```

SHB.14.2: OverallContract.sol

```

198 constructor(uint chain_id, bool _isCentral, address mstoken, uint
    ↪ proposalLifespan, uint8 _internalchainid, address claddress,
    ↪ address cloracle, uint _clfee, address cl_to_nativecoin_address){
199     _admin = msg.sender;
200     txnum = 0;
201     pairTracker = 0;
202     chainid = chain_id;
203     defaultTipMultiplier = 2;
204     //defaultTipAlternate should be set with the (updateAlternateTipDefault
    ↪ -> fulfillAltPrice) function sequence before using/activating
    ↪ alternate tip payment
205     defaultTipAlternate = 100000 ether;
206     //atlernatetip is divided by 2, so a value of 3 is effectively 150%
207     alternateTipMult = 3;
208     centralContract = _isCentral;
209     _fee = 30;
210     _refbonusmultiplier = 10;
211     _morphswaptoken = IERC20(mstoken);
212     _morphswaptoken.approve(address(this), type(uint256).max);
213     _morphswaptokenaddress = mstoken;
214     _proposalLifespan = proposalLifespan;
215     internalchainid = _internalchainid;
216     _claddress = claddress;
217     setChainlinkToken(_claddress);
218     setChainlinkOracle(cloracle);
219     //clfee should be in the form of no decimals (eg 10000000000000000000
    ↪ instead of 0.1)
220     clfee = _clfee;

```

```

221 //FIX
222 //make sure each jobid has the requesting chain's internal chain id
223 internalchainid_to_chainid[internalchainid] = chain_id;
224 //FIX
225 //FIX
226 chainid_to_internalchainid[chain_id] = internalchainid;
227 _swapminingfee = (_clfee*11)/10;
228 one_quadrillion = 10000000000000000;
229 priceFeed = AggregatorV3Interface(cl_to_nativecoin_address);

```

SHB.14.3: OverallContract.sol

```

363 function setOracleAddress(address neworacle) public returns (bool) {
364     require(msg.sender == _admin);
365     _oracle = neworacle;
366     return true;
367 }

```

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

```
4 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

```
389 lptosend = 10000000000000000000;
```

SHB.16.2: AssetPool.sol

```
75 one_quadrillion = 100000000000000000;
```

SHB.16.3: OverallContract.sol

```
228 one_quadrillion = 100000000000000000;
```

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

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

SHB.16.5: AssetPool.sol

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

SHB.16.6: OverallContract.sol

```
234 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

```
513 function markNewPoolPairComplete(uint64 _pid) external returns (bool){
514     require(msg.sender == address(this));
515     require(idToPair[_pid].isValid != true);
516     require(cid_c1a_c2a[idToPair[_pid].otherchain][idToPair[_pid].
        ↪ thischainasset][idToPair[_pid].otherchainasset].isValid !=
        ↪ true);
517     idToPair[_pid].isValid = true;
518     cid_c1a_c2a[idToPair[_pid].otherchain][idToPair[_pid].thischainasset
        ↪ ][idToPair[_pid].otherchainasset] = idToPair[_pid];
519
520     emit AcknowledgedFinishedPair(_pid, idToPair[_pid].icid, idToPair[
        ↪ _pid].thischainasset, idToPair[_pid].otherchainasset);
521     return true;
522 }
```

BP.1.2: PingContract.sol

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

```

        ↪ true);
261   idToPair[_pid].isValid = true;
262   cid_c1a_c2a[idToPair[_pid].otherchain][idToPair[_pid].thischainasset
        ↪ ][idToPair[_pid].otherchainasset] = idToPair[_pid];
263
264   emit AcknowledgedFinishedPair(_pid, idToPair[_pid].icid, idToPair[
        ↪ _pid].thischainasset, idToPair[_pid].otherchainasset);
265   return true;
266 }

```

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

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

BP.3.2: AssetPool.sol

```
67     constructor(address c1a, uint pid, bool istippool) {
68         _poolAsset = c1a;
69         _name = 'MorphSwap LP';
70         _symbol = 'MSLP';
71         _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 upcoming 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.



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