EmiSwap Smart Contract Preliminary Audit Report

Project Synopsis

Project Name	**EmiSwap**
Platform	Ethereum, Solidity
Github Repo	https://github.com/EMISWAP-COM/emiswap/tree/master/contracts
Deployed Contract	Not Deployed
Total Duration	15 Days
Timeline of Audit	**17th May 2021 to 3rd June 2021**

Contract Details

Total Contract(s)	5
Name of Contract(s)	EMIVamp, EMIVault, EMISwapLib, ESW, EMIFactory, EMIPrice, EMIRefferal, EMINFT, TimeLock, VotableProxyAdmin, Sqrt, UniERC20, TransferHelper
Language	Solidity
Commit Hash	ab5221643be9d79b335573bc1a271fb4e43f6b60

Contract Vulnerabilities Synopsis

Issues	Open Issues	Closed Issues
Critical Severity	2	0
Medium Severity	3	0
Low Severity	5	0
Information	1	0
Total Found	11	0

Detailed Results

The contract has gone through several stages of the audit procedure that includes structural analysis, automated testing, manual code review etc.

All the issues have been explained and discussed in detail below. Along with the explanation of the issue found during the audit, the recommended way to overcome the issue or improve the code quality has also been mentioned.

High Severity Issues

A.1 Contract uses Uninitialized State Variables

SWC Reference : SWC 131 – Presence of unused variables
CWE Reference : CWE 1164 - Irrelevant Code

Contract - EMIVamp.sol
Line no - 36, 151

Description:

The **EMIVamp** contract uses some imperative State Variables that have never been initialized throughout the contract.

Mentioned below are the specific lines where the State Variable has been declared and used but never initialized:

- **_voting (***Line 36***) state variable** is being declared.

```
36      address private _voting;
37
38      // !!!In updates to contracts se
```

- ***changeFactory function (****Line 151****)- _voting*** *variable* is being used in the **changeFactory function** without ever being initialized with a specific address.

```
148     function changeFactory(uint256 _proposalId) external onlyAdmin {
149         address _newFactory;
150
151         _newFactory = IEmiVoting(_voting).getVotingResult(_proposalId);
152         require(_newFactory != address(0), "EmiVamp: New factory address is wrong");
153
```

Recommendation:

State variable ***_voting,*** must be initialized within the constructor before being used in any particular function, to avoid any unwanted scenario during function executions.

A.2 State Variables used but never initialized in ESW Contract

SWC Reference : SWC 131 – Presence of unused variables
CWE Reference : CWE 1164 - Irrelevant Code

Contract - ESW.sol
Line no - 36, 151

Description:

The **EMIVamp** contract accesses some imperative State Variable that has never been initialized throughout the contract.

Mentioned below are the specific lines where the State Variable has been declared and used but never initialized:
State variables are being declared at:

- ***vesting* (***Line 12***)**
- ***initialSupply* (**Line 13***)*

```
12      address public vesting;
13      uint256 internal _initialSupply;
```

State variables are being used at the following lines without being initialized throughout the contract

- ***burnFromVesting at Line 96***
- ***initialSupply at Line 135***

```
95      function burnFromVesting(uint256 amount) external {
96          require(msg.sender == vesting, "Only vesting!");
97          burn(amount);
98      }
```

```
134     function initialSupply() public view returns (uint256) {
135         return _initialSupply;
136     }
```

Recommendation:
The above-mentioned state variables must be initialized within the constructor before being used in any particular functions, to avoid any unwanted scenario during function executions.

Medium Severity Issues

A.3 Require Statement Validation contains a tautology or contradiction

Contract - EMIVamp
SWC Reference : SWC 123 – Requirement Violation
CWE Reference : CWE 345 - Insufficient Verification of Data Authenticity

Line - 381-384

Explanation:

The following function in the contract includes a **require statement** that might contain some contradictory condition.

- ***addLiquidity at Line 382 - 384***

```
381             require(
382                 amountBOptimal >= 0,
383                 "EmiswapRouter: INSUFFICIENT_B_AMOUNT"
384             );
```

The condition ensures that the value of **amountBOptimal** is either greater than or equal to ZERO.

Is this Intended?

Recommended:

If the above-mentioned scenario was not intended, the **require** statement should be modified accordingly.

For instance:

```
require(amountBOptimal > 0,"EmiswapRouter: INSUFFICIENT_B_AMOUNT");
```

A.4 Loops are extremely Costly

SWC Reference : SWC 128 – DoS With Block Gas Limit

CWE Reference : CWE 400 - Uncontrolled Resource Consumption

Contract - EMIPrice

Explanation:

EMIPrice contract includes a **for loop** that uses state variables like ***.length*** of a non-memory array, in the condition of the for loops.

As a result, these state variables consume a lot more extra gas for every iteration of the for loop.

The following function includes such loops at the mentioned lines of the contracts mentioned below:

a. **EMIPrice Contract**

- ***_getUniswapPrice at Line 94***
- ***_getOurPrice at Line 122***
- ***_get1inchPrice at Line 146***

Recommendation:

It's quite effective to use a local variable instead of a state variable like .length in a loop. This will be a significant step in optimizing gas usage. For instance,

```
local_variable =  _coins.length;
 for (uint256 i = 0; i <local_variable;  i++) {
         (_prices[i], ) = _factory.getExpectedReturn(
            IERC20(_coins[i]),
            IERC20(_DAI),
            10**uint256(ERC20(_coins[i]).decimals()),
            1,
            0
  );
```

A.5 Multiplication is being performed on the result of Division

Contract - EMISwapLib

SWC Reference : SWC 101 – Integer Overflow and Underflow

CWE Reference : CWE 682 - Incorrect Calculation

Line no - 533-573

***Explanation*:**

During the automated testing of the contracts, it was found that some of the functions in the contracts are performing multiplication on the result of a Division.

Integer Divisions in Solidity might truncate. Moreover, this performing division before multiplication might lead to loss of precision.

The following functions involve division before multiplication in the mentioned lines of the respective contracts:

a. **EMISwapLib**

- *getAmountsIn - 74-77*
- *getAmountsOUt - 94-95*

```
EmiswapLib.getAmountIn(address,uint256,uint256,uint256) (flat/myEmiVAMP_Flat.sol#1036-1053) performs a multiplication on the result of a division:
        -denominator = reserveOut.sub(amountOut).mul(uint256(1000000000000000000).sub(fee(factory)).div(1e15)) (flat/myEmiVAMP_Flat.sol#1048-1051)
EmiswapLib.getAmountOut(address,uint256,uint256,uint256) (flat/myEmiVAMP_Flat.sol#1056-1075) performs a multiplication on the result of a division:
        -amountInWithFee = amountIn.mul(uint256(1000000000000000000).sub(fee(factory)).div(1e15)) (flat/myEmiVAMP_Flat.sol#1066-1069)
```

Recommendation:

Solidity doesn't encourage arithmetic operations that involve division before multiplication. Therefore the above-mentioned functions should be checked once and redesigned if they do not lead to expected results.

Low Severity Issues

A. 6 Return Value of an External Call is never used Effectively

Contract - EMIVault

SWC Reference : SWC 104 – Unchecked Call Return Value

CWE Reference : CWE 252 - Unchecked Return Value

Line no - 78

Explanation:

The external calls made in the above-mentioned lines do return a boolean value that indicates whether or not the external call made was successful. These boolean return values can be used in the function as a check to ensure that the further execution of the function is only allowed if the external is successfully made.

However, the **EMIVault** contract never uses these return values throughout the contract.

```
77          for (uint256 index = 0; index < tokenAddresses.length; index++) {
78              IERC20(tokenAddresses[index]).transfer(recipient, amounts[index]);
79          }
```

Recommendation:

Effective use of all the return values from external calls must be ensured within the contract.

A.7 Contract uses Hardcoded addresses

Contract - ESW.sol

CWE Reference : CWE 665 - Improper Initialization

Line no - 333

Explanation:

Keeping in mind the immutable nature of smart contracts, it is not considered a better practise to hardcode any address or imperative uint in the contract before deployment.

```
22      address public constant firstMinter =
23          0xdeb5A983AdC9b25b8A96ae43a65953Ded3939de6; // set to Oracle
24      address public constant secondMinter =
25          0x9Cf73e538acC5B2ea51396eA1a6DE505f6a68f2b; //set to EmiVesting
```

However, during the audit procedure it was found that certain addresses in the following contracts were hardcoded.

Recommendation:
It is recommended to use a state variable for storing such an integer and initialize them in the constructor.

A.8 Too many Digits used

Contract -EMISwapLib
SWC Reference : SWC 135 – Code With No Effects
CWE Reference : CWE 1164 - Irrelevant Code

Description:

The above-mentioned lines have a large number of digits that makes it difficult to review and reduce the readability of the code.

The following State Variables/Functions of respective contracts mentioned below include large digits:

a. EMISwapLib

- *getAmountsIn - 76*
- *getAmountsOUt - 94*

Recommendation:

Ether Suffix couldshutt be used to symbolize the 10^18 zeros.

A.9 External Visibility should be preferred

SWC Reference : SWC 100 – Function Visibility
CWE Reference : CWE 710 - Improper Adherence to Coding Standards

Contract - EmiVamp, EMIVault, ESW, EMIRefferal

Explanation:

Those functions that are never called throughout the contract should be marked as ***external*** visibility instead of ***public*** visibility.
This will effectively result in Gas Optimization as well.

Therefore, the following functions of respective contracts mentioned below should must be marked as **external** within the contract:

a. **EmiVamp Contract**
   - ***deposit()***
   - ***isPairAvailable()***

b. **EMIVault**
   - ***getWalletNonce()***
   - ***withdrawTokens()***

c. ***ESW***
   - ***updateTokenName***
   - ***switchMinter()***
   - ***setMintLimit***
   - ***mintSigned***
   - ***initialSupply()***
   - ***getMintLimit()***
   - ***getWalletNonce()***

d. ***EMIRefferal***
   - ***grantRef()***
   - ***revokeRef()***
   - ***setAdminOnce()***
   - ***initialize()***

e. **EMINFT**
   - ***mint()***
   - ***mintBatch***

Recommendation:

If the ***PUBLIC*** visibility of the above-mentioned functions is not intended, then the ***EXTERNAL*** Visibility keyword should be preferred.

A.10 Absence of Zero Address Validation

Contract - EMIVamp

SWC Reference : SWC 123 – Requirement Violation

CWE Reference : CWE 345 - Insufficient Verification of Data Authenticity

Description:

Some of the contracts initialize imperative state variables in the constructor or functions.

However, during the automated testing of the contact it was found that no Zero Address Validation is implemented to ensure that no invalid argument is passed while initializing such state variables.

The following functions in lacks a Zero Address Validations at the lines mentioned in specific contracts:

1. **EMIVamp**
    - *changeReferral at Line 160*

Recommendation:

A **require** statement should be included in such functions to ensure no zero address is passed in the arguments.

Informational

A.11 Coding Style Issues in the Contract

CWE Reference : CWE 710 - Improper Adherence to Coding Standards

Explanation:

Code readability of a Smart Contract is largely influenced by the Coding Style issues and in some specific scenarios may lead to bugs in the future.

During the automated testing, it was found that the following contracts have quite a few code style issues:

a. EMIVamp

```
Parameter EmiVamp.initialize(address[],uint8[],address,address)._types (flat/myEmiVAMP_Flat.sol#1199) is not in mixedC
Parameter EmiVamp.initialize(address[],uint8[],address,address)._ourfactory (flat/myEmiVAMP_Flat.sol#1200) is not in m
Parameter EmiVamp.initialize(address[],uint8[],address,address)._ourvoting (flat/myEmiVAMP_Flat.sol#1201) is not in mi
Parameter EmiVamp.lpTokenDetailedInfo(uint256)._pid (flat/myEmiVAMP_Flat.sol#1229) is not in mixedCase
Parameter EmiVamp.addLPToken(address,uint16)._token (flat/myEmiVAMP_Flat.sol#1253) is not in mixedCase
Parameter EmiVamp.addLPToken(address,uint16)._tokenType (flat/myEmiVAMP_Flat.sol#1253) is not in mixedCase
Parameter EmiVamp.removeLPToken(uint256)._idx (flat/myEmiVAMP_Flat.sol#1276) is not in mixedCase
Parameter EmiVamp.changeLPToken(uint256,address,uint16)._idx (flat/myEmiVAMP_Flat.sol#1286) is not in mixedCase
Parameter EmiVamp.changeLPToken(uint256,address,uint16)._token (flat/myEmiVAMP_Flat.sol#1287) is not in mixedCase
Parameter EmiVamp.changeLPToken(uint256,address,uint16)._tokenType (flat/myEmiVAMP_Flat.sol#1288) is not in mixedCase
Parameter EmiVamp.changeFactory(uint256)._proposalId (flat/myEmiVAMP_Flat.sol#1301) is not in mixedCase
Parameter EmiVamp.changeReferral(address)._ref (flat/myEmiVAMP_Flat.sol#1313) is not in mixedCase
Parameter EmiVamp.deposit(uint256,uint256)._pid (flat/myEmiVAMP_Flat.sol#1321) is not in mixedCase
Parameter EmiVamp.deposit(uint256,uint256)._amount (flat/myEmiVAMP_Flat.sol#1321) is not in mixedCase
Parameter EmiVamp.isPairAvailable(address,address)._token0 (flat/myEmiVAMP_Flat.sol#1466) is not in mixedCase
Parameter EmiVamp.isPairAvailable(address,address)._token1 (flat/myEmiVAMP_Flat.sol#1466) is not in mixedCase
```

b. EMISharp

```
Parameter EmiPrice.initialize(address,address,address,address)._market1 (EmiPrice.Full.sol#435) is not in mixedCase
Parameter EmiPrice.initialize(address,address,address,address)._market2 (EmiPrice.Full.sol#436) is not in mixedCase
Parameter EmiPrice.initialize(address,address,address,address)._market3 (EmiPrice.Full.sol#437) is not in mixedCase
Parameter EmiPrice.initialize(address,address,address,address)._daitoken (EmiPrice.Full.sol#438) is not in mixedCase
Parameter EmiPrice.getCoinPrices(address[],uint8)._coins (EmiPrice.Full.sol#454) is not in mixedCase
Parameter EmiPrice.getCoinPrices(address[],uint8)._market (EmiPrice.Full.sol#454) is not in mixedCase
```

Recommendation:

Therefore, it is highly recommended to fix the issues like naming convention, indentation, and code layout issues in a smart contract.