SECURITY AUDIT OF

HOLDSTATION DEFUTURES

Public Report

May 31, 2023

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.
BSC	Binance Smart Chain or BSC is an innovative solution for introducing interoperability and programmability on Binance Chain.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on May 31, 2023. We would like to thank the Holdstation for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Holdstation Defutures. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team found some vulnerabilities in the application.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Holdstation Defutures

Holdstation Defutures is a decentralized leveraged trading platform that combines liquidity efficiency, robustness, and user-friendliness.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the smart contracts of Holdstation Defutures.

It was conducted on commit `6fb1151dd05395a94a00bd73fb8c6095d5cd055d` from git repository link: *https://gitlab.com/hspublic/contract-holdstation-dex*.

The latest version of the following files were made available in the course of the review:

SHA256 Sum	File
78c0ded5891fd044ac7c53dd32316eb8da45fc91340d0b93d139550f8f424b23	contracts/functions/HSNftRewards.sol
c6d32507166aa24f800ddd0432edd82d799a3bd1b6b58763d368f8444d159c40	contracts/functions/HSPairInfos.sol
d013545314f99078490f089015322e02383a4e4c1e97909c050baf4471f00ae4	contracts/functions/HSPairStorage.sol
d23ac771a844782dd10fc70a68ab4c481588da1b03077582e972ecd5d30cc2fe	contracts/functions/HSPriceAggregatorV1.sol
6f4536cc38f97e87451b9dc69c97d88e729a197a4164e9cb39f4cdb301a25a17	contracts/functions/HSReferrals.sol
3a29877284b6b9e0f4c71109422f0abf473cfb5b34d07b655bc9c1d8df2dfb5c	contracts/functions/HSTokenOpenPnlFeed.sol
0dc50a330ecd7b31e465353bc7d782c5c1fd33a699af5ecb7f696ee00c101b35	contracts/functions/HSTrading.sol
f5577c55fc3b6c731bfadc628356ba93c5e446ce2c448962b6b41ce74ddb3cba	contracts/functions/HSTradingCallbacks.sol
8973ddc9661d7c16cfb132ed0ccd5e8a9741ffd3e41490588acb16dd38f4fd25	contracts/functions/HSTradingStorage.sol
89f7d6a211f2d59dfe85b85eef0e1e6875828113fec40d4f0441573102af86ef	contracts/functions/HSTradingVault.sol

b4aa32b25e29c3d28e630b862e28bf8c79e885ef4086c5409649e79fe331fc19	contracts/helpers/AccessControlMixin.sol
548a4430699a2c015097adcb16423309f9e87af456115cb3088158606e42b05c	contracts/helpers/ArrayAddress.sol
fc4604be74f240425c435f61beae58ad72530a03203dede1e62ace909c928351	contracts/helpers/ArrayBytes.sol
6acfad9cb390bc07d8dbe8dc545112b49e7f78b72e5e0c050c3c9f8088189f4a	contracts/helpers/ArrayUint256.sol
857ecc0d151854d9196f35e1ed490008e078031f9e2fedc99d1f55370d070ec8	contracts/helpers/ContextMixin.sol
006890c3cff6a2c6456c513112ddaa203c3ab7d77c9b60dd7474bcc5eb0b5d99	contracts/helpers/DateTime.sol
876f941c372788ec4b7b3ab1a1bab8eb72a9137d34f4dbd8a8d1942ad6186874	contracts/helpers/Delegatable.sol
bdb4eeef9c23d84a77692bc0aa9c91baf77939b75b027053441bb48d4954fd3f	contracts/helpers/EIP712Base.sol
0a97445383212227aa7cfc314f81f3578701484c5129eb2f41d31713d30e6fc0	contracts/helpers/FullMath.sol
43989ba8c8a58335bf9d9ff21ae44b3cb35248400d85063555277e2b5daa91b9	contracts/helpers/Initializable.sol
b6ebf3c8c2cbaebef01f872ced91259e1d4383bb4768e96292bd4598b4aba774	contracts/helpers/NativeMetaTransaction.sol
0323a2b07ce9562e160d331ca809d674cb6ea9205ee086cf11334268db15b9d3	contracts/proxies/ProxyAdminPattern.sol
6a5ae260a3bf8a81f4ab3ab0f762c20bf9bb3cc9167f997dd3414fafcdf691d0	contracts/proxies/ProxyPattern.sol
7bf6b36510cd9fcf4af63ac15fb83fbd72607b3a3df6edad6017172debc664fa	contracts/tokens/HSNft.sol
9e83cd23c52ede58b7a34f9889f4b2c0341f580335954d798b63ab92f0e34bf2	contracts/tokens/HSToken.sol
b4d06eb56b53574b2385fbf8c617e349c66326788e383c96d4164d83b0b1c324	contracts/tokens/HSTokenCredit.sol

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The Holdstation Defutures was written in `Solidity` language. The source code was written based on OpenZeppelin's library.

2.1.1. HSTradingVault contract

`HSTradingVault` contract is a `USDC vault` following `ERC-4626`, a standard API for tokenized yield-bearing vaults that represent shares of a single underlying `ERC-20` asset.

The `USDC vault` is used in every trade that happens on the platform. If a user wins a trade, they get paid by the vault in `USDC`. If they lose a trade, they pay the vault in `USDC`. This way, the platform creates incentives for liquidity providers based on the trading volume. The higher the volume, the higher the annual percentage rate (APR) for them.

2.1.2. HSTrading contract

This contract serves as the platform for decentralized trading by users. Regardless of the trading pair, trades are initiated with `USDC` collateral. The leverage is synthetic and supported by the `USDC vault`. When a trader earns a positive PNL, `USDC` is withdrawn from the `USDC vault` to compensate them. Conversely, negative PNL trades require the trader to pay `USDC` to the `USDC vault`.

To determine the median price for each trading order, the trading contract utilizes a custom decentralized oracle network. If any nodes have been set by the `gov` address, the contract fetches the price from those nodes. Otherwise, it falls back to using the `Chainlink Price Feed`.

The governance entity has the ability to pause the opening of new trades through a function, typically used during contract upgrades. However, this function does not close any open positions held by traders, and traders retain full control over closing their own trades. Additionally, governance has another function to prevent any interaction with the contract.

It should be noted that this contract relies on a third-party contract called `whitelist` in the `checkWhitelist` modifier. However, the code and details of this `whitelist` contract are not provided and are outside the scope of the current contract code audit.

```
modifier checkWhitelist() {
    require(address(whitelist) == address(0) || whitelist.isWhitelists(msg.sender),
"NOT_IN_WHITELIST");
    _;
}
```

This modifier can limit who can call `openTrade` and `executeOrderByHSTBot`.

2.2. Findings

During the audit process, the audit team found some vulnerabilities in the given version of Holdstation Defutures.

Holdstation team fixed some issues, according to Verichains's draft report in commit `e10a4414170594fa337e6cf25a6a5f7650b0aaad`.

#	Issue	Severity	Status
1	Centralized risk with upgradable contracts	HIGH	Acknowledged
2	Not enough USDC to pay NFT reward fee	HIGH	Acknowledged
3	`gov` address can withdraw USDC from storage	HIGH	Fixed
4	Incompatible function parameter	HIGH	Fixed
5	Test functions must be removed	HIGH	Fixed
6	Wrong USDC decimal on Ethereum chain	HIGH	Fixed
7	Zero reward	HIGH	Acknowledged
8	High gas cost	MEDIUM	Acknowledged
9	High gas cost with larger number of pending orders	MEDIUM	Acknowledged
10	`checkIndex` logic was not implemented	MEDIUM	Acknowledged
11	`mint` does not `contributeToken`	MEDIUM	Acknowledged
12	Chainlink price feed has deviation threshold	LOW	Acknowledged

2.2.1. Centralized risk with upgradable contracts HIGH

Many contracts in Holdstation Defutures are upgradable, but there is no mechanism to protect upgrading (such as Multi-sig or DAO). The contract deployer has permission to upgrade the contract to modify any logic code, so there is a risk that the contract deployer's wallet could be compromised and the current contracts could be upgraded to withdraw users' money from the vault or other contracts.

UPDATES

- *May 26, 2023*: This issue has been acknowledged.

2.2.2. Not enough USDC to pay NFT reward fee HIGH

Affected files:

- HSTradingCallbacks.sol

In `unregisterTrade` function, when an order is closed by NFT bots, `HSTradingCallbacks` keeps `nftFeeUsdc` in storage for NFT bots to claim. If `usdcSentToTrader` <= `usdcLeftInStorage` and `usdcLeftInStorage` < `usdcSentToTrader + usdcKeepInStorage`, contract will send `usdcLeftInStorage - usdcSentToTrader` to vault and send `usdcSentToTrader` to trader so there may be some case that there is not enough `usdcKeepInStorage` to pay for NFT bots.

e.g:`usdcSentToTrader = 100`, `usdcLeftInStorage = 100` and `usdcKeepInStorage = 1`The contract will send to vault 100 - 100 = 0 USDC and send to trader 100 USDC, so there is no `usdcKeepInStorage` left for NFT bots to claim later.

```
if (usdcSentToTrader > usdcLeftInStorage) {
    storageT.vault().sendAssets(usdcSentToTrader - usdcLeftInStorage + usdcKeepInStorage,
trade.trader);
    storageT.transferUsdc(address(storageT), trade.trader, usdcLeftInStorage -
usdcKeepInStorage);
} else {
    sendToVault(
        usdcLeftInStorage > usdcSentToTrader + usdcKeepInStorage
            ? usdcLeftInStorage - usdcSentToTrader - usdcKeepInStorage
            : usdcLeftInStorage - usdcSentToTrader,
        trade.trader
    );
    storageT.transferUsdc(address(storageT), trade.trader, usdcSentToTrader);
}
```

UPDATES

- *May 30, 2023*: This issue has been acknowledged.

2.2.3. `gov` address can withdraw USDC from storage HIGH

Affected files:

- HSTradingStorage.sol

When initializing the `HSTradingStorage` contract, `gov` is set to `msg.sender` (the contract deployer), who can then set any addresses to be the `tradingContract` and update the `token` to any contract. With control of both the `tradingContract` and the `token`, they can use `transferUsdc` to withdraw `USDC` from storage. This leaves a centralized risk that the owner of the project can withdraw users' money.

```
function initialize(
    TokenInterfaceV5 _usdc,
    TokenInterfaceV5 _token,
    TokenInterfaceV5 _linkErc677,
    NftInterfaceV5[5] memory _nfts
) external initializer {
    ...
    token = _token;
    gov = msg.sender;
    ...
}

modifier onlyGov() {
    require(msg.sender == gov);
    _;
}

function addTradingContracts(address[] memory _tradings) external onlyGov {
    for (uint i = 0; i < _tradings.length; i++) {
        _addTradingContract(_tradings[i]);
    }
}

function _addTradingContract(address _trading) private {
    require(token.hasRole(MINTER_ROLE, _trading), "NOT_MINTER");
    require(_trading != address(0));
    isTradingContract[_trading] = true;
    emit TradingContractAdded(_trading);
}

function updateToken(TokenInterfaceV5 _newToken) external onlyGov {
    require(trading.isPaused() && callbacks.isPaused(), "NOT_PAUSED");
    require(address(_newToken) != address(0));
    token = _newToken;
    emit AddressUpdated("token", address(_newToken));
}

modifier onlyTrading() {
    require(isTradingContract[msg.sender] && token.hasRole(MINTER_ROLE, msg.sender));
    _;
}

function transferUsdc(address _from, address _to, uint256 _amount) external onlyTrading {
    if (_from == address(this)) {
        usdc.transfer(_to, _amount);
    } else {
        usdc.transferFrom(_from, _to, _amount);
    }
}
```

UPDATES

- *May 26, 2023*: The client has removed `updateToken` function to limit `gov` permission. To use `transferUsdc` function now, the caller must have `MINTER_ROLE` in `token` contract and was registered as `TradingContract` by `gov`.

2.2.4. Incompatible function parameter HIGH

Affected files:

- HSTrading.sol
- HSTradingStorage.sol

There is an incompatible in struct `OpenLimitOrder` between `HSTrading` and `HSTradingStorage`. In `openTrade` function in `HSTrading`, it calls `storeOpenLimitOrder` function in `HSTradingStorage` contract but the passed in parameter `OpenLimitOrder` (from `StorageInterfaceV5`) is incompatible with `OpenLimitOrder` in `HSTradingStorage` so this function call will be reverted. There are some other places which are using wrong `OpenLimitOrder` struct too.

```
function openTrade(
    StorageInterfaceV5.Trade memory t,
    NftRewardsInterfaceV6.OpenLimitOrderType orderType, // LEGACY => market
    uint256 spreadReductionId,
    uint256 slippageP, // for market orders only
    address referrer
) external notContract notDone checkWhitelist {
    ...
    if (orderType != NftRewardsInterfaceV6.OpenLimitOrderType.LEGACY) {
        uint256 index = storageT.firstEmptyOpenLimitIndex(sender, t.pairIndex);

        storageT.storeOpenLimitOrder(
        StorageInterfaceV5.OpenLimitOrder(
        StorageInterfaceV5.OrderInfo(
        t.pairIndex,
        t.positionSizeUsdc,
        t.buy,
        t.leverage,
        t.tp,
        t.sl,
        t.openPrice,
        t.openPrice
        ),
        sender,
        index,
        spreadReductionId > 0 ? storageT.spreadReductionsP(spreadReductionId - 1) : 0,
        block.number
        )
        );
```

```
        nftRewards.setOpenLimitOrderType(sender, t.pairIndex, index, orderType);

        emit OpenLimitPlaced(sender, t.pairIndex, index);
    }
    ...
}

// StorageInterfaceV5.sol
    struct OrderInfo {
        uint256 pairIndex;
        uint256 positionSize;
        bool buy;
        uint256 leverage;
        uint256 tp;
        uint256 sl;
        uint256 minPrice;
        uint256 maxPrice;
    }
    struct OpenLimitOrder {
        OrderInfo orderInfo;
        address trader;
        uint256 index;
        uint256 spreadReductionP;
        uint256 block;
    }

// HSTradingStorage.sol
    struct OpenLimitOrder {
        address trader;
        uint256 pairIndex;
        uint256 index;
        uint256 positionSize; // 1e18 (USDC or GFARM2)
        uint256 spreadReductionP;
        bool buy;
        uint256 leverage;
        uint256 tp; // PRECISION (%)
        uint256 sl; // PRECISION (%)
        uint256 minPrice; // PRECISION
        uint256 maxPrice; // PRECISION
        uint256 block;
        uint256 tokenId; // index in supportedTokens
    }

    function storeOpenLimitOrder(OpenLimitOrder memory o) external onlyTrading {
        o.index = firstEmptyOpenLimitIndex(o.trader, o.pairIndex);
        o.block = block.number;
        openLimitOrders.push(o);
        openLimitOrderIds[o.trader][o.pairIndex][o.index] = openLimitOrders.length - 1;
```

```
        openLimitOrdersCount[o.trader][o.pairIndex]++;
    }
```

RECOMMENDATION

Fix `StorageInterfaceV5` to reflect the right struct for `HSTradingStorage` and fix the function call in `openTrade` function.

UPDATES

- *May 24, 2023*: This issue has been acknowledged and fixed.

2.2.5. Test functions must be removed HIGH

Affected files:

- HSTrading.sol
- HSPriceAggregatorV1.sol

There are many places that are using `aggregator.emptyNodeFulFill` to test without oracle nodes in testnet. This test function must be removed before deploying in mainnet or `gov` can control `fakeFeedPrice` to manipulate users' positions through fake price.

```
// HSTrading.sol
function openTrade(
    StorageInterfaceV5.Trade memory t,
    NftRewardsInterfaceV6.OpenLimitOrderType orderType, // LEGACY => market
    uint256 spreadReductionId,
    uint256 slippageP, // for market orders only
    address referrer
) external notContract notDone checkWhitelist {
    ...
    storageT.storePendingMarketOrder(
        StorageInterfaceV5.PendingMarketOrder(
            StorageInterfaceV5.Trade(sender, t.pairIndex, 0, 0, t.positionSizeUsdc, 0,
t.buy, t.leverage, t.tp, t.sl),
            0,
            t.openPrice,
            slippageP,
            spreadReductionId > 0 ? storageT.spreadReductionsP(spreadReductionId - 1) : 0,
            0
        ),
        orderId,
        true
    );

    aggregator.emptyNodeFulFill(t.pairIndex, orderId,
AggregatorInterfaceV6.OrderType.MARKET_OPEN); // test function must be removed.
```

```
    emit MarketOrderInitiated(orderId, sender, t.pairIndex, true);
    ...
}

// HSPriceAggregatorV1.sol
function emptyNodeFulFill(uint256 pairIndex, uint256 orderId, OrderType orderType) external
onlyTrading {
    if (nodes.length != 0) {
        return;
    }
    PairsStorageInterfaceV6.Feed memory f = pairsStorage.pairFeed(pairIndex);

    uint256 feedPrice;
    if (fakeFeedPrice == 0) {
        (, int256 feedPrice1, , , ) = ChainlinkFeedInterfaceV5(f.feed1).latestRoundData();
        if (f.feedCalculation == PairsStorageInterfaceV6.FeedCalculation.DEFAULT) {
            feedPrice = uint256((feedPrice1 * int256(PRECISION)) / 1e8);
        } else if (f.feedCalculation == PairsStorageInterfaceV6.FeedCalculation.INVERT) {
            feedPrice = uint256((int256(PRECISION) * 1e8) / feedPrice1);
        } else {
            (, int256 feedPrice2, , , ) =
ChainlinkFeedInterfaceV5(f.feed2).latestRoundData();
            feedPrice = uint256((feedPrice1 * int256(PRECISION)) / feedPrice2);
        }
    } else feedPrice = fakeFeedPrice;

    CallbacksInterfaceV6_2.AggregatorAnswer memory a;

    a.orderId = orderId;
    a.price = feedPrice;
    a.spreadP = pairsStorage.pairSpreadP(pairIndex);

    CallbacksInterfaceV6_2 c = CallbacksInterfaceV6_2(storageT.callbacks());

    if (orderType == OrderType.MARKET_OPEN) {
        c.openTradeMarketCallback(a);
    } else if (orderType == OrderType.MARKET_CLOSE) {
        c.closeTradeMarketCallback(a);
    } else if (orderType == OrderType.LIMIT_OPEN) {
        c.executeNftOpenOrderCallback(a);
    } else if (orderType == OrderType.LIMIT_CLOSE) {
        c.executeNftCloseOrderCallback(a);
    } else {
        c.updateSlCallback(a);
    }

    emit PriceReceived(bytes32(block.timestamp), orderId, msg.sender, pairIndex, feedPrice,
feedPrice, 0);
}
```

RECOMMENDATION

Remove all the test functions and parameters.

UPDATES

- *May 23, 2023*: This issue has been acknowledged and fixed.

2.2.6. Wrong USDC decimal on Ethereum chain HIGH

Affected files:

- HSReferrals.sol

The functions `getReferrerFeeP` and `getPercentOfOpenFeeP_calc` in the `HSReferrals` contract calculate fees based on the volume of `USDC` tokens. However, these calculations assume that `USDC` tokens have 18 decimal places. In reality, `USDC` tokens on the Ethereum blockchain and some chains use 6 decimal places (while other use 18 decimal). Consequently, the calculations performed on Ethereum using 6 decimal places for `USDC` will yield incorrect results.

```
function registerTrade(
    StorageInterfaceV5.Trade memory trade,
    uint256 nftId,
    uint256 limitIndex
) private returns (StorageInterfaceV5.Trade memory, uint256) {
    ...
    v.reward1 = referrals.distributePotentialReward(
        trade.trader,
        v.levPosUsdc, // 1e6
        pairsStored.pairOpenFeeP(trade.pairIndex),
        v.tokenPriceUsdc
    );
    ...
}
function distributePotentialReward(
    address trader,
    uint256 volumeUsdc, // 1e6
    uint256 pairOpenFeeP,
    uint256
) external onlyCallbacks returns (uint256) {
    ...
    uint256 referrerRewardValueUsdc = (volumeUsdc * getReferrerFeeP(pairOpenFeeP,
r.volumeReferredUsdc)) /
    PRECISION /
    100;
    ...
}
function getReferrerFeeP(uint256 pairOpenFeeP, uint256 volumeReferredUsdc) public view
returns (uint256) {
```

```
    uint256 maxReferrerFeeP = (pairOpenFeeP * 2 * openFeeP) / 100;
    uint256 minFeeP = (maxReferrerFeeP * startReferrerFeeP) / 100;

    uint256 feeP = minFeeP + ((maxReferrerFeeP - minFeeP) * volumeReferredUsdc) / 1e18 /
targetVolumeUsdc; // volumeReferredUsdc is 1e6

    return feeP > maxReferrerFeeP ? maxReferrerFeeP : feeP;
}

function getPercentOfOpenFeeP_calc(uint256 volumeReferredUsdc) public view returns (uint256
resultP) {
    resultP =
    (openFeeP *
        (startReferrerFeeP *
        PRECISION +
            (volumeReferredUsdc * PRECISION * (100 - startReferrerFeeP)) / //
volumeReferredUsdc is 1e6
            1e18 /
            targetVolumeUsdc)) /
    100;

    resultP = resultP > openFeeP * PRECISION ? openFeeP * PRECISION : resultP;
}
```

RECOMMENDATION

Change calculation to 1e6.

```
function getReferrerFeeP(uint256 pairOpenFeeP, uint256 volumeReferredUsdc) public view
returns (uint256) {
    uint256 maxReferrerFeeP = (pairOpenFeeP * 2 * openFeeP) / 100;
    uint256 minFeeP = (maxReferrerFeeP * startReferrerFeeP) / 100;

    uint256 feeP = minFeeP + ((maxReferrerFeeP - minFeeP) * volumeReferredUsdc) / 1e6 /
targetVolumeUsdc;

    return feeP > maxReferrerFeeP ? maxReferrerFeeP : feeP;
}

function getPercentOfOpenFeeP_calc(uint256 volumeReferredUsdc) public view returns (uint256
resultP) {
    resultP =
    (openFeeP *
        (startReferrerFeeP *
        PRECISION +
            (volumeReferredUsdc * PRECISION * (100 - startReferrerFeeP)) /
            1e6 /
            targetVolumeUsdc)) /
    100;
```

```
    resultP = resultP > openFeeP * PRECISION ? openFeeP * PRECISION : resultP;
}
```

UPDATES

- *May 23, 2023*: The client said that they will update `targetVolumeUsdc` to the decimals of `volumeReferredUsdc` and `getPercentOfOpenFeeP_calc` is not currently being used, so they will update if needed.

2.2.7. Zero reward HIGH

Affected files:

- HSTokenCredit.sol

In `_distributeReward` function, `currentRewardValue` will be added by an amount of `delta time` `PRECISION` / `totalStake`. *If* `totalStake` *(with decimals) is higher than* `delta time` `PRECISION`, the next added rewards will be round down to zero.

For example: if `currentTimeStamp - currentReward.blockTime` is `12` (seconds) and `totalStake` is `200000 * 1e6` so `12 * 1e10 / (200000 * 1e6)` will be 0.6 and round down to 0.

```
currentRewardValue = int256(previousReward) + int256(((currentTimeStamp -
currentReward.blockTime) * PRECISION) / totalStake);
uint256 public constant PRECISION = 1e10;

function _distributeReward(uint256 _amount, uint256 currentTimeStamp, bool out) private {
    if (totalCreditSupplied >= totalCreditSupply) {
        return;
    }

    uint256 arrayLength = contributionRequests.length;

    if (arrayLength == 0 && !out) {
        contributionRequests.push(ContributionRequest(currentTimeStamp, 0));
        totalStake += _amount;
        return;
    }

    ContributionRequest storage currentReward = contributionRequests[arrayLength - 1];
    uint256 previousReward = currentReward.rewardDistribution;

    if (currentReward.blockTime == currentTimeStamp) {
        int256 currentRewardValue = 0;
        if (arrayLength != 1 && totalStake != 0) {
            if (!out) {
                currentRewardValue =
                int256(previousReward) +
```

```
                int256((((currentTimeStamp - contributionRequests[arrayLength -
2].blockTime) * PRECISION) / totalStake));
            } else {
                currentRewardValue =
                int256(previousReward) -
                int256((((currentTimeStamp - contributionRequests[arrayLength -
2].blockTime) * PRECISION) / totalStake));
            }
        }
        currentReward.rewardDistribution = currentRewardValue < 0 ? 0 :
uint256(currentRewardValue);
    } else {
        int256 currentRewardValue = 0;
        if (totalStake != 0) {
            if (!out) {
                currentRewardValue =
                int256(previousReward) +
                int256(((currentTimeStamp - currentReward.blockTime) * PRECISION) /
totalStake);
            } else {
                currentRewardValue =
                int256(previousReward) -
                int256(((currentTimeStamp - currentReward.blockTime) * PRECISION) /
totalStake);
            }
        }
        ContributionRequest memory newReward = ContributionRequest(
            currentTimeStamp,
            currentRewardValue < 0 ? 0 : uint256(currentRewardValue)
        );
        contributionRequests.push(newReward);
    }
    if (!out) {
        totalStake += _amount;
    } else {
        totalStake = totalStake >= _amount ? totalStake - _amount : 0;
    }
}
```

UPDATES

- *May 26, 2023*: This issue has been acknowledged.

2.2.8. High gas cost MEDIUM

Affected files:

- HSTokenCredit.sol

In `HSTokenCredit` contract, there are many places that loop over an increasing array. Looping through a large array in solidity cost a lot of gas. So it must be avoided, instead, use mapping whenever possible.

For example, in `contributeToken`, the contract loop through `userContributed` array with `indexOf` to only insert user if not existed. This will cost more and more gas for every user contributed. Instead, we only need to check that `userCurrentContributed[_userAddress] == 0`.

```
function contributeToken(uint256 _amount, address _userAddress) external nonReentrant
onlyVault {
    require(_amount > 0, "ZERO_AMOUNT");
    uint256 currentTimeStamp = block.timestamp;

    userRequests[_userAddress].push(UserRequest(currentTimeStamp, _amount));
    _distributeReward(_amount, currentTimeStamp, false);
    if (userContributed.length == 0 || userContributed.indexOf(_userAddress) < 0) {
        userContributed.push(_userAddress);
    }
    userCurrentContributed[_userAddress] += _amount;
    emit Contributed(_userAddress, _amount, currentTimeStamp);
}
```

In `forceNewEpoch`, `countUserContributed` array is looped 2 times to `getUserReward` which is duplicated `getUserReward` and cost more gas. Instead, we only need to calculate for 1 time, store it and use it to calculate `userEarn` instead of recalculating again.

```
function forceNewEpoch() external nonReentrant onlyOpenPnl {
    ...
    uint256 tmpRewards = 0;
    for (uint256 i = 0; i < countUserContributed; i++) {
        tmpRewards += getUserReward(userContributed[i]); // store result
    }

    for (uint256 i = 0; i < countUserContributed; i++) {
        users[i] = userContributed[i];
        uint256 tmpReward = getUserReward(userContributed[i]); // use stored result instead
of recalculating getUserReward
        uint256 userRewardValue = uint256((totalRewardEachEpoch * tmpReward) / tmpRewards);
        rewards[i] = userRewardValue;
        userEarn[userContributed[i]] += userRewardValue;
    }
    ...
}
```

The function also uses 2 loop to reset staked data. Overall, this function will cost a huge amount of gas.

RECOMMENDATION

Avoid looping through large arrays whenever possible.

UPDATES

- *May 23, 2023*: This issue has been acknowledged.

2.2.9. High gas cost with larger number of pending orders MEDIUM

Affected files:

- HSTokenCredit.sol

The contract is using an array to store pending orders and the pending order array is looped to remove pending order while using `currentPendingOrderIds.indexOf(_id)`. The more the number of pending orders are, the more gas will cost for running this function.

```
function unregisterPendingMarketOrder(uint256 _id, bool _open) external onlyTrading {
    PendingMarketOrder memory _order = reqID_pendingMarketOrder[_id];
    uint256[] storage orderIds = pendingOrderIds[_order.trade.trader];

    for (uint256 i = 0; i < orderIds.length; i++) {
        if (orderIds[i] == _id) {
            if (_open) {
                pendingMarketOpenCount[_order.trade.trader][_order.trade.pairIndex]--;
            } else {
                pendingMarketCloseCount[_order.trade.trader][_order.trade.pairIndex]--;

openTradesInfo[_order.trade.trader][_order.trade.pairIndex][_order.trade.index].beingMarket
Closed = false;
            }

            orderIds[i] = orderIds[orderIds.length - 1];
            orderIds.pop();

            delete reqID_pendingMarketOrder[_id];
            if (currentPendingOrderIds.length > 0) {
                int256 index = currentPendingOrderIds.indexOf(_id); // High gas cost with
larger number of pending orders.
                if (index >= 0) {
                    currentPendingOrderIds.remove(uint256(index));
                }
            }
            return;
        }
    }
}
```

RECOMMENDATION

Use a mapping to store index of `_id` in `currentPendingOrderIds` array and look it up instead of using `indexOf`.

UPDATES

- *May 23, 2023*: This issue has been acknowledged.

2.2.10. `checkIndex` logic was not implemented MEDIUM

Affected files:

- HSTradingVault.sol

Modifier `checks` receiver `checkIndex` parameter but does not implement any logic with it.

```
modifier checks(uint256 assetsOrShares, bool checkIndex) {
    require(shareToAssetsPrice > 0, "PRICE_0");
    require(assetsOrShares > 0, "VALUE_0");
    _;
}
```

RECOMMENDATION

Implement `checkIndex` or remove it if it is redundant.

UPDATES

- *May 26, 2023*: This issue has been acknowledged.

2.2.11. `mint` does not `contributeToken` MEDIUM

Affected files:

- HSTradingVault.sol

Both `mint` and `deposit` functions are using for deposit `asset` and receive `shares` but while `deposit` does `contributeToken`, `mint` function does not.

```
function deposit(uint256 assets, address receiver) public override checks(assets, false)
returns (uint256) {
    require(assets <= maxDeposit(receiver), "ERC4626: deposit more than max");

    uint256 shares = previewDeposit(assets);
    scaleVariables(shares, assets, true);

    _deposit(_msgSender(), receiver, assets, shares);
    if (address(tokenCredit) != address(0)) {
        tokenCredit.contributeToken(assets, receiver);
    }
    return shares;
```

```
}

function mint(uint256 shares, address receiver) public override checks(shares, false)
returns (uint256) {
    require(shares <= maxMint(receiver), "ERC4626: mint more than max");

    uint256 assets = previewMint(shares);
    scaleVariables(shares, assets, true);

    _deposit(_msgSender(), receiver, assets, shares);
    return assets;
}
```

RECOMMENDATION

Add `contributeToken` logic to `mint` function.

UPDATES

- *May 23, 2023*: This issue has been acknowledged.

2.2.12. Chainlink price feed has deviation threshold LOW

Affected files:

- HSPriceAggregatorV1.sol

Please note that if the contract gets the price only with Chainlink price feed using `emptyNodeFulFill` (when `nodes.length == 0`), each pair has its own deviation threshold which only triggers price update when the price changed more than the threshold. The result is that the Chainlink price could be different from real time price for that pair's deviation threshold percentage (from 0.5% to 2%).

UPDATES

- *May 23, 2023*: This issue has been acknowledged.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*May 11, 2023*	Draft Report	Verichains Lab
1.1	*May 26, 2023*	Private Report	Verichains Lab
1.2	*May 30, 2023*	Public Report	Verichains Lab
1.3	*May 31, 2023*	Public Report	Verichains Lab

Table 2. Report versions history