HashEx
BLOCKCHAIN SECURITY

Hera AggregatorV2

smart contracts
final audit report

October 2022

hashex.org

contact@hashex.org

Contents

1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2. Overview

HashEx was commissioned by the Hera Finance team to perform an audit of their smart contract. The audit was conducted between 20/09/2022 and 26/09/2022.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The code is available at @heraaggregator/contractv2 GitHub repository after the b30c3c9 commit. The HeraPowerCompiler contract was excluded from the scope of this audit.

The same contracts are deployed to Metis Andromeda:

HeraAggregatorV2	0x0000000009dC0FD7Ba31d83A5d328b9BCAa0Bc8d
HeraExecutor	0xd5723d7622EF97aA8D16fC3757b6D24D66FB41d6
HeraFeeSequencer	0x0000000FeE9CF96490006d1c833298c72AAff7A5

Update: the Hera Finance team has responded to this report. The updated code is located in the GitHub repository after the commit ced6b67. The contracts are deployed to the Metis Andromeda network.

HeraAggregatorV2	0x0000000000924fb1969e719edeD2feD54AFB183A

HeraExecutor	0x852d1fDd3982D8e21145845af74Db7ae37D1F383
HeraFeeSequencer	0x000000fEe322aAA0a5772e7F92DE10180f9fAB15

The HeraExecutor contract has been validated with its bytecode.

2.1 Summary

Project name	Hera AggregatorV2
URL	https://hera.finance
Platform	Metis
Language	Solidity

2.2 Contracts

Name	Address
HeraAggregatorV2	`0x0000000000924fb1969e719edeD2feD54AFB183A`
Queen	
HeraExecutor	`0x852d1fDd3982D8e21145845af74Db7ae37D1F383`
ExecutorManagement	
HeraERC20	
HeraFeeSequencer	`0x000000fEe322aAA0a5772e7F92DE10180f9fAB15`
RevertReasonParser	

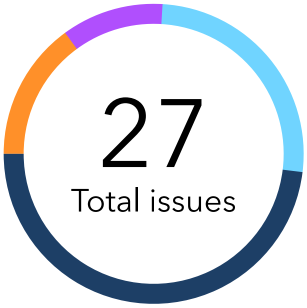
RevertReasonForwarder

StringUtil

/interfaces folder

HeraSecurity

3. Found issues



●	High	4 (15%)
●	Medium	3 (11%)
●	Low	7 (26%)
●	Info	13 (48%)

C1. HeraAggregatorV2

ID	Severity	Title	Status
C1-01	● High	Owner governed selfdestruct	✓ Resolved
C1-02	● Medium	Input amount is not validated	✓ Acknowledged
C1-03	● Low	Gas optimization	✓ Resolved
C1-04	● Info	No explicit visibility	✓ Resolved
C1-05	● Info	Lack of events	✓ Resolved

C2. Queen

ID	Severity	Title	Status
C2-01	● Low	fallback() duplicates functionality of receive()	✓ Resolved

C3. HeraExecutor

ID	Severity	Title	Status
C3-01	● Medium	Fee charging could be evaded	✓ Acknowledged
C3-02	● Medium	Swap validation	✓ Acknowledged
C3-03	● Info	No explicit visibility	✓ Resolved
C3-04	● Info	Swaps with 100% slippage	✓ Resolved
C3-05	● Info	Commented code	✓ Resolved

C4. ExecutorManagement

ID	Severity	Title	Status
C4-01	● High	Owner governed selfdestruct	✓ Resolved
C4-02	● High	Using changeFeeSequencer() function	✓ Resolved
C4-03	● Low	Unused parameters	✓ Resolved
C4-04	● Low	fallback() duplicates functionality of receive()	✓ Resolved
C4-05	● Low	Function visibility	✓ Resolved
C4-06	● Info	Commented code	✓ Resolved
C4-07	● Info	Typos	✓ Resolved
C4-08	● Info	Lack of events	✓ Resolved
C4-09	● Info	No explicit visibility	✓ Resolved

C6. HeraFeeSequencer

ID	Severity	Title	Status
C6-01	● High	Exaggerated owner's rights	✓ Resolved
C6-02	● Low	Division before multiplication	✓ Resolved
C6-03	● Low	Gas optimizations	✓ Resolved
C6-04	● Info	Authorization in view functions	✓ Resolved
C6-05	● Info	Typos	✓ Resolved
C6-06	● Info	Lack of events	✓ Resolved
C6-07	● Info	No explicit visibility	✓ Resolved

4. Contracts

C1. HeraAggregatorV2

Overview

Aggregator contract that can be used for aggregated calls to arbitrary addresses with limited signatures or to limited list of DEX routers for swaps.

It is expected that the contract will receive formatted input data from third-party applications. The audit team haven't checked third-party applications, warns about possible risks when using such applications.

Issues

C1-01 Owner governed selfdestruct ● High ✓ Resolved

The contract owner can destroy the contract using `destroy()` function. Moreover, he or she can further apply a contract with altered functionality to the same address (using a metamorphic pattern). This can lead to the fact that users, who work with the contract, can be deceived. In other words, the owners can change the implementation so that the contract only makes a profit (or benefit) only for themselves.

```
    function destroy() external onlyOwner{
        selfdestruct(payable(msg.sender));
    }
```

Recommendation

Consider removing the `destoy()` function.

C1-02 Input amount is not validated

● Medium ✓ Acknowledged

`Swap()` and `StableSwap()` functions don't validate input token amount for swaps when calling internal `_swap()` function. `detail.amountIn` amount of input tokens is transferred to executor contract but actual amount encoded in bytes data may be different, causing users to lose part of their funds.

```
    function Swap(IHeraExecutor executor, Detail calldata detail, bytes memory data)
public payable nonReentrant returns (uint256 returnAmount)  {
       return _swap( executor,  detail, data);
    }

    function StableSwap(IHeraExecutor executor, Detail calldata detail, bytes memory
data)  public payable nonReentrant returns (uint256 returnAmount)  {
       return _swap( executor,  detail, data);
    }

    function _swap(
        IHeraExecutor executor,
        Detail calldata detail,
        bytes memory data
    ) internal returns (uint256 returnAmount) {
        require(detail.amountIn > 0, "AmountIn cannot be zero");
        ...
        (detail.inputToken).HeraTransferFrom(
            msg.sender,
            address(executor),
            detail.amountIn,
            msg.value,
            USE_NATIVE
        );
        (bool success, bytes memory result) = executeCall(
            address(executor),
            msg.value,
            abi.encodeWithSelector(IHeraExecutor.execute.selector,msg.sender, data)
        );
        ...
    }
```

Recommendation

Add requirements for input data, and input amount in particular.

C1-03 Gas optimization ● Low ✓ Resolved

a. The variable `srcAccount` of the `Detail` structure is never used in contract code and can be removed.

b. The `setNative()` function can be declared as external to save gas.

C1-04 No explicit visibility ● Info ✓ Resolved

The `USE_NATIVE` variable has no explicit visibility set; default value `internal` is used.

C1-05 Lack of events ● Info ✓ Resolved

Governance function `setNative()` should emit corresponding event.

C2. Queen

Overview

Parent of HeraAggregatorV2 that implements EVM native currency receive functions.

Issues

C2-01 fallback() duplicates functionality of receive() ● Low ✓ Resolved

The `receive()` function is used only for messages with positive `msg.value` and zero-length `msg.data`, while `fallback()` function extends this functionality to non-zero `msg.data`. Removing the `fallback()` function may decrease the contract size.

C3. HeraExecutor

Overview

The contract is used for decoding aggregated input calls and forwarding them to the external executor addresses.

Issues

C3-01 Fee charging could be evaded ● Medium ✓ Acknowledged

Input parameters of the `execute()` function, i.e., `bytes datas`, could be constructed such as protocol fee is bypassed. For example, `amountOutMax` could be always set to `type(uint256).max` , denying any positive slippage in `ExecutorManagement.transferProtocol()`, or fee could be paid in arbitrary token `protocolData.tokenIn` that is not related to actual `tokenIn` in `callDatas[]`.

Recommendation

Reconsider the logic behind the fees and add proper documentation.

C3-02 Swap validation ● Medium ✓ Acknowledged

The `for()` loop of the `execute()` function cannot guarantee that the balance of the `tokenOut` token will increase after the execution. For example, in cases where there were no swaps or `calldatas[]` do not contain swaps with the receipt of output tokens.

Thus, the values of the variables `lastBalance` and `startBalance` may remain the same. This will lead to the fact that the execution of the function `HeraAggregatorV2._swap()` will fail on L74.

Moreover, if `callers[]` and `calldatas[]` contain a specific call to a contract with allowed signature that transfers a positive amount of `tokenOut` to the `HeraExecutor` contract, the transaction will be mined, and input tokens could be locked in the contract.

Recommendation

Make sure that the function `execute()` works as expected and consider adding validation for swaps of the `callDatas` values.

C3-03 No explicit visibility ● Info ✓ Resolved

The `_SAPPROVE`, `_SDEPOSITX`, `_SDEPOSIT`, and `_SWITHDRAW` variables have no explicit visibility set; default value `internal` is used.

C3-04 Swaps with 100% slippage ● Info ✓ Resolved

The `decodeBytes()` function calls DEX routers for swaps with `amountOutMin = 0`. Although this function is private and could be called only by HeraAggregatorV2, which is implementing the slippage check, we strictly recommend adding NatSpec descriptions with a warning about slippage.

C3-05 Commented code ● Info ✓ Resolved

There are commented code on L118. This may indicate the incompleteness of the functionality of the contract.

Update

The updated code still has a lot of commented lines of code.

C4. ExecutorManagement

Overview

Parent of HeraExecutor that implements interaction with fee calculator and governance functions.

Issues

C4-01 Owner governed selfdestruct ● High ✓ Resolved

The contract owner can destroy the contract using `destroy()` function. Moreover, he or she can further apply a contract with altered functionality to the same address (using a metamorphic pattern). This can lead to the fact that users, who work with the contract, can be deceived. In other words, the owners can change the implementation so that the contract only makes a profit (or benefit) only for themselves.

Recommendation

Consider removing the `destoy()` function.

C4-02 Using changeFeeSequencer() function ● High ✓ Resolved

The contract owner can change the `FEE_SEQUENCER` address. This can lead to fees being calculated at very different rates. For example, the owner of the contract is able to make a fee of 90% or even more.

Recommendation

It is necessary to restrict the rights of the owner to change the address of the `FEE_SEQUENCER`.

The ownership could be transferred to a Timelock-like governance contract with MultiSig admin and minimum delay of at least 24 hours.

Update

According to developer's comment the `changeFeeSequencer()` function will be managed by the Timelock contract.

Thus, users must keep track of the timelock contract, which will be able to change the address of contract FEE_SEQUENCER.

C4-03 Unused parameters ● Low ✓ Resolved

The parameters `tokenIn` and `amountFee` are never used in the `transferProtocol()` function and can be removed.

C4-04 fallback() duplicates functionality of receive() ● Low ✓ Resolved

`receive()` function is used only for messages with positive `msg.value` and zero-length `msg.data` , while `fallback()` function extends this functionality to non-zero `msg.data`. Removing the `fallback()` function may decrease the contract size.

C4-05 Function visibility ● Low ✓ Resolved

Consider changing the visibility of the `calculateAmountWithFee()` function to `internal` type to prevent unexpected calls.

C4-06 Commented code ● Info ✓ Resolved

There are commented code on L33-L37. This may indicate the incompleteness of the functionality of the contract.

Update

The updated code still has commented lines of code (L186-L188).

C4-07 Typos ● Info ✓ Resolved

Typos reduce code readability.

Typos in 'POSSITIVE', 'possitive', 'Slipapge'.

C4-08 Lack of events ● Info ✓ Resolved

Governance functions `changeAggregatorAddress()`, `changeFeeSequencer()`, `changeProtocolAddress()`, `changePossitiveAddress()`, `changeRouterType()` should emit corresponding event.

C4-09 No explicit visibility ● Info ✓ Resolved

The `PROTOCOL_ADDRESS`, `POSSITIVE_ADDRESS`, `AGGREGATOR_ADDRESS`, `FEE_SEQUENCER`, and `RouterTypes[]` variables have no explicit visibility set; default value `internal` is used.

C5. HeraERC20

Overview

Library for handling native transfers alongside with ERC-20 tokens. No issues were found.

C6. HeraFeeSequencer

Overview

Fee calculator contract implementing piecewise functions for arbitrary ERC-20 tokens and for owner-defined list of stable tokens.

Issues

C6-01 Exaggerated owner's rights ● High ✓ Resolved

The owner can break the contract's interaction by setting maliciously wrong parameters:

1. `DIVIDER` could be set lower than rate or even 0, breaking the main `getAmountWithFee()` function.
2. Rates boundaries could be inversed `STABLE_MAX_FEE_RATE < STABLE_MIN_FEE_RATE` and `MAX_FEE_RATE < MIN_FEE_RATE`, resulting in permanent fail of `getAmountWithFee()` function.

3. `LEVELS_COUNT` and `RATE_LEVELS[]` could be set incongruous, resulting in failed `getAmountWithFee()`.
4. Changing the `POWER_CONTRACT` address to a maliciously wrong one, it's possible to halt `getAmountWithFee()` requests.

```
function changePowerContract(IHeraPowerCompiler powerAddr) public onlyOwner {
    POWER_CONTRACT = powerAddr;
}

function changeDivider(uint256 rate) public onlyOwner {
    DIVIDER = rate;
}

function changeMinFeeRate(uint256 rate) public onlyOwner {
    if (rate >= PROTOCOL_MIN_FEE_RATE && rate <= PROTOCOL_MAX_FEE_RATE) {
        MIN_FEE_RATE = rate;
    } else {
        revert("Out of Fee Rate");
    }
}

function changeMaxFeeRate(uint256 rate) public onlyOwner {
    if (rate >= PROTOCOL_MIN_FEE_RATE && rate <= PROTOCOL_MAX_FEE_RATE) {
        MAX_FEE_RATE = rate;
    } else {
        revert("Out of Fee Rate");
    }
}

function checkStandart(address account) internal view returns (uint256) {
    if (DYNAMIC_REDUCER) {
        uint256 level = getLevel(account);
        return
            MIN_FEE_RATE.add(
                MAX_FEE_RATE.sub(MIN_FEE_RATE).div(LEVELS_COUNT.sub(1)).mul(
                    LEVELS_COUNT.sub(1).sub(level)
                )
            );
    } else {
        return AMM_FEE_RATE;
    }
}

function changeLevelRate(uint256 level, Level memory newdata) public onlyOwner {
    RATE_LEVELS[level] = newdata;
}

function changeLevelsCount(uint256 level) public onlyOwner {
    LEVELS_COUNT = level;
}

function getLevel(address account) internal view returns (uint256 level) {
    uint256 power = POWER_CONTRACT.getUserPower(account);
    for (uint i = 0; i < LEVELS_COUNT; i++) {
        Level memory rate = RATE_LEVELS[i];
        if (power >= rate.min && power < rate.max) {
            level = i;
            break;
        }
    }
}
```

Recommendation

Add safety checks to governance functions.

`DIVIDER` variable should be declared as constant, `changeDivider()` functions should be removed.

Invariant of minimum fee rate to be lower than maximum rate must be ensured.

Level system should be reworked to eliminate possible contradictions and ambiguities.

Transfer ownership of the contract to a Timelock-like contract with minimum delay of at least 24 hours.

C6-02 Division before multiplication ● Low ✓ Resolved

Division before multiplication can cause loss of precision of calculations. In this contract, division before multiplication is performed in `checkStandart()` and `checkStable()` functions.

C6-03 Gas optimizations ● Low ✓ Resolved

a. The variable `LEVELS_COUNT` is read multiple times in `checkStandart()`, `checkStable()`, and `getLevel()` functions. The variable `blessedTokens.length` is read multiple times in `checkBlessed()`, `getBlessedTokenIndex()`, `getBlessedTokens()`, and `removeBlessedToken()`.

The variable `STABLE_TOKENS_LIST.length` is read multiple times in `getStableTokens()` and `_removeStableToken()`. The variable `WHITELIST_ACCOUNTS_LIST.length` is read multiple times in `_removeWhitelistAccount()`.

b. Removing an item from array could be improved by moving the last element to the removing index and the perform a pop action. Affected functions: removeBlessedToken(), _removeStableToken(), and _removeWhitelistAccount().

c. getBlessedTokens() and getLevelRates() functions should be removed, tracking of the values on back-end side should be performed with events.

d. The blessedTokens[] array should be transformed into mapping keccak256(tokenIn, tokenOut) => bool, making checkBlessed() much more efficient.

e. Arrays WHITELIST_ACCOUNTS_LIST[] and STABLE_TOKENS_LIST[] could be removed, tracking of the values on back-end side should be performed with events. Another possibility to reduce needed gas is using Enumerables instead of arrays.

f. The state variables PROTOCOL_MIN_FEE_RATE and PROTOCOL_MAX_FEE_RATE can be declared as constant to save gas.

Update

The functions getBlessedTokens(), addBlessedPair(), removeBlessedPair(), getLevelRates(), addLevelRate(), changeLevelRate(), removeLevelRate(), changeMinFeeRate(), changeMaxFeeRate(), changeAmmFeeRate(), changeStableMinFeeRate(), changeStableMaxFeeRate(), changeStableFeeRate(), changePowerContract(), changeFeeStatus(), changeDynamicReducer(), changeDynamicStableReducer(), changeBlessedDayStatus(), getStableTokens(), removeStableToken(), addWhitelistAccount() , removeWhitelistAccount() can be declared as external to save gas.

C6-04 Authorization in view functions ● Info ✓ Resolved

Storage could be accessed for reading regardless the authorization requirements. No need to restrict view functions to onlyOwner, authorization should be moved to mutative functions. Affected functions: getBlessedTokenIndex(), getBlessedTokens(), getLevelRates(), getStableTokens(), and getWhitelistAccounts().

C6-05 Typos ● Info ✓ Resolved

Typos reduce code readability.

Typo in 'Standart'.

C6-06 Lack of events ● Info ✓ Resolved

Governance functions `changeLevelRate()`, `changeLevelsCount()`, `changeMinFeeRate()`, `changeMaxFeeRate()`, `changeAmmFeeRate()`, `changeStableMinFeeRate()`, `changeStableMaxFeeRate()`, `changeStableFeeRate()`, `changeDivider()`, `changePowerContract()`, `changeFeeStatus()`, `changeDynamicReducer()`, `changeDynamicStableReducer()`, `changeBlessedDayStatus()`, `addStableToken()`, `removeStableToken()`, `addWhitelistAccount()`, and `removeWhitelistAccount()` should emit corresponding events.

C6-07 No explicit visibility ● Info ✓ Resolved

The `STABLE_TOKENS[]`, `STABLE_TOKENS_LIST[]`, `blessedTokens[]`, `WHITELIST_ACCOUNTS[]`, and `WHITELIST_ACCOUNTS_LIST[]` variables have no explicit visibility set; default value `internal` is used.

Update

In the updated code the `PROTOCOL_MIN_FEE_RATE`, `PROTOCOL_MAX_FEE_RATE` variables have no explicit visibility set; default value `internal` is used.

C7. RevertReasonParser

Overview

Simple bytes-to-string parser for returned `bytes` data. No issues were found.

C8. RevertReasonForwarder

Overview

A simple library for reverting the latest call with returned data as error message. No issues were found.

C9. StringUtil

Overview

Library for bytes-to-hex conversion. No issues were found.

C10. /interfaces folder

Overview

Interfaces for Hera contracts and external router calls. No issues were found.

C11. HeraSecurity

Overview

This contract was introduced with the code update. It's a fork of TimelockController from OpenZeppelin library.

No issues were found.

5. Conclusion

4 high, 3 medium, 7 low severity issues were found during the audit. 4 high, 7 low issues were resolved in the update.

The contracts are highly dependent on third-party applications and the owner's account. Users using the project have to trust the project owner and that the third-party applications work properly.

Ownership of the HeraExecutor contract has been transferred to the TimeLock contract so that users can track important changes.

We strongly suggest adding documentation as well as unit and functional tests for all contracts.

This audit includes recommendations on improving the code and preventing potential attacks.

Note: no third-party application has been audited by the audit team.

Appendix A. Issues' severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Informational.** Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

Appendix B. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

✉ contact@hashex.org

@hashex_manager

blog.hashex.org

linkedin

github

twitter

HashEx
BLOCKCHAIN SECURITY