SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Emiswap
Date: April 6th, 2021

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Emiswap.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Token, Token sale, Exchange, Exchanges aggregator.
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/EMISWAP-COM/emiswap
Commit	THIRD REVIEW COMMIT: 329B5031A362EDD30B2B6470AFDCAD88CB3727E2 FOURTH REVIEW COMMIT: A6A94FFFAA95C0C761DEF8F60C77CE60199A3032
Deployed contract	
Timeline	04 DEC 2020 – 20 JAN 2021
Changelog	09 DEC 2020 – INITIAL AUDIT 23 DEC 2020 – SECOND REVIEW 21 JAN 2021 – THIRD REVIEW 18 FEB 2021 – FOURTH REVIEW 06 APR 2021 – REPORT UPDATE

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Emiswap (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of Customer's smart contract and its code review conducted between December 4th, 2020 - December 9th, 2020.

The second review conducted on December 23rd, 2020.

The third review conducted on January 21, 2021.

The fourth review conducted on February 18, 2021.

Scope

The scope of the project is smart contracts in the repository:

```
Contract deployment address:
Repository
Commit      a6a94fffaa95c0c761def8f60c77ce60199a3032
Files:
      CrowdSale.sol
      EmiFactory.sol
      EmiPrice.sol
      EmiReferral.sol
      EmiRouter.sol
      Emiswap.sol
      EmiVamp.sol
      EmiVault.sol
      EmiVesting.sol
      EmiVoting.sol
      ESW.sol
      VotableProxyAdmin.sol
      EmiswapLib.sol
      Priviledgeable.sol
      ProxiedERC20.sol
      Sqrt.sol
      TransferHelper.sol
      UniERC20.sol
      Timelock.sol
```

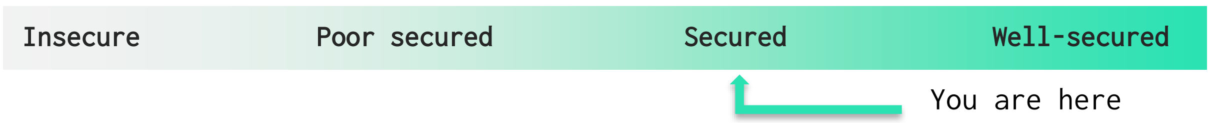
We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence

	▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency ▪ Data Consistency
Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart contracts contained issues that should be fixed. Four iterations of the review were done, during this engagement, many fixes were applied to the code, some business logic was change. Based on the above and the importance of a product we recommend doing 2nd independent Smart Contract audit.



Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. A general overview is presented in AS-IS section, and all found issues can be found in the Audit overview section.

Security engineers found **2** critical **6** high, **9** medium, **4** low and **4** informational issues during the audit.

After the **second** review, the code contains **1** critical, **3** high, **2** medium, **4** low, and **3** informational issues.

After the **third** review, the code contains **4** critical, **8** high, **2** medium and **5** low severity issues.

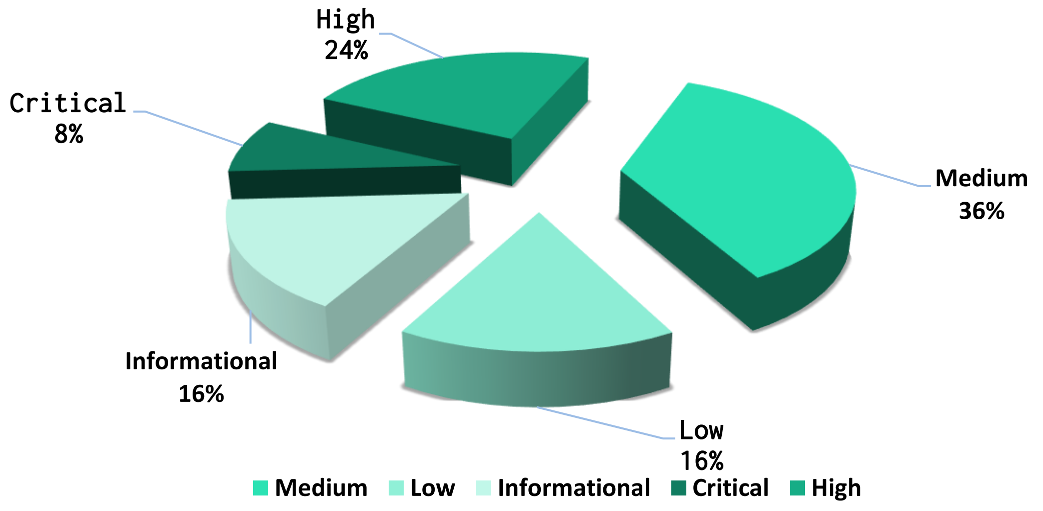
After the **fourth** review, the code contains **0** critical, **0** medium, and **6** low severity issues.

Notice: the overall low-quality development of custom contracts can lead to unexpected, hidden errors.

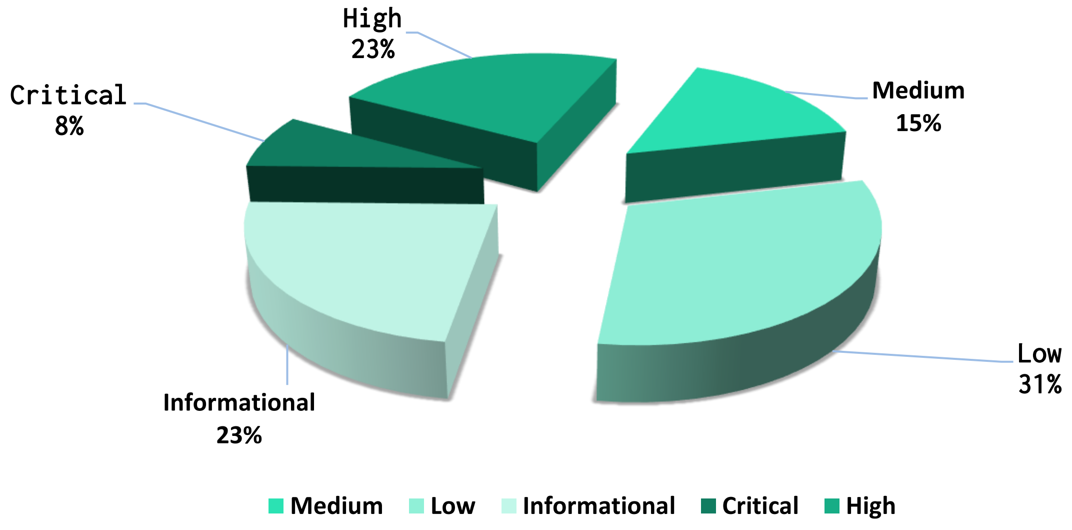
Notice 2: additional reviews do not include a full audit of the provided code. As soon as Emi contracts reviewed 4 times, we may not guaranty their secureness.

Notice 3: tests are failing in the latest version of the code.

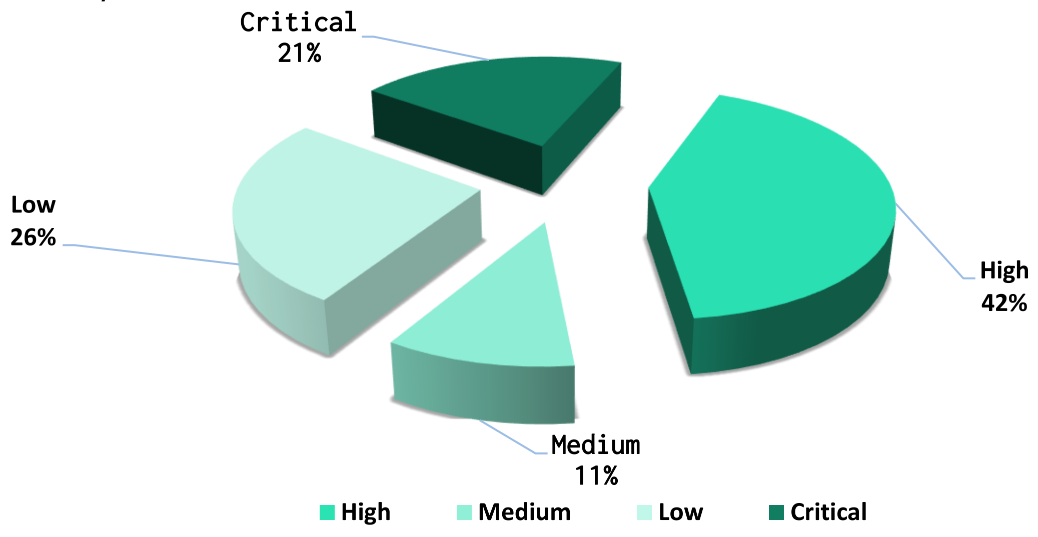
Graph 1. The distribution of vulnerabilities after the first review.



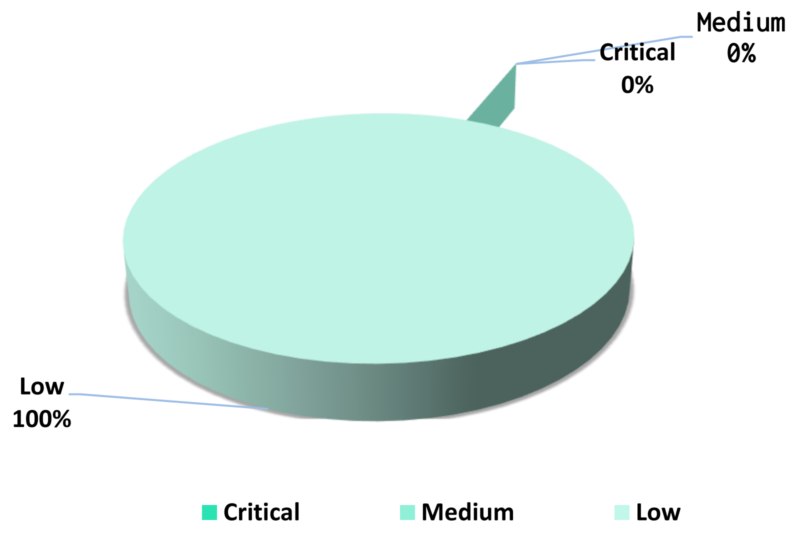
Graph 2. The distribution of vulnerabilities after the second review.



Graph 3. The distribution of vulnerabilities after the third review.



Graph 4. The distribution of vulnerabilities after the fourth review.



Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

AS-IS overview

CrowdSale.sol

Description

CrowdSale is a contract used for the token crowdsale.

Imports

CrowdSale contract has the following imports:

- @openzeppelin/contracts/math/SafeMath.sol
- @openzeppelin/contracts/proxy/Initializable.sol
- @openzeppelin/contracts/token/ERC20/IERC20.sol
- @openzeppelin/contracts/token/ERC20/SafeERC20.sol
- ./interfaces/IEmiReferral.sol
- ./interfaces/IESW.sol
- ./interfaces/IERC20Detailed.sol
- ./uniswapv2/interfaces/IUniswapV2Factory.sol
- ./uniswapv2/interfaces/IUniswapV2Pair.sol
- ./libraries/Priviledgeable.sol

Inheritance

CrowdSale contract is Initializable and Priviledgeable.

Usages

CrowdSale contract has the following custom usages:

- SafeMath for uint256
- SafeMath for uint32
- SafeERC20 for IERC20

Structs

CrowdSale contract has the following data structures:

- Coin - stores coins allowed in sale and their info.

Enums

CrowdSale contract has no custom enums.

Events

CrowdSale contract has the following events:

- Buy

Modifiers

CrowdSale has following modifiers:

- crowdSaleworking - checks whether crowdsale is active.

Fields

CrowdSale contract has following constants:

- mapping(uint16 => Coin) internal _coins
- mapping(address => uint16) public coinIndex
- uint16 internal _coinCounter
- uint32 internal _ratePrecision
- address internal _token
- address internal _wethToken
- address internal _uniswapFactory
- address internal referralStore
- address payable public foundationWallet
- address public teamWallet
- address internal defRef
- string public codeVersion = "CrowdSale v1.0-18-g44fc1eb"
- uint256 public crowdSalePool = 40_000_000e18
- bool public isStoped

Functions

CrowdSale has following public functions:

- ***initialize***
  Description
  Initializes the contract
  Visibility
  public
  Input parameters
    - address eswToken
    - address uniswapFactory
    - address referralStoreInput
    - address wethToken
    - address payable _foundationWallet
    - address _teamWallet
  Constraints
    - `initializer` modifier.
    - All input parameters could not be 0 addresses.

Events emit
None
Output
None

- *updateParams*
  Description
  Updates addresses.
  Visibility
  public
  Input parameters
    - address eswToken
    - address uniswapFactory
    - address referralStoreInput
    - address wethToken
    - address payable _foundationWallet
    - address _teamWallet
    - address _defRef
  Constraints
    - \`onlyAdmin\` modifier.
    - All input parameters could not be 0 addresses.
  Events emit
  None
  Output
  None
- *stopCrowdSale*
  Description
  Changes status of the \`isStoped\` variable to a \`isStopedNewValue\`.
  Visibility
  public
  Input parameters
    - bool isStopedNewValue
  Constraints
    - \`onlyAdmin\` modifier.
  Events emit
  None
  Output
  None
- *setPoolsize*
  Description
  Updates the \`crowdSalePool\` variable.
  Visibility
  public
  Input parameters
    - uint256 _newcrowdSalePoo
  Constraints

- `onlyAdmin` modifier.

Events emit
None
Output
None

- *updateParams*
  Description
  Updates addresses.
  Visibility
  public
  Input parameters
  - address eswToken
  - address uniswapFactory
  - address referralStoreInput
  - address wethToken
  - address payable _foundationWallet
  - address _teamWallet
  - address _defRef

  Constraints
  - `onlyAdmin` modifier.
  - All input parameters could not be 0 addresses.

  Events emit
  None
  Output
  None

- *fetchCoin*
  Description
  Adds a new token that can be accepted in the crowdsale.
  Visibility
  public
  Input parameters
  - address coinAddress
  - uint32 rate
  - uint8 status

  Constraints
  - `onlyAdmin` modifier.
  - A token should not be added yet.

  Events emit
  None
  Output
  None
- *setStatusByID*
  Description
  Updates status of a token.

Visibility
public
Input parameters
   ○ uint16 coinId
   ○ uint8 status
Constraints
   ○ `onlyAdmin` modifier.
Events emit
None
Output
None

- *setRateByID*
Description
Updates an exchange rate of a token.
Visibility
public
Input parameters
   ○ uint16 coinId
   ○ uint32 rate
Constraints
   ○ `onlyAdmin` modifier.
Events emit
None
Output
None
- *getToken*, *coinCounter, coin, coinRate, coinData, getBuyCoinAmountByID*
Description
Simple getter functions.
- *presaleBulkLoad*
Description
Uploads presale info.
Visibility
public
Input parameters
   ○ address[] memory beneficiaries
   ○ uint256[] memory tokens
   ○ uint32[] memory sinceDate
Constraints
   ○ `onlyAdmin` modifier.
Events emit
None
Output
None
- *buyView*
Description

Calculates a required incoming token amount to buy a specified `amount` of ESW token or vise-versa.

Visibility

public

Input parameters

- address coinAddress
- uint256 amount
- bool isReverse

Constraints

None

Events emit

None

Output

- uint256 currentTokenAmount
- uint16 coinId
- uint256 coinAmount

- *buy*

Description

Buy ESW tokens.

Visibility

public

Input parameters

- address coinAddress
- uint256 amount
- address referralInput
- bool isReverse

Constraints

- Crowdsale should be active.
- `amount` should be greater than 0.
- `coinAddress` should be registered.
- The CrowdsaleLimit should not be exceeded.

Events emit

Emits the `Buy` event.

Output

None

- *buyWithETHView*

Description

Calculates a required incoming ETH amount to buy a specified `amount` of ESW token or vise-versa.

Visibility

public

Input parameters

- uint256 amount
- bool isReverse

Constraints

None
Events emit
None
Output
  - uint256 currentTokenAmount
  - uint256 coinA

- *buyWithETH*
  Description
  Buy ESW tokens for ETH.
  Visibility
  public
  Input parameters
  - address referralInput
  - uint256 amount
  - bool isReverse

  Constraints
  - Crowdsale should be active.
  - A msg.value should be greater than 0.
  - The CrowdsaleLimit should not be exceeded.

  Events emit
  Emits the `Buy` event.
  Output
  None

ESW.sol

Description

ESW is an ERC-20 token.

Imports

ESW contract has the following imports:

- @openzeppelin/contracts/proxy/Initializable.sol
- ./interfaces/IEmiVesting.sol
- ./libraries/Priviledgeable.sol
- ./libraries/ProxiedERC20.sol

Inheritance

ESW contract is ProxiedERC20, Initializable and Priviledgeable

Usages

ESW contract has no custom usages.

Structs

ESW contract has no custom data structures.

Enums

ESW contract has no custom enums.

Events

ESW contract has no custom events.

Modifiers

ESW has the following modifiers:

- mintGranted - checks whether a massage sender has the minting role.

Fields

ESW contract has following constants:

- address public dividendToken
- address public vesting
- uint256 internal _initialSupply
- mapping(address => uint256) internal _mintLimit
- mapping(address => bool) internal _mintGranted
- string public codeVersion = "ESW v1.0-18-g44fc1eb"

Functions

ESW has following public functions:

- ***initialize***
  Description
  Initializes the contract.
  Visibility
  public
  Input parameters
  None
  Constraints
    - Can only be called once.
  Events emit
  None
  Output
  None
- ***grantMint***
  Description
  Assigns the minting role.
  Visibility

public
Input parameters
- address _newIssuer

Constraints
- Can only be called by the admin.

Events emit
None
Output
None

- *revokeMint*
  Description
  Revokes the minting role.
  Visibility
  public
  Input parameters
  - address _revokeIssuer

  Constraints
  - Can only be called by the admin.

  Events emit
  None
  Output
  None
- *setVesting*
  Description
  Sets an address of the vesting contract.
  Visibility
  public
  Input parameters
  - address _vesting

  Constraints
  - Can only be called by the admin.

  Events emit
  None
  Output
  None
- *balanceOf2*
  Description
  Returns the token balance plus balance locked on the vesting contract.
  Visibility
  public
  Input parameters
  - address account

  Constraints
  None
  Events emit

None
Output
  - uint256

- *balanceOf2*
  Description
  Returns the token balance plus balance locked on the vesting contract.
  Visibility
  public
  Input parameters
    - address account

  Constraints
  None
  Events emit
  None
  Output
    - uint256
- *getMintLimit*
  Description
  Returns a minting limit of an `account`.
  Visibility
  public view
  Input parameters
    - address account

  Constraints
    - `onlyAdmin` modifier.

  Events emit
  None
  Output
    - uint256
- *setMintLimit*
  Description
  Sets a minting limit of an `account`.
  Visibility
  public
  Input parameters
    - address account
    - uint256 amount

  Constraints
    - `onlyAdmin` modifier.

  Events emit
  None
  Output
  None
- *mintAndFreeze*

Description
Mints and freezes tokens.
Visibility
external
Input parameters
- address recipient
- uint256 amount
- uint256 category

Constraints
- `mintGranted` modifier.

Events emit
None
Output
None

- *mintVirtualAndFreeze*

Description
Freezes an `amount` of virtual tokens.
Visibility
external
Input parameters
- address recipient
- uint256 amount
- uint256 category

Constraints
- `mintGranted` modifier.

Events emit
None
Output
None

- *mintVirtualAndFreezePresale*

Description
Freezes an `amount` of virtual tokens bought during the presale.
Visibility
external
Input parameters
- address recipient
- uint32 sinceDate
- uint256 amount
- uint256 category

Constraints
- `mintGranted` modifier.

Events emit
None
Output

None

- *currentCrowdsaleLimit*
  Description
  Returns a current crowd sale limit.
  Visibility
  external view
  Input parameters
  None
  Constraints
  None
  Events emit
  None
  Output
    - uint256

EmiVesting.sol

Description

EmiVesting is a vesting contract.

Imports

EmiVesting contract has the following imports:

- @openzeppelin/contracts/token/ERC20/IERC20.sol
- @openzeppelin/contracts/math/SafeMath.sol
- @openzeppelin/contracts/token/ERC20/SafeERC20.sol
- @openzeppelin/contracts/proxy/Initializable.sol
- ./interfaces/IEmiVesting.sol
- ./interfaces/IERC20Detailed.sol
- ./libraries/Priviledgeable.sol

Inheritance

EmiVesting contract is Initializable, Priviledgeable, IEmiVesting.

Usages

EmiVesting contract has the following custom usages:

- SafeMath for uint
- SafeMath for uint256
- SafeERC20 for IERC20

Structs

EmiVesting contract has the following data structures:

- LockRecord
- CategoryRecord

Enums

EmiVesting contract has no custom enums.

Events

EmiVesting contract has the following custom events:

- TokensLocked
- TokensClaimed
- TokenChanged

Modifiers

EmiVesting has the no custom modifiers.

Fields

EmiVesting contract has following constants:

- uint32 constant QUARTER = 3 * 43776 minutes
- uint constant WEEK = 7 days
- uint constant CROWDSALE_LIMIT = 40000000e18
- uint constant CATEGORY_COUNT = 12
- uint32 constant VIRTUAL_MASK = 0x80000000
- uint32 constant PERIODS_MASK = 0x0000FFFF
- mapping(address => LockRecord[]) private _locksTable
- mapping(address => CategoryRecord[CATEGORY_COUNT]) private _statsTable
- address public _token
- uint public version
- uint public currentCrowdsaleLimit
- string public codeVersion = "EmiVesting v1.0-18-g44fc1eb"

Functions

EmiVesting has following public functions:

- ***initialize***
  Description
  Initializes the contract.
  Visibility
  public

Input parameters
- address _ESW

Constraints
- Can only be called once.

Events emit
None
Output
None

- *getLock,* **getLocksLen, getStats**
  Description
  Getter functions available only for admins.
- *getNextUnlock,* **getMyLock, getMyLocksLen, getMyStats, unlockedBalanceOf, balanceOf, balanceOfVirtual, getCrowdsaleLimit**
  Description
  Simple view functions.
- *freeze*
  Description
  Freezes an amount `tokens` of the EMS token for a `beneficiary`.
  Visibility
  external
  Input parameters
  - address beneficiary
  - uint tokens
  - uint category

  Constraints
  - onlyAdmin modifier.
  - `beneficiary` should not be zero.
  - `tokens` value should be greater or equal to zero.
  - The `currentCrowdsaleLimit` should not be exceeded.
  - A `category` should exist.

  Events emit
  Emits the TokensLocked event.
  Output
  None
- *freeze*
  Description
  Freezes an amount `tokens` of the EMS token for a `beneficiary`.
  Visibility
  external
  Input parameters
  - address beneficiary
  - uint tokens
  - uint category

Constraints
- onlyAdmin modifier.
- \`beneficiary\` should not be zero.
- \`tokens\` value should be greater or equal to zero.
- The \`currentCrowdsaleLimit\` should not be exceeded.
- A \`category\` should exist.

Events emit
Emits the \`TokensLocked\` event.
Output
None

- *freezeBulk*

Description
Freezes tokens in bulk.
Visibility
external
Input parameters
- address[] calldata beneficiaries
- uint[] calldata sinceDate
- uint[] calldata tokens
- uint category

Constraints
- onlyAdmin modifier.
- All input arrays should be of the same length.

Events emit
Emits multiple \`TokensLocked\` event.
Output
None

- *freezeVirtual*

Description
Freezes a virtual amount \`tokens\` of the EMS token for a \`beneficiary\`.
Visibility
external
Input parameters
- address beneficiary
- uint tokens
- uint category

Constraints
- onlyAdmin modifier.
- \`beneficiary\` should not be zero.
- \`tokens\` value should be greater or equal to zero.
- The \`currentCrowdsaleLimit\` should not be exceeded.
- A \`category\` should exist.

Events emit
None

Output
None

- *claim*
  Description
  Claim available tokens.
  Visibility
  external
  Input parameters
  None
  Constraints
  None
  Events emit
  Emits the `TokensClaimed` event.
  Output
  bool
- *changeToken*
  Description
  Changes the token address.
  Visibility
  external
  Input parameters
    - address _newtoken
  Constraints
    - onlyAdmin modifier.
  Events emit
  Emits the `TokenChanged` event.
  Output
  bool
- *transferAnyERC20Token*
  Description
  Transfers accidentally locked tokens.
  Visibility
  public
  Input parameters
    - address tokenAddress
    - address beneficiary
    - uint tokens
  Constraints
    - onlyAdmin modifier.
    - A `tokenAddress` should not be the EMS token address.
  Events emit
  None
  Output
  bool

EmiVoting.sol

Description

EmiVoting is a contract used for voting.

EmiVault.sol

Description

EmiVault purpose is a contract used for storing tokens. Allows withdrawing tokens if a message is signed by the ORACLE address.

EmiFactory.sol

Description

EmiFactory is a factory used to deploy Emiswap pairs. Is a copy of the MooniFactory of the Mooniswap with some minor changes.

Detailed description is not required because the contract is a compy of another well known and audited contract.

Emiswap.sol

Description

Emiswap is a LP token. Is almost a copy of the Mooniswap with some minor changes.

Detailed description is not required because the contract is a copy of another well known and audited contract.

EmiReferral.sol

Description

EmiReferral is a contract used to store referrals.

Detailed description is not required because the contract is simple, and its functionality is corrupted.

EmiVamp.sol

Description

EmiVamp is used to convert liquidity from Mooniswap and Uniswap.

Imports

EmiVamp contract has the following imports:

- @openzeppelin/contracts/proxy/Initializable.sol
- ./uniswapv2/interfaces/IUniswapV2Pair.sol

- ./uniswapv2/interfaces/IUniswapV2Factory.sol
- ./libraries/Priviledgeable.sol
- @openzeppelin/contracts/token/ERC20/SafeERC20.sol
- ./interfaces/IEmiRouter.sol
- ./interfaces/IEmiswap.sol
- ./libraries/TransferHelper.sol

Inheritance

EmiVamp contract is Initializable, Priviledgeable.

Usages

EmiVamp contract has the following custom usages:

- SafeERC20 for IERC20

Structs

EmiVamp contract has the following data structures:

- LPTokenInfo

Enums

EmiVamp contract has no custom enums.

Events

EmiVamp contract has the following custom events:

- Deposit

Modifiers

EmiVamp has the no custom modifiers.

Fields

EmiVamp contract has following constants:

- IERC20 [] public allowedTokens
- LPTokenInfo [] public lpTokensInfo
- string public codeVersion = "EmiVamp v1.0-18-g44fc1eb"
- IEmiRouter public ourRouter

Functions

EmiVamp has following public functions:

- *initialize*
  Description
  Initializes the contract.
  Visibility
  public
  Input parameters
  - address[] calldata _lptokens
  - uint8[] calldata _types
  - address _ourrouter

  Constraints
  - Can only be called once.
  - Can only be called by an admin.

  Events emit
  None
  Output
  None
- *getAllowedTokensLength, lpTokensInfoLength*
  Description
  Simple getter functions.
- *addAllowedToken*
  Description
  Adds new entry to the list of allowed tokens
  Visibility
  external
  Input parameters
  - address _token

  Constraints
  - Can only be called by an admin.
  - A `_token` should not be zero.

  Events emit
  None
  Output
  None
- *addLPToken*
  Description
  Adds new entry to the list of convertible LP-tokens
  Visibility
  external
  Input parameters
  - address _token
  - uint16 _tokenType

  Constraints
  - Can only be called by an admin.
  - A `_token` should not be zero.
  - A `tokenType` should be less or equal to 2.

  Events emit

None
Output
None

- *changeRouter*
  Description
  Change emirouter address.
  Visibility
  external
  Input parameters
    - address _newEmiRouter

  Constraints
    - onlyAdmin modifier

  Events emit
  None
  Output
  None
- *deposit*
  Description
  Convert third-party liquidity.
  Visibility
  public
  Input parameters
    - uint256 _pid
    - uint256 _amount

  Constraints
    - '_pid' should exist.
    - Allowance should be set to at least `_amount`

  Events emit
  Emits the `Deposit` event.
  Output
  None
- *isPairAvailable*
  Description
  Checks an LP token pair availability.
  Visibility
  public view
  Input parameters
    - address _token0
    - address _token1

  Constraints
    - _token0 and _token1 should not be 0 addresses.

  Events emit
  None
  Output
  None
- *isPairAvailable*

Description
Checks an LP token pair availability.
Visibility
external
Input parameters
   ○ address _token0
   ○ address _token1
Constraints
   ○ _token0 and _token1 should not be 0 addresses.
Events emit
None
Output
unit16 - 0 if not available.

- *transferAnyERC20Token*
  Description
  Allows owners to transfer accidentally sent tokens.

EmiPrice.sol

Description

EmiPrice is used retrieve a token prices from 3 markets.

Imports

EmiPrice contract has the following imports:

- @openzeppelin/contracts/proxy/Initializable.sol
- @openzeppelin/contracts/math/SafeMath.sol
- ./uniswapv2/interfaces/IUniswapV2Factory.sol
- ./uniswapv2/interfaces/IUniswapV2Pair.sol
- ./libraries/Priviledgeable.sol

Inheritance

EmiPrice contract is Initializable, Priviledgeable.

Usages

EmiPrice contract has the following custom usages:

- SafeMath for uint
- SafeMath for uint256

Structs

EmiPrice contract has no custom data structures.

Enums

EmiPrice contract has no custom enums.

Events

EmiPrice contract has the following custom events.

Modifiers

EmiPrice has the no custom modifiers.

Fields

EmiPrice contract has following constants:

- address [3] public market
- address private _DAI
- string public codeVersion = "EmiPrice v1.0-18-g44fc1eb"

Functions

EmiPrice has following public functions:

- *initialize*
  Description
  Initializes the contract.
  Visibility
  public
  Input parameters
    - address _market1
    - address _market2
    - address _market3
    - address _daitoken
  Constraints
    - Can only be called once.
  Events emit
  None
  Output
  None
- *getCoinPrices*
  Description
  Returns coin prices * 10e5
  Visibility
  external view
  Input parameters
    - address [] calldata _coins
    - uint8 _market
  Constraints
    - A market should exists.

Events emit
None
Output
- uint[] memory prices

- *changeDAI*
  Description
  Changes the token.
  Visibility
  external
  Input parameters
    - address _daiToken
  Constraints
    - onlyAdmin modifier
    - A `_daiToken` should not be zero.
  Events emit
  None
  Output
  None

- *changeMarket*
  Description
  Changes a market address.
  Visibility
  external
  Input parameters
    - uint8 idx
    - address _market
  Constraints
    - onlyAdmin modifier
    - A `_market` should not be zero.
    - `idx` should be less than 3.
  Events emit
  None
  Output
  None

Timlock.sol

Description

Timelock is a copy of the SushiSwap Timelock contract.

Audit overview

■■■■ Critical

1. The `addReferral` function of the `EmiReferral` can be called by anyone. The data can be corrupted.

   We recommend restricting access to the `addReferral` function.

   Fixed before the second audit.

2. The `EmiVoting` contract is not actually designed for any kind of a voting process. The positive voting result is always set after an end period of a proposal.

   Fixed before the third review.

3. The `burn` function of the `ESW` allows burning tokens of any account without permissions.

   Fixed before the fourth review.

4. The `ESW` contract is not compliant with the `IESW` interface that is used in other contracts.

   Fixed before the fourth review.

5. The `EmiVoting` contract relies on the `getPriorVotes` function of the `ESW` token that is not implemented.

   Fixed before the fourth review.

■■■ High

1. The `presaleBulkLoad` function can be called an unlimited number of times. Tokens supply manipulation is possible.

   We recommend to disallow calling this function more than 1 time or to lock it forever when the load process is finished.

   Fixed before the third review. Final presale upload date added.

2. Tokens bought during a presale are `virtual`. Those tokens cannot be claimed from the vesting contract.

   Also, `virtual` balances are not described in the whitepaper and their purpose is unknown.

   We recommend describing this behavior in the whitepaper.

Partially fixed. Virtual tokens can now be claimed via the ‘mint‘ function of the ‘EmiVesting‘ contract.

Fixed before the third review. Virtual tokens functionality was removed.

3. Minters can be added to the ESW token unlimitedly. Token supply manipulation is possible.

   We recommend to allow minting only to those contracts that are specified in the whitepaper.

   Fixed before the third review.

4. The token total supply is not limited to 200,000,000 tokens as it is stated in the whitepaper.

   Fixed before the second audit.

5. The `changeToken` function of the `EmiVesting` contract changes a token address but does guarantee a valid balance of a new token.

   Fixed before the second audit.

6. The `presaleBulkLoad` function does not use values from `beneficiaries` input parameters.

   We recommend removing this parameter or to use its values.

   Fixed before the fourth review.

7. `freezePresale`, `freezeBulk`, `freezeVirtual` and `freezeVirtualWithCrowdsale` functions are not actually doing anything except validations and can be removed. Also, as soon as no locks could be added, the contract itself can be removed.

   Fixed before the fourth review.

8. The `switchMinter` function of the `ESW` contract sets the `minterChangeBlock` value to 35 blocks ahead that is approximately 490 seconds. Not 24h as its stated.

   Fixed before the fourth review.

9. The `mintSigned` function of the `ESW` may be used by owners to mint any number of tokens. The `oracle` address may only be a privately owned account that will have an ability to sign messages.

   Fixed before the fourth review.

10. The `_mint` function of the `ESW` is never used. Parent function is used instead.

    Fixed before the fourth review.

■■ Medium

1. The `freeze` function of the `EmiVesting` contract has invalid validation `tokens >= 0`.

   Fixed before the second audit.

2. The `EmiVault` contract is not finalized.

   Fixed before the third review.

3. Assigning of the admin in the `initialize` function of the `EmiVamp` contract is redundant because the function already has the `onlyAdmin` modifier and can only be called by a deployer.

   Fixed before the second audit.

4. The `_getBalance` function of the `EmiVesting`s can return both locked and total balances to reduce gas consumption.

   Fixed before the second audit.

5. Consider moving of the `referralInput` validation in the `_saveReferrals` function to the top of the function to reduce gas consumption in a case when the `referralInput` is 0.

   Fixed before the second audit.

6. The `freeze2` function of the `EmiVesting` has hardcoded values.

   Fixed before the second audit.

7. The `newUpgradeVoting` function of the `EmiVoting` contract has no validation of the `_hash` parameter. A vote with the same `_hash` can be passed into the function.

   Fixed before the second audit.

8. Signatures recover functions are copied in multiple contracts.

   We recommend moving this code to a library.

   Fixed before the forth audit.

Low

1. The `defRef` variable of the CrowdSale is never used.
2. The `deposit` and `swap` functions are too long. We recommend to decompose those functions to smaller ones.
3. The`dividendToken` field of the `ESW` contract is never used.
4. The `Initializable` inheritance in the `ESW` contract is redundant. The ProxiedERC20 is responsible for this functionality.
5. The presale end date is hardcoded in the `presaleBulkLoad` function of the `CrowdSale` contract.

   We recommend moving its value to a field and initialize it in the constructor.

9. The `_mintGranted` of the `ESW` contract is redundant and can be removed.

Lowest / Code style / Best Practice

1. Multiple code style issues found by the static code analyzer.
2. The EmiVesting contract is not following the solidity code style and naming guides.

   Fixed before the third audit.

3. The EmiVesting contract has functions with names like `_freeze` and `_freeze3`. It is a bad practice to name functions in that way.

   Fixed before the second audit.

4. `Swapped` and `Swapped2` events exists. We recommend merging them.

   Not an issue.

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. For the contract, high-level description of functionality was presented in As-Is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security engineers found **2** critical **6** high, **9** medium, **4** low and **4** informational issues during the audit.

After the second review, the code contains **1** critical, **3** high, **2** medium, **4** low, and **3** informational issues.

After the **third** review, the code contains **4** critical, **8** high, **2** medium and **5** low severity issues.

After the **fourth** review, the code contains **0** critical, **0** medium, and **6** low severity issues.

Notice: the overall low quality of custom contracts can lead to extra hidden errors.

Notice 2: additional reviews do not include a full audit of the provided code. As soon as Emi contracts reviewed 4 times, we may not guaranty their secureness at all.

Notice 3: tests are failing in the latest version of the code.

Violations in the following categories were found and addressed to Customer:

Category	Check Item	Comments
	▪ Asset’s integrity	▪ No resulting tokens is guaranteed when using the CrowdSale contract.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on blockchain platform. The platform, its programming language, and other software related to the smart contract can have its vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.