# Starbase



| | |
|---|---|
| **Date** | August 2024 |
| **Auditors** | Sergii Kravchenko, Vladislav Yaroshuk |

# 1 Executive Summary

This report presents the results of our engagement with **Starbase** to review their smart contract system.

The review was conducted over two weeks, from **Jul 22, 2024** to **Aug 23, 2024**, by **Sergii Kravchenko** and **Vladislav Yaroshuk**. A total of 40 person-days were spent. After that the fix review was conducted over 1 week, from **Oct 28, 2024** to **Nov 1, 2024**, by **Vladislav Yaroshuk**. A total of 5 person-days were spent. The second round of the fix review was necessary afterwards, it was conducted over 2 weeks, from **Nov 25, 2024** to **Dec 5, 2024**, by **Sergii Kravchenko**. A total of 10 person-days were spent.

During the initial engagement we have identified multiple findings, which allowed malicious actors to steal funds from the protocol, as well as some of the execution flow hasn't been working. The Starbase team has addressed these findings, redesigning how `allowance` is granted by removing the `Permit2` library and restricting order fulfillment to whitelisted takers only. During the second fix review, the team substantially increased the test suite and improved the quality of the codebase by adding comments and documentation. Additionally, they have incorporated necessary validations within the `starbase-contract` repository, particularly in the `AggregatedSwapRouter` contract, while maintaining flexibility in its functionality.

The review was done under the assumption that specific bots will be used to fill orders. These bots are in the scope of this review: `StarBaseLimitOrderBot` and `StarBaseDCABot`. The team plans to enable other third-party bots in the future. We recommend performing security audits for these bots before using them, as they can potentially break the system's logic.

# 2 Scope

Our review focused on the smart contracts provided by the team. The list and the version of files in scope can be found in the Appendix. After the initial engagement the Starbase team has decided to reduce the scope.

For the first fix review the team has provided 2 commits for both codebases, which included already all of the fixes for the findings: `d81b6f90d52b12dcfd6f05f023b19ca6e9a8c9e2` for `starbase-contract` repository and `2b508ff772206751317e8b0c6f5f70d4987a2b5e` for `starbase-limitorder` repository.

For the second and final fix review the following commit hashes were settled: `d323fe3cc9c939518cd631d63a8952bf4465ba16` for `starbase-contract` repository and `28a1471010e82968445e92c7a1fd1def7c3de380` for `starbase-limitorder` repository.

## 2.1 Objectives

Together with the **Starbase** team, we identified the following priorities for our review:

1. Correctness of the implementation, consistent with the intended functionality and without unintended edge cases.
2. Identify known vulnerabilities particular to smart contract systems, as outlined in our Smart Contract Best Practices, and the Smart Contract Weakness Classification Registry.

# 3 System Overview

The system comprises two primary components: "**Limit Order**" and "**Swaps**".

## 3.1 Limit orders

The contracts can handle two types of orders: `StarBaseDCA` and `StarBaseLimitOrder`. Both types are signed off-chain by the makers and await fulfillment by any participant. These orders can also be canceled by their creators. After the fix review the fulfillment of the order can be done only by the whitelisted addresses.

- **Limit orders** - Simple orders with fixed price, amount, and expiration time.
- **DCA (Dollar-Cost Averaging) orders** - Orders that are executed in multiple tranches over time. A portion of the trade is executed every `cycleSecondsApart` until the entire order is fulfilled. These orders include a minimum and maximum price range for the swaps.

Makers of both order types approve their tokens using the `Permit2` mechanism, however after audit the team has decided to remove `Permit2` library and use direct approves instead.

Orders can be filled either directly by takers or through a bot mechanism (`StarBaseDCABot`, `StarBaseLimitOrderBot`). These bots can only be triggered by a restricted set of users listed in the `isAdminListed` function by the owner of the bot contracts. The bots attempt to execute swaps of the maker's tokens via a separate swapping component (`StarBaseRouteProxy`).

## 3.2 Swaps

The second component consists of multiple contracts designed to perform token swaps across various exchanges.

The main swap routing logic resides in the `SwapByteIn` contract, which is inherited by `MyDefiSwapCore`. This contract routes swaps to different exchanges based on the input parameters, including platforms like Uniswap V2, V3, and their forks. These contracts have been removed from scope after fix review.

The primary entry point for swaps is the `AggregatedSwapRouter`. It interacts with `MyDefiSwapCore` to execute swaps and verifies the amount of output tokens, as well as validates the receiver address.

# 4 Security Specification

This section describes, **from a security perspective**, the expected behavior of the system under audit. It is not a substitute for documentation. The purpose of this section is to identify specific security properties that were validated by the audit team.

## 4.1 Actors

The relevant actors are listed below with their respective abilities:

- **Users**:
  - **Makers**: Can sign limit and DCA (Dollar-Cost Averaging) orders, then wait for others to fulfill them. Makers also have the ability to cancel their orders.
  - **Takers**: Can execute pending signed limit orders by filling them.
- **Admins**: Have the authority to initiate limit order/DCA bots to swap the maker's tokens and fulfill the orders.
- **Owners**: Have the power to modify key contract parameters, such as fees and the recipient of those fees, the list of admins, the approved proxy address, and more.

## 4.2 Trust Model

In any system, it's important to identify what trust is expected/required between various actors. For this audit, we established the following trust model:

- The system is designed to be trustless.
- Takers should fill the order returning full amount received after swap to the maker, not returning minimal amount to the maker and keeping the difference.
- The system is expected to be initialized correctly.
- Users are approving & interacting with the `AggregatedSwapRouter` if the want to execute a swap.

# 5 Findings

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

## 5.1 Malicious Taker Can Use Permit From Different Trades for Gains Critical ✓ Fixed

**Resolution**

In the `2b508ff772206751317e8b0c6f5f70d4987a2b5e` commit provided for the fix review the issue has been fixed by fully removing `PERMIT2` library and thus removing `permitSingle.details.token` data from the order, leaving only `inputToken` address, prohibiting the taker to choose which token to use.

### Description

In the `fillLimitOrder` function of the `StarBaseLimitOrder` contract and in the `fillDCA` function of the `StarBaseDCA` contract, there is no validation between the `inputToken` or `makerToken` in the order and the token in the `permitSingle` variable. This lack of validation puts all users with two or more active orders at risk.

Consider the following scenario:

1. The maker has created, signed, and signed permits for two limit orders in `StarBaseLimitOrder`:
   - The first order exchanges `1000` USDT tokens with `6` decimals into `0.39` ETH tokens.
   - The second order exchanges `10` WBTC tokens with `8` decimals into `228` ETH tokens.

   In both cases, the `makerAmount` is `10**9`.
2. A malicious taker begins filling the first order with the USDT tokens by calling the `fillLimitOrder` function but uses `permitSingle` data from the second order with the WBTC order.
3. The taker passes `takerFillAmount` as `0.39 * 10**18` to fill the order fully. The `StarBaseLimitOrder` contract validates that the order is not filled, the `orderHash` has been signed by the maker, and the order is not expired. However, the `permitSingle` is not part of the original order, so it is not validated.
4. The `curTakerFillAmount` and the `fee` variables are calculated and then validated. `curTakerFillAmount` is equal to `0.39 * 10**18`.
5. In the `claimTokens` call to the `_StarBase_APPROVE_PROXY_` contract, `curMakerFillAmount` is passed, along with the addresses of the maker and taker, and the `permit` data itself, missing all validations of the `permit` data.
6. In the `claimTokens` function, the `permit` is executed, granting allowance for WBTC and then transferring `10` WBTC tokens to the taker.

7. The taker returns `0.39` ETH tokens to the maker, completing the order and keeping the leftover WBTC, resulting in a gain of `227.61` ETH. The second order cannot be fully filled anymore.

## Examples

**starbase-limitorder/src/StarBaseLimitOrder.sol:L64-L99**

```
function fillLimitOrder(
    Order calldata order,
    bytes memory signature,
    uint160 takerFillAmount,
    uint160 thresholdTakerAmount,
    bytes memory takerInteraction,
    IAllowanceTransfer.PermitSingle calldata permitSingle,
    bytes calldata permitSignature
) public nonReentrant returns(uint160 curTakerFillAmount, uint160 curMakerFillAmount) {
    bytes32 orderHash = _orderHash(order);
    uint160 filledTakerAmount = _FILLED_TAKER_AMOUNT_[orderHash];

    require(filledTakerAmount < order.takerAmount, "SLOP: ALREADY_FILLED");

    if (_isContract(order.maker)) {
        _verifyERC1271WalletSignature(order.maker, orderHash, signature);
    } else {
        require(ECDSA.recover(orderHash, signature) == order.maker, "SLOP:INVALID_SIGNATURE");
    }
    require(order.expiration > block.timestamp, "SLOP: EXPIRE_ORDER");


    uint160 leftTakerAmount = order.takerAmount - filledTakerAmount;
    curTakerFillAmount = takerFillAmount < leftTakerAmount ? takerFillAmount:leftTakerAmount;
    curMakerFillAmount = curTakerFillAmount * order.makerAmount / order.takerAmount;
    uint160 fee = curTakerFillAmount * _FEE_RATE_ / 10000;

    require(curTakerFillAmount > 0 && curMakerFillAmount > 0, "SLOP: ZERO_FILL_INVALID");
    require(curTakerFillAmount >= thresholdTakerAmount, "SLOP: FILL_AMOUNT_NOT_ENOUGH");

    _FILLED_TAKER_AMOUNT_[orderHash] = filledTakerAmount + curTakerFillAmount;

    //Maker => Taker
    IStarBaseApproveProxy(_StarBase_APPROVE_PROXY_).claimTokens(
        order.maker, msg.sender, curMakerFillAmount,
        permitSingle,permitSignature);
```

**starbase-limitorder/src/StarBaseDCA.sol:L77-L123**

```
function fillDCA(
    Order memory order,
    bytes memory signature,
    bytes memory takerInteraction,
    IAllowanceTransfer.PermitSingle calldata permitSingle,
    bytes calldata permitSignature
) public nonReentrant returns (uint256 curTakerFillAmount) {
    bytes32 orderHash = _orderHash(order);
    DCAStates storage DCAFilledTimes = _DCA_FILLEDTIMES_[orderHash];

    require(DCAFilledTimes.numberOfTrade < order.numberOfTrade, "DCAP: ALREADY_FILLED");
    require( block.timestamp - DCAFilledTimes.lastUpdateTime >= order.cycleSecondsApart, "DCAP: TIME_NOT_ENOUGH");


    if (_isContract(order.maker)) {
        _verifyERC1271WalletSignature(order.maker, orderHash, signature);
    } else {
        require(ECDSA.recover(orderHash, signature) == order.maker, "DCAP:INVALID_SIGNATURE");
    }
    require(order.expiration > block.timestamp, "DCAP: EXPIRE_ORDER");


    uint160 fee = (order.inAmount * _FEE_RATE_) / 10000;

    require(IERC20(order.inputToken).balanceOf(order.maker) > fee + order.inAmount,"DCAP: INFINCIENT_BALANCE");


    DCAFilledTimes.lastUpdateTime = block.timestamp;
    DCAFilledTimes.numberOfTrade = DCAFilledTimes.numberOfTrade + 1;

    //Maker => Taker
    IStarBaseApproveProxy(_StarBase_APPROVE_PROXY_).claimTokens(
        order.maker,
        msg.sender,
        order.inAmount,
        permitSingle,
        permitSignature
    );

    //Maker => Fee
    IStarBaseApproveProxy(_StarBase_APPROVE_PROXY_).claimTokens(
        order.maker,
        _FEE_RECEIVER_,
        fee,
        permitSingle,
        permitSignature
    );
```

**starbase-limitorder/src/StarBaseApprove.sol:L81-L95**

```
function claimTokens(
    address who,
    address dest,
    uint160 amount,
    IAllowanceTransfer.PermitSingle calldata permitSingle,
    bytes calldata signature
)  external {
    require(msg.sender == _StarBase_PROXY_, "StarBaseApprove:Access restricted");
    if (amount > 0) {
        //IERC20(token).safeTransferFrom(who, dest, amount);
        // Transfer tokens from the caller to ourselves.
        require(permitSingle.spender == address(this),"PERMIT_DENY");
        PERMIT2.permit(who, permitSingle, signature);
        PERMIT2.transferFrom(who, dest, amount, permitSingle.details.token);
    }
```

**starbase-limitorder/src/StarBaseApproveProxy.sol:L68-L83**

```
function claimTokens(
    address who,
    address dest,
    uint160 amount,
    IAllowanceTransfer.PermitSingle calldata permitSingle,
    bytes calldata signature
) external {
    require(_IS_ALLOWED_PROXY_[msg.sender], "StarBaseApproveProxy:Access restricted");
    IStarBaseApprove(_StarBase_APPROVE_).claimTokens(
        who,
        dest,
        amount,
        permitSingle,
        signature
    );
}
```

### Recommendation

We recommend adding validation to ensure that the token and amount in the `permit` data correlate with every specific order.

## 5.2 Anyone Can Steal All Funds From a DCA Order Critical ✓ Fixed

### Resolution

In the `2b508ff772206751317e8b0c6f5f70d4987a2b5e` commit provided for the fix review the issue has been fixed by adding validation in the end of the if block: `require(curTakerFillAmount > 0 && curTakerFillAmount >= order.minOutAmountPerCycle, "Invalid curTakerFillAmount");`, and also reverting in cases when the `takerInteraction` length is zero: `revert takerInteractionFail(takerInteraction);`.

### Description

Anyone can fill an existing DCA order by calling the `fillDCA` function of the `StarBaseDCA` contract:

**starbase-limitorder/src/StarBaseDCA.sol:L77-L83**

```
function fillDCA(
    Order memory order,
    bytes memory signature,
    bytes memory takerInteraction,
    IAllowanceTransfer.PermitSingle calldata permitSingle,
    bytes calldata permitSignature
) public nonReentrant returns (uint256 curTakerFillAmount) {
```

While executing this function, a callback to the sender occurs if the `takerInteraction` parameter is not empty:

**starbase-limitorder/src/StarBaseDCA.sol:L125-L142**

```
if (takerInteraction.length > 0) {
    takerInteraction.patchUint256(0, order.inAmount);
    takerInteraction.patchUint256(1, order.minOutAmountPerCycle);
    takerInteraction.patchUint256(2, order.maxOutAmountPerCycle);

    require(isWhiteListed[msg.sender], "DCAP: Not Whitelist Contract");

    (bool success, bytes memory data) = msg.sender.call(takerInteraction);
    if (!success) {
        assembly {
            revert(add(data, 32), mload(data))
        }
    }
    curTakerFillAmount = data.decodeUint256(0);
}

//Taker => Maker
IERC20(order.outputToken).safeTransferFrom(msg.sender, order.maker, curTakerFillAmount);
```

This parameter is submitted by the `msg.sender` and has no validation. If the `takerInteraction` is zero, the `curTakerFillAmount` will also remain zero. The taker (`msg.sender`) will transfer zero tokens to the maker at the end of the function:

**starbase-limitorder/src/StarBaseDCA.sol:L141-L142**

```
//Taker => Maker
IERC20(order.outputToken).safeTransferFrom(msg.sender, order.maker, curTakerFillAmount);
```

But will receive the `order.inAmount`, essentially stealing tokens from the order:

**starbase-limitorder/src/StarBaseDCA.sol:L107-L114**

```
//Maker => Taker
IStarBaseApproveProxy(_StarBase_APPROVE_PROXY_).claimTokens(
    order.maker,
    msg.sender,
    order.inAmount,
    permitSingle,
    permitSignature
);
```

### Recommendation

Ensure that `order.minOutAmountPerCycle` is checked in all circumstances.

## 5.3 Missing Validation of `_FEE_RATE_` Variable Major ✓ Fixed

### Resolution

In the `2b508ff772206751317e8b0c6f5f70d4987a2b5e` commit provided for the fix review the issue has been only partially fixed with the comment "solved `require(feeRate <= MAX_FEE_RATE, "Fee rate too high");`", but apparently this fix has been added only to the `changeFeeFeeRate` function of the `StarBaseDCA` contract, and haven't been added to other contracts functions and `constructor`. Also, the new variable `uint160 constant MAX_FEE_RATE = 5000; // 100%` has a comment `100` %, while in reality it's `50` %. We recommend adding validation to leftover functions and `constructor`'s, fixing the comment, and setting the max fee to `20` % at most.

**Update (commit hash `7415929c5d5d1958f131847242d853290b378597`):** The `_FEE_RATE_` is now limited to 10%. The protocol owner can instantly change the fees to a higher amount, and existing orders won't have time to cancel if they disagree. Since the system is designed to be trustless, it would be good to add a timelock mechanism for updating the fees. However, because the maximum fee limit is 10%, the impact on users is limited and everyone should be aware of that risk. It's the responsibility of the protocol to warn users beforehand about changing the fees.

### Description

In the `init` and `changeFeeReceiver` functions of the `StarBaseDCA` contract, as well as in the `init` and `changeFeeRate` functions of the `StarBaseLimitOrder` contract and the `changeFeeRate` function of the `StarBaseLimitOrderBot` contract, the `_FEE_RATE_` variable can be set to any `uint160` value, while the denominator is `10000`. This allows the fee rate to be set as high as 100%, which is problematic since a trustless system is expected. This could enable the owner to steal all of the tokens with every trade, leading to excessive fees being charged during transactions, as well as a full block of other function execution when the fee is higher than 100%.

### Examples

**starbase-limitorder/src/StarBaseDCA.sol:L69-L73**

```
function init(address owner, address StarBaseApproveProxy, address feeReciver, uint160 feeRate) external {
    initOwner(owner);
    _StarBase_APPROVE_PROXY_ = StarBaseApproveProxy;
    _FEE_RECEIVER_ = feeReciver;
    _FEE_RATE_ = feeRate;
```

**starbase-limitorder/src/StarBaseDCA.sol:L191-L194**

```
function changeFeeReceiver(uint160 feeRate) public onlyOwner {
    _FEE_RATE_ = feeRate;
    emit ChangeFeeRate(feeRate);
}
```

**starbase-limitorder/src/StarBaseLimitOrder.sol:L56-L60**

```
function init(address owner, address StarBaseApproveProxy, address feeReciver,uint160 feeRate) external {
    initOwner(owner);
    _StarBase_APPROVE_PROXY_ = StarBaseApproveProxy;
    _FEE_RECEIVER_ = feeReciver;
    _FEE_RATE_ = feeRate;
```

**starbase-limitorder/src/StarBaseLimitOrder.sol:L183-L185**

```
function changeFeeRate (uint160 feeRate) public onlyOwner {
    _FEE_RATE_ = feeRate;
    emit ChangeFeeRate(feeRate);
```

**starbase-limitorder/src/StarBaseLimitOrderBot.sol:L119-L122**

```
function changeFeeRate (uint160 feeRate) public onlyOwner {
    _FEE_RATE_ = feeRate;
    emit ChangeFeeRate(feeRate);
}
```

### Recommendation

We recommend adding validation checks for the `_FEE_RATE_` to ensure it is within an acceptable range, such as below 100%.

## 5.4 Incorrect `inAmount` Passed to the `Order` in `StarBaseDCA` Major ✓ Fixed

#### Resolution

In the `2b508ff7722067513l7e8b0c6f5f70d4987a2b5e` commit provided for the fix review the issue has been fixed with the comment: "The precondition is not aligned, `inAmount` Variable is not total principal.(Code comments are wrong). Missunderstand that there are problems with the business and realign the audit". Additionally the comment has been changed to `// One of the total (numberOfTrade)` instead of `// total principal`.

### Description

In the `StarBaseDCA` contract, there is an `Order` struct designed to be executed `numberOfTrade` times, with the total amount of principal in the order represented by the `inAmount` variable. The StarBase team has confirmed that `inAmount` represents the full amount of tokens.

However, in the `fillDCA` function, starting with the very first order execution in the `claimTokens` call, all of the `inAmount` tokens are transferred from the maker to the taker. The problem with this design is that any subsequent call to `fillDCA` for the order cannot be executed after the first call, as all of the allowance is used. This means the dollar cost averaging mechanism will not function as intended. Additionally, other variables in the struct are meant to be correct for one cycle, such as the `minOutAmountPerCycle` or `maxOutAmountPerCycle` variables. Executing the order for the full amount of principal while using the `minOutAmountPerCycle` variable for one cycle will allow frontrunning of the order, leading to the loss of tokens, or using the `maxOutAmountPerCycle` variable for one cycle will allow keeping all of the leftover tokens at the maker's address.

### Examples

**starbase-limitorder/src/StarBaseDCA.sol:L26-L37**

```
struct Order {
    uint16 cycleSecondsApart; // executed per minute
    uint16 numberOfTrade; // executed 5 times
    address inputToken; // sell
    address outputToken; // buy
    address maker;
    uint160 inAmount; // total principal
    uint256 minOutAmountPerCycle; //min out amount
    uint256 maxOutAmountPerCycle; //max out amount
    uint256 expiration;
    uint256 salt;
}
```

**starbase-limitorder/src/StarBaseDCA.sol:L108-L114**

```
IStarBaseApproveProxy(_StarBase_APPROVE_PROXY_).claimTokens(
    order.maker,
    msg.sender,
    order.inAmount,
    permitSingle,
    permitSignature
);
```

### Recommendation

We recommend executing the `claimTokens` function not for the full `inAmount` value, but only for the amount of tokens suitable for one cycle. Additionally, we recommend writing tests for the code.

## 5.5 Improper Use of `ArgumentsDecoder` Leads to Incorrect `curTakerFillAmount` Decoding

Major ✓ Fixed

#### Resolution

In the `2b508ff7722067513l7e8b0c6f5f70d4987a2b5e` commit provided for the fix review the issue has been fixed by using `abi.decode(data, (uint256));`.

### Description

In the `decodeUint256` function of the `ArgumentsDecoder` contract, the padding is set to `0x24` bytes, where the additional `0x04` bytes correspond to a `selector`. In the `fillDCA` function of the `StarBaseDCA` contract, the `decodeUint256` function is used to decode the `curTakerFillAmount` variable returned after the `takerInteraction` call. However, the `data` variable returned after the call does not

include a `selector`, so the `0x24` bytes padding is incorrect for decoding this data. This leads to a problem where `curTakerFillAmount` is decoded incorrectly, resulting in a much larger value than it should be.

```
contract Test {

    function decodeUint256(bytes memory data, uint256 argumentIndex) public pure returns(uint256 value) {
        assembly { // solhint-disable-line no-inline-assembly
            value := mload(add(add(data, 0x24), mul(argumentIndex, 0x20)))
        }
        // returned value is 47244640256
    }

}

contract Test1 {

    function foo() public pure returns (uint256) {
        return 11; // returned data is 0x000000000000000000000000000000000000000000000000000000000000000b
    }

}
```

Consider the following scenario using the current StarBase architecture:

1. A whitelisted admin calls the `fillStarBaseDCA` function of the `StarBaseDCABot` contract with the `callExternalData` call, where the `fillDCA` function is encoded.
2. The `fillDCA` function of the `StarBaseDCA` contract is executed with the `takerInteraction` call, where the `doDCASwap` function is encoded.
3. The `doDCASwap` function of the `StarBaseDCABot` contract is executed with the `StarBaseApiData` call, where the swap is performed. The contract then calculates the amount of tokens received after the swap and returns this amount as the `returnTakerAmount` variable, for example, `11`, so the variable returned in the low-level call is `0x000000000000000000000000000000000000000000000000000000000000000b`.
4. In the `fillDCA` function, the `returnTakerAmount` variable is decoded using the `decodeUint256` function. Due to incorrect padding, the variable is decoded as `0x000000000000000000000000000000000000000000000000000000b00000000`, so the `curTakerFillAmount` equals `47244640256`.
5. Tokens from the `taker` are returned to the `maker` in the `safeTransferFrom` call for the `curTakerFillAmount` value. Since the `allowance` from `StarBaseDCABot` to `StarBaseDCA` is infinite, the tokens from `StarBaseDCABot` will be stolen to the maker's address, if `StarBaseDCABot` had this amount of tokens on its balance. However, since `StarBaseDCABot` doesn't keep tokens on the balance, all function executions will revert, making this code unusable.

## Examples

**starbase-limitorder/src/lib/ArgumentsDecoder.sol:L20-L24**

```
function decodeUint256(bytes memory data, uint256 argumentIndex) internal pure returns(uint256 value) {
    assembly { // solhint-disable-line no-inline-assembly
        value := mload(add(add(data, 0x24), mul(argumentIndex, 0x20)))
    }
}
```

**starbase-limitorder/src/StarBaseDCABot.sol:L45-L59**

```
function fillStarBaseDCA(
   bytes memory callExternalData, //call StarBaseDCA
   address outputToken,
   uint256 minOutputTokenAmount
) external {
   require(isAdminListed[msg.sender], "ACCESS_DENIED");
   uint256 originTakerBalance = IERC20(outputToken).balanceOf(address(this));

   (bool success, bytes memory data) = _StarBase_DCA_.call(callExternalData);
   if (!success) {
       assembly {
           revert(add(data, 32), mload(data))
       }
   }
```

**starbase-limitorder/src/StarBaseDCABot.sol:L71-L102**

```
function doDCASwap(
    uint256 inAmount,
    uint256 minOutAmount,
    uint256 maxOutAmount,
    address inputToken, //fromToken
    address outputToken, //toToken
    address StarBaseRouteProxy,
    bytes memory StarBaseApiData
) external returns (uint256 returnTakerAmount){
    require(msg.sender == _StarBase_DCA_, "ACCESS_NENIED");
    uint256 originTakerBalance = IERC20(outputToken).balanceOf(address(this));

    _approveMax(IERC20(inputToken), _StarBase_APPROVE_, inAmount);

    (bool success, bytes memory data) = StarBaseRouteProxy.call(StarBaseApiData);
    if (!success) {
        assembly {
            revert(add(data, 32), mload(data))
        }
    }

    uint256 takerBalance = IERC20(outputToken).balanceOf(address(this));
    returnTakerAmount = takerBalance - originTakerBalance;

    require(returnTakerAmount >= minOutAmount, "SWAP_TAKER_AMOUNT_NOT_ENOUGH");
    if(returnTakerAmount > maxOutAmount){
        returnTakerAmount = maxOutAmount;
    }


    _approveMax(IERC20(outputToken), _StarBase_DCA_, returnTakerAmount);
}
```

**starbase-limitorder/src/StarBaseDCA.sol:L77-L142**

```
function fillDCA(
    Order memory order,
    bytes memory signature,
    bytes memory takerInteraction,
    IAllowanceTransfer.PermitSingle calldata permitSingle,
    bytes calldata permitSignature
) public nonReentrant returns (uint256 curTakerFillAmount) {
    bytes32 orderHash = _orderHash(order);
    DCAStates storage DCAFilledTimes = _DCA_FILLEDTIMES_[orderHash];

    require(DCAFilledTimes.numberOfTrade < order.numberOfTrade, "DCAP: ALREADY_FILLED");
    require( block.timestamp - DCAFilledTimes.lastUpdateTime >= order.cycleSecondsApart, "DCAP: TIME_NOT_ENOUGH");


    if (_isContract(order.maker)) {
        _verifyERC1271WalletSignature(order.maker, orderHash, signature);
    } else {
        require(ECDSA.recover(orderHash, signature) == order.maker, "DCAP:INVALID_SIGNATURE");
    }
    require(order.expiration > block.timestamp, "DCAP: EXPIRE_ORDER");


    uint160 fee = (order.inAmount * _FEE_RATE_) / 10000;

    require(IERC20(order.inputToken).balanceOf(order.maker) > fee + order.inAmount,"DCAP: INFINCIENT_BALANCE");


    DCAFilledTimes.lastUpdateTime = block.timestamp;
    DCAFilledTimes.numberOfTrade = DCAFilledTimes.numberOfTrade + 1;

    //Maker => Taker
    IStarBaseApproveProxy(_StarBase_APPROVE_PROXY_).claimTokens(
        order.maker,
        msg.sender,
        order.inAmount,
        permitSingle,
        permitSignature
    );

    //Maker => Fee
    IStarBaseApproveProxy(_StarBase_APPROVE_PROXY_).claimTokens(
        order.maker,
        _FEE_RECEIVER_,
        fee,
        permitSingle,
        permitSignature
    );

    if (takerInteraction.length > 0) {
        takerInteraction.patchUint256(0, order.inAmount);
        takerInteraction.patchUint256(1, order.minOutAmountPerCycle);
        takerInteraction.patchUint256(2, order.maxOutAmountPerCycle);

        require(isWhiteListed[msg.sender], "DCAP: Not Whitelist Contract");

        (bool success, bytes memory data) = msg.sender.call(takerInteraction);
        if (!success) {
            assembly {
                revert(add(data, 32), mload(data))
            }
        }
        curTakerFillAmount = data.decodeUint256(0);
    }

    //Taker => Maker
    IERC20(order.outputToken).safeTransferFrom(msg.sender, order.maker, curTakerFillAmount);
```

### Recommendation

We recommend reviewing and correcting the padding logic in the `decodeUint256` function or creating a separate function that decodes the variable without padding:

```
function decodeUint256(bytes memory data, uint256 argumentIndex) public pure returns(uint256 value) {
    assembly { // solhint-disable-line no-inline-assembly
        value := mload(add(add(data, 0x20), mul(argumentIndex, 0x20)))
    }
}
```

Alternatively, consider using the `abi.decode` operation. We also strongly recommend adding tests to the code before deployment.

## 5.6 Incorrect Integration of `Permit2` in `StarBaseDCA`, `StarBaseLimitOrder` Major ✓ Fixed

| Resolution |
| --- |
| In the `2b508ff7722067513l7e8b0c6f5f70d4987a2b5e` commit provided for the fix review the issue has been fixed by fully removing `PERMIT2` library and using `transferFrom` calls instead. |

### Description

In the `claimTokens` function of the `StarBaseDCA` contract, the same `permitSignature` and `permitSingle` variables are called twice in the `IStarBaseApproveProxy(_StarBase_APPROVE_PROXY_).claimTokens` call. However, this approach does not work because the nonce changes after

the first use of the permit, invalidating the second `permit` and leading to the revert of the function, making this function unusable. Specifically, in the `claimTokens` function, there is a call to the `permit` function of the `AllowanceTransfer` contract with the `PermitSingle` struct as an input, where in the `_updateApproval` call there is a check:

```
if (allowed.nonce != nonce) revert InvalidNonce();
```

This check will pass during the first execution of the `permit` call; however, in the `updateAll` call, the `storedNonce` variable is incremented, making the second `permit` call invalid, resulting in the `InvalidNonce` error.

The same problem exists in the `StarBaseLimitOrder` contract. The `fillLimitOrder` function is supposed to be executed multiple times with the same `permitSingle`, as the taker can specify `takerFillAmount` and fill the order partially. With the next execution of this order, the `fillLimitOrder` function will revert with another `permit` call and the same `nonce`.

## Examples

**starbase-limitorder/src/StarBaseDCA.sol:L108-L123**

```
IStarBaseApproveProxy(_StarBase_APPROVE_PROXY_).claimTokens(
    order.maker,
    msg.sender,
    order.inAmount,
    permitSingle,
    permitSignature
);

//Maker => Fee
IStarBaseApproveProxy(_StarBase_APPROVE_PROXY_).claimTokens(
    order.maker,
    _FEE_RECEIVER_,
    fee,
    permitSingle,
    permitSignature
);
```

**starbase-limitorder/src/StarBaseLimitOrder.sol:L96-L100**

```
//Maker => Taker
IStarBaseApproveProxy(_StarBase_APPROVE_PROXY_).claimTokens(
    order.maker, msg.sender, curMakerFillAmount,
    permitSingle,permitSignature);
```

**starbase-limitorder/lib/permit2/src/AllowanceTransfer.sol:L33-L40**

```
function permit(address owner, PermitSingle memory permitSingle, bytes calldata signature) external {
    if (block.timestamp > permitSingle.sigDeadline) revert SignatureExpired(permitSingle.sigDeadline);

    // Verify the signer address from the signature.
    signature.verify(_hashTypedData(permitSingle.hash()), owner);

    _updateApproval(permitSingle.details, owner, permitSingle.spender);
}
```

**starbase-limitorder/lib/permit2/src/AllowanceTransfer.sol:L128-L143**

```
    /// @notice Sets the new values for amount, expiration, and nonce.
    /// @dev Will check that the signed nonce is equal to the current nonce and then incrememnt the nonce value by 1.
    /// @dev Emits a Permit event.
    function _updateApproval(PermitDetails memory details, address owner, address spender) private {
        uint48 nonce = details.nonce;
        address token = details.token;
        uint160 amount = details.amount;
        uint48 expiration = details.expiration;
        PackedAllowance storage allowed = allowance[owner][token][spender];

        if (allowed.nonce != nonce) revert InvalidNonce();

        allowed.updateAll(amount, expiration, nonce);
        emit Permit(owner, token, spender, amount, expiration, nonce);
    }
}
```

**starbase-limitorder/lib/permit2/src/libraries/Allowance.sol:L10-L30**

```
/// @notice Sets the allowed amount, expiry, and nonce of the spender's permissions on owner's token.
/// @dev Nonce is incremented.
/// @dev If the inputted expiration is 0, the stored expiration is set to block.timestamp
function updateAll(
    IAllowanceTransfer.PackedAllowance storage allowed,
    uint160 amount,
    uint48 expiration,
    uint48 nonce
) internal {
    uint48 storedNonce;
    unchecked {
        storedNonce = nonce + 1;
    }

    uint48 storedExpiration = expiration == BLOCK_TIMESTAMP_EXPIRATION ? uint48(block.timestamp) : expiration;

    uint256 word = pack(amount, storedExpiration, storedNonce);
    assembly {
        sstore(allowed.slot, word)
    }
}
```

### Recommendation

We recommend reviewing the logic of using separate `claimTokens` calls for both `msg.sender` and `_FEE_RECEIVER_` addresses, or using the `permit` function with the `PermitBatch` input struct. Additionally, we recommend adding an `if` statement to check when the `allowance` has already been granted, eliminating the need to execute `permit` again. Finally, we strongly recommend adding test cases to ensure the code works as expected.

## 5.7 DCA Economic Model Failure Medium Partially Addressed

#### Resolution

In the original version of the code anyone could fill DCA orders. Now only protocol admins can fill orders via the DCA bot. The attack can only be performed by the admins but they are supposed to be non-malicious. It reduces the severity of the issue to the trust issue. The malicious behavior of the admins is easily noticeable and would heavily damage the reputation of the protocol. Given this and the fact that, only a portion of order's value can be obtained risk-free, the incentives for this attack are not very good for the protocol owners to rug the users. But if any third party admins would be enabled to execute bots or third-party bots would be allowed, the attack will be valid again.

### Description

The concept behind **DCA (Dollar-Cost Averaging)** orders is to break down a large order into smaller portions and execute them over time. This approach aims to reduce slippage and avoid executing the entire order at a potentially unfavorable price in a single transaction. In the current contracts, this is achieved by executing the order at regular intervals, defined by the cycleSecondsApart parameter. Each portion of the order must be executed at a price within the range of `order.minOutAmountPerCycle` and `order.maxOutAmountPerCycle`.

The main issue with this implementation is that any user can execute the order. If an arbitrage opportunity arises, there is little incentive for the executor to aim for a price better than the lowest allowable price, which corresponds to `order.minOutAmountPerCycle`. As a result, even if the price improves over time, the maker may still only receive the minimum amount, as arbitrageurs are likely to execute the order at this lower threshold. Conversely, if the price drops below `order.minOutAmountPerCycle`, the order will remain unexecuted until the price becomes favorable again.

In summary, **DCA** orders offer little to no advantage in this context, as takers are incentivized to execute at the `order.minOutAmountPerCycle`, effectively rendering the **DCA** strategy equivalent to a standard limit order.

### Recommendation

Consider revising the economic incentives and adjusting the `order.minOutAmountPerCycle` dynamically. For instance, this parameter could be updated over time in response to price fluctuations by integrating price oracles. This approach would help align the minimum output amount with the current market conditions, reducing the likelihood of arbitrage opportunities that disadvantage the maker and making the **DCA** strategy more effective.

## 5.8 `init` Function Frontrun Medium ✓ Fixed

#### Resolution

In the `2b508ff772206751317e8b0c6f5f70d4987a2b5e` commit provided for the fix review the issue has been fixed by removing the `init` functions, and moving its logic to the constructors.

### Description

The `deploy/000_deploy.ts` and `scripts/deploy.ts` files outline the deployment scripts used by the StarBase team. Despite the fact that numerous contracts have both `constructor` and `init` functions, analysis of the StarBase team wallets activity on the mainnet and the deployment scripts reveals that the contracts do not utilize the proxy pattern. The contracts are implementations without a proxy, and the `init` function is just a separate configuration function called after deployment.

In a typical proxy pattern, when users make calls to the proxy contract, the proxy contract delegates the call to the underlying implementation contract. Implementation contracts, which contain the logic, usually include an `initialize()` function that

replaces the `constructor()` when deploying proxy contracts. It is important that these proxy contracts are deployed and initialized in the same transaction to avoid any malicious front-running.

However, the `deploy/000_deploy.ts` and `scripts/deploy.ts` files do not follow this pattern when deploying contracts, as there is no proxy involved, and the `init` function is called in a separate transaction. As a result, a malicious attacker could monitor the blockchain for bytecode that matches, for example, the `StarBaseApprove` contract and front-run the `init()` transaction to gain ownership of the contract and set their own address as `_StarBase_PROXY_`. This could allow a malicious user to steal any `permit` approvals made to the contract and access user funds.

Additionally, there is a risk that the deployer might forget to call the `init` function, leaving the contract uninitialized.

## Examples

**starbase-limitorder/src/StarBaseLimitOrderBot.sol:L35-L45**

```
function init(
    address owner,
    address StarBaseLimitOrder,
    address tokenReceiver,
    address StarBaseApprove
) external {
    initOwner(owner);
    _StarBase_LIMIT_ORDER_ = StarBaseLimitOrder;
    _TOKEN_RECEIVER_ = tokenReceiver;
    _StarBase_APPROVE_ = StarBaseApprove;
}
```

**starbase-limitorder/src/StarBaseLimitOrder.sol:L56-L61**

```
function init(address owner, address StarBaseApproveProxy, address feeReciver,uint160 feeRate) external {
    initOwner(owner);
    _StarBase_APPROVE_PROXY_ = StarBaseApproveProxy;
    _FEE_RECEIVER_ = feeReciver;
    _FEE_RATE_ = feeRate;
}
```

**starbase-limitorder/src/StarBaseDCABot.sol:L33-L43**

```
function init(
    address owner,
    address StarBaseDCA,
    address tokenReceiver,
    address StarBaseApprove
) external {
    initOwner(owner);
    _StarBase_DCA_ = StarBaseDCA;
    _TOKEN_RECEIVER_ = tokenReceiver;
    _StarBase_APPROVE_ = StarBaseApprove;
}
```

**starbase-limitorder/src/StarBaseDCA.sol:L69-L74**

```
function init(address owner, address StarBaseApproveProxy, address feeReciver, uint160 feeRate) external {
    initOwner(owner);
    _StarBase_APPROVE_PROXY_ = StarBaseApproveProxy;
    _FEE_RECEIVER_ = feeReciver;
    _FEE_RATE_ = feeRate;
}
```

**starbase-limitorder/src/StarBaseApproveProxy.sol:L42-L46**

```
function init(address owner, address[] memory proxies) external {
    initOwner(owner);
    for(uint i = 0; i < proxies.length; i++)
        _IS_ALLOWED_PROXY_[proxies[i]] = true;
}
```

**starbase-limitorder/src/StarBaseApprove.sol:L53-L57**

```
function init(address owner, address initProxyAddress) external {
    initOwner(owner);
    _StarBase_PROXY_ = initProxyAddress;
}
```

**starbase-limitorder/scripts/deploy.ts:L3-L86**

```
async function main() {
  const accounts = await ethers.getSigners()
  // use local accounts for testing
  const owner = accounts[0].address
  const feeReciver = accounts[1].address
  const AllowanceTransfer = await ethers.getContractFactory("AllowanceTransfer");
  const StarBaseApprove = await ethers.getContractFactory("StarBaseApprove");
  const StarBaseApproveProxy = await ethers.getContractFactory("StarBaseApproveProxy");
  const LimitOrder = await ethers.getContractFactory("StarBaseLimitOrder");
  const LimitOrderBot = await ethers.getContractFactory("StarBaseLimitOrderBot");
  const DCA = await ethers.getContractFactory("StarBaseDCA");
  const DCABot = await ethers.getContractFactory("StarBaseDCABot");
  const permit2 = await AllowanceTransfer.deploy();
  await permit2.deployed();
  const approve = await StarBaseApprove.deploy(permit2.address);
  await approve.deployed();
  const approveProxy = await StarBaseApproveProxy.deploy(approve.address);
  await approveProxy.deployed();
  const limitOrder = await LimitOrder.deploy();
  await limitOrder.deployed();
  const limitOrderBot = await LimitOrderBot.deploy();
  await limitOrderBot.deployed();
  const dca = await DCA.deploy();
  await dca.deployed();
  const dcaBot = await DCABot.deploy();
  await dcaBot.deployed();

  await delay(10000);


  console.log("permit2 deployed to:", permit2.address);
  console.log("approve deployed to:", approve.address);
  console.log("approveProxy deployed to:", approveProxy.address);
  console.log("limitOrder deployed to:", limitOrder.address);
  console.log("limitOrderBot deployed to:", limitOrderBot.address);
  console.log("DCA deployed to:", dca.address);
  console.log("DCABot deployed to:", dcaBot.address);

  console.log(`Set up LimitOrder and DCA...`);

  console.log(`init approve...`);
  await approve.init(owner, approveProxy.address);
  console.log(`init approveProxy...`);
  await approveProxy.init(owner,[limitOrder.address]);
  console.log(`init limitOrder...`);
  await limitOrder.init(owner, approveProxy.address, feeReciver,30);
  console.log(`init limitOrderBot...`);
  await limitOrderBot.init(owner, limitOrder.address, owner, approve.address);
  console.log(`init dca...`);
  await dca.init(owner, approveProxy.address, feeReciver,30);
  console.log(`init dcaBot...`);
  await dcaBot.init(owner, dca.address, owner, approve.address);

  await delay(10000);

  console.log(`owner is ${await limitOrder._OWNER_()}`);

  console.log(`addWhiteList limitOrder...`);
  await limitOrder.addWhiteList(limitOrderBot.address);

  console.log(`botowner is ${await limitOrderBot._OWNER_()}`);

  console.log(`addAdminList limitOrderBot...`);
  await limitOrderBot.addAdminList(owner);

  console.log(`owner is ${await dca._OWNER_()}`);

  console.log(`addWhiteList dca...`);
  await dca.addWhiteList(dcaBot.address);

  console.log(`botowner is ${await dcaBot._OWNER_()}`);
  console.log(`addAdminList dcaBot...`);
  await dcaBot.addAdminList(owner);


  verifyContract(permit2.address,[]);
  verifyContract(approve.address,[permit2.address]);
  verifyContract(approveProxy.address,[approve.address]);
  verifyContract(limitOrder.address,[]);
  verifyContract(limitOrderBot.address,[]);
  verifyContract(dca.address,[]);
  verifyContract(dcaBot.address,[]);

}
```

**starbase-limitorder/deploy/sepolia/000_deploy.ts:L12-L84**

```
async function main() {
  await deployLimitOrder();
  await deployLimitOrderBot();
  await setupLimitOrder();
  await setupLimitOrderBot();
}

async function deployContract(name: string, contract: string, args: any[]) {
  if (!config[name as keyof typeof config] || config[name as keyof typeof config] == "") {
    console.log("Deploying contract:", name);
    const deployResult = await deploy(contract, {
      from: deployer,
      args: args,
      log: true,
    });
    await verifyContract(deployResult.address, args);
    return deployResult.address;
  } else {
    console.log("Fetch previous deployed address for", name, config[name as keyof typeof config]);
    return config[name as keyof typeof config];
  }
}

async function verifyContract(address: string, args?: any[]) {
  if (typeof args == "undefined") {
    args = [];
  }
  try {
    await hre.run("verify:verify", {
      address: address,
      constructorArguments: args,
    });
  } catch (e) {
    if (e instanceof Error) {
      if (e.message != "Contract source code already verified") {
        throw e;
      }
      console.log(e.message);
    }
  }
}

async function deployLimitOrder() {
  await deployContract("StarBaseLimitOrder", "StarBaseLimitOrder", []);
}

async function deployLimitOrderBot() {
  await deployContract("StarBaseLimitOrderBot", "StarBaseLimitOrderBot", []);
}

  async function setupLimitOrder() {
    // const contractAddress = config.StarBaseLimitOrder;
    // const StarBaseLimitOrder = await ethers.getContractAt("StarBaseLimitOrder", contractAddress);
    // console.log("StarBaseLimitOrder.init()...");
    // await StarBaseLimitOrder.init(deployer, config.StarBaseApproveProxy, config.FeeReceiver);
    // console.log("StarBaseLimitOrder.addWhiteList()...");
    // await StarBaseLimitOrder.addWhiteList(config.StarBaseLimitOrderBot);
  }

  async function setupLimitOrderBot() {
    // const contractAddress = config.StarBaseLimitOrderBot;
    const StarBaseLimitOrderBot = await ethers.getContractAt("StarBaseLimitOrderBot", contractAddress);
    console.log("StarBaseLimitOrderBot.init()...");
    // await StarBaseLimitOrderBot.init(
    //   deployer,
    //   config.StarBaseLimitOrder,
    //   config.FeeReceiver,
    //   config.StarBaseApprove
    // );
    console.log("StarBaseLimitOrderBot.addAdminList()...");
    // await StarBaseLimitOrderBot.addAdminList(config.Admin);
  }
};
```

## Recommendation

We recommend reviewing the project architecture, refactoring the `init` function, and moving its logic to the constructor.

## 5.9 Cancellation of Order Does Not Invalidate `permitSingle` in `cancelOrder` Medium ✓ Fixed

### Resolution

In the `2b508ff772206751317e8b0c6f5f70d4987a2b5e` commit provided for the fix review the issue has been fixed by redesigning architecture and removing `Permit2` library with the comment: “For now, we remove the use of permit2 and use direct licensing”.

### Description

In the `cancelOrder` function, the cancellation process does not render the `permitSingle` associated with the order unusable. This omission could allow the `permitSingle` to be reused even after the order has been canceled, leading to potential misuse or unintended consequences, such as the risk of the maker’s funds being stolen.

### Examples

**starbase-limitorder/src/StarBaseLimitOrder.sol:L138-L153**

```
 function cancelOrder(Order memory order,bytes memory signature) public {
    bytes32 orderHash = _orderHash(order);


    require(order.maker == msg.sender, "SLOP:PRIVATE_ORDER");

    if (_isContract(order.maker)) {
        _verifyERC1271WalletSignature(order.maker, orderHash, signature);
    } else {
        require(ECDSA.recover(orderHash, signature) == order.maker, "SLOP:INVALID_SIGNATURE");
    }
    require(order.expiration > block.timestamp, "SLOP: EXPIRE_ORDER");

    _FILLED_TAKER_AMOUNT_[orderHash] = order.takerAmount;
        emit OrderCancelled(orderHash);
}
```

**starbase-limitorder/src/StarBaseDCA.sol:L147-L162**

```
    function cancelOrder(Order memory order, bytes memory signature) public {
        bytes32 orderHash = _orderHash(order);

        require(order.maker == msg.sender, "DCAP:PRIVATE_ORDER");


        if (_isContract(order.maker)) {
            _verifyERC1271WalletSignature(order.maker, orderHash, signature);
        } else {
            require(ECDSA.recover(orderHash, signature) == order.maker, "DCAP:INVALID_SIGNATURE");
        }
        require(order.expiration > block.timestamp, "DCAP: EXPIRE_ORDER");

        _DCA_FILLEDTIMES_[orderHash] = DCAStates(block.timestamp, order.numberOfTrade);
            emit OrderCancelled(orderHash);
    }
```

**starbase-limitorder/lib/permit2/src/AllowanceTransfer.sol:L113-L126**

```
function invalidateNonces(address token, address spender, uint48 newNonce) external {
    uint48 oldNonce = allowance[msg.sender][token][spender].nonce;

    if (newNonce <= oldNonce) revert InvalidNonce();

    // Limit the amount of nonces that can be invalidated in one transaction.
    unchecked {
        uint48 delta = newNonce - oldNonce;
        if (delta > type(uint16).max) revert ExcessiveInvalidation();
    }

    allowance[msg.sender][token][spender].nonce = newNonce;
    emit NonceInvalidation(msg.sender, token, spender, newNonce, oldNonce);
}
```

### Recommendation

We recommend calling the `invalidateNonces` function to make the `permitSingle` permit invalid together with the order cancellation. We also recommend adding tests to verify the functionality of order cancellation.

## 5.10 Missing Usage of `SafeERC20` Library Medium ✓ Fixed

**Resolution**

The Starbase team has acknowledged the issue without implementing fixes with the comment: “We did not use the `SafeErc20` library, but implemented the same functionality, with the return value already judged.”

**Update (commit hash `1693a92cc15b71f848e9ceb1d30bfa6d699b0ed3`):** Fixed by using `SafeErc20` for every token transfer.

### Description

Tokens compliant with the ERC20 specification could return `false` from the `transfer` or `transferFrom` function call to indicate the transfer has failed, while the calling contract would not notice the failure if the return value is not checked. Checking the return value is a requirement, as written in the EIP-20 specification:

> Callers MUST handle false from returns (`bool success`). Callers MUST NOT assume that `false` is never returned!

### Examples

**starbase_swap/contracts/AggregatedSwapRouter.sol:L29-L38**

```
functionCall(
    tokenIn,
    abi.encodeWithSelector(
        0x23b872dd,
        msg.sender,
        callSwapAddr,
        amountIn
    ),
    "F"
); //Swap: TRANSFER_FROM_FAILED
```

#### starbase_swap/contracts/AggregatedSwapRouter.sol:L95-L105

```
//0xa9059cbb=bytes4(keccak256(bytes('transfer(address,uint256)')));
functionCall(
    tokenIn,
    abi.encodeWithSelector(
        0x23b872dd,
        msg.sender,
        callSwapAddr,
        amountIn
    ),
    "F"
); //Swap: TRANSFER_FROM_FAILED
```

### Recommendation

We recommend using `SafeERC20` to support all tokens.

## 5.11 Unsynchronized Fee Rates Minor ✓ Fixed

### Resolution

In the `2b508ff772206751317e8b0c6f5f70d4987a2b5e` commit provided for the fix review the issue has been partially addressed with the comment: "solved(init have change construction)", where now the `StarBaseLimitOrderBot` contract initializes the initial `_FEE_RATE_` , but it's still left unsynchronized between the contracts.

**Update (commit hash `1693a92cc15b71f848e9ceb1d30bfa6d699b0ed3`):** Fixed by removing `_FEE_RATE_` from the `StarBaseLimitOrderBot` . The fees are now taken from a maker instead of a taker.

### Description

The `_FEE_RATE_` variable is being updated or initialized without ensuring synchronization with the corresponding bot contract. Specifically, the `StarBaseLimitOrder` contract is not in sync with the `StarBaseLimitOrderBot` contract. If the `_FEE_RATE_` between the contracts differs, it could result in the order reverting or behaving unexpectedly during execution.

### Examples

#### starbase-limitorder/src/StarBaseLimitOrderBot.sol:L35-L45

```
function init(
    address owner,
    address StarBaseLimitOrder,
    address tokenReceiver,
    address StarBaseApprove
) external {
    initOwner(owner);
    _StarBase_LIMIT_ORDER_ = StarBaseLimitOrder;
    _TOKEN_RECEIVER_ = tokenReceiver;
    _StarBase_APPROVE_ = StarBaseApprove;
}
```

#### starbase-limitorder/src/StarBaseLimitOrder.sol:L56-L61

```
function init(address owner, address StarBaseApproveProxy, address feeReciver,uint160 feeRate) external {
    initOwner(owner);
    _StarBase_APPROVE_PROXY_ = StarBaseApproveProxy;
    _FEE_RECEIVER_ = feeReciver;
    _FEE_RATE_ = feeRate;
}
```

### Recommendation

We recommend implementing a mechanism to ensure that the `_FEE_RATE_` is always kept in sync between the `StarBaseLimitOrder` and `StarBaseLimitOrderBot` contracts. Additionally, we recommend initializing the `_FEE_RATE_` variable in the `init` function of the `StarBaseLimitOrderBot` contract to prevent any discrepancies.

## 5.12 Missing Events in `init` Functions Minor ✓ Fixed

### Resolution

In the `2b508ff7722067513l7e8b0c6f5f70d4987a2b5e` commit provided for the fix review the finding has **not** been fixed with the comment “solved”.

**Update (commit hash `1693a92cc15b71f848e9ceb1d30bfa6d699b0ed3`):** Events added.

### Description

The code is missing events that should be emitted when certain state variables are updated, even though the events have already been declared. Specifically:

- The `_TOKEN_RECEIVER_` variable is updated without emitting a `ChangeReceiver` event. Emitting an event when the receiver is changed would provide transparency and allow off-chain systems to track these changes.
- The `_FEE_RATE_` variable is updated without emitting a `ChangeFeeRate` event. Emitting an event when the fee rate is changed would ensure that any changes to the fee structure are properly logged and can be monitored.

### Examples

**starbase-limitorder/src/StarBaseDCA.sol:L69-L74**

```
function init(address owner, address StarBaseApproveProxy, address feeReciver, uint160 feeRate) external {
    initOwner(owner);
    _StarBase_APPROVE_PROXY_ = StarBaseApproveProxy;
    _FEE_RECEIVER_ = feeReciver;
    _FEE_RATE_ = feeRate;
}
```

**starbase-limitorder/src/StarBaseDCABot.sol:L30-L43**

```
event changeReceiver(address newReceiver);
event Fill();

function init(
    address owner,
    address StarBaseDCA,
    address tokenReceiver,
    address StarBaseApprove
) external {
    initOwner(owner);
    _StarBase_DCA_ = StarBaseDCA;
    _TOKEN_RECEIVER_ = tokenReceiver;
    _StarBase_APPROVE_ = StarBaseApprove;
}
```

**starbase-limitorder/src/StarBaseLimitOrder.sol:L52-L61**

```
event ChangeFeeReceiver(address newFeeReceiver);
event OrderCancelled(bytes32 orderHash);
event ChangeFeeRate(uint160 feeRate);

function init(address owner, address StarBaseApproveProxy, address feeReciver,uint160 feeRate) external {
    initOwner(owner);
    _StarBase_APPROVE_PROXY_ = StarBaseApproveProxy;
    _FEE_RECEIVER_ = feeReciver;
    _FEE_RATE_ = feeRate;
}
```

**starbase-limitorder/src/StarBaseLimitOrderBot.sol:L31-L45**

```
event changeReceiver(address newReceiver);
event Fill();
event ChangeFeeRate(uint160 feeRate);

function init(
    address owner,
    address StarBaseLimitOrder,
    address tokenReceiver,
    address StarBaseApprove
) external {
    initOwner(owner);
    _StarBase_LIMIT_ORDER_ = StarBaseLimitOrder;
    _TOKEN_RECEIVER_ = tokenReceiver;
    _StarBase_APPROVE_ = StarBaseApprove;
}
```

### Recommendation

We recommend emitting the following events when the corresponding state variables are updated:

- Emit the `ChangeReceiver` event when the `_TOKEN_RECEIVER_` variable is updated.
- Emit the `ChangeFeeRate` event when the `_FEE_RATE_` variable is updated.

This will allow the monitoring of values and their changes off-chain, providing greater transparency and traceability of important contract operations.

## 5.13 Lack of Validation in `removeStarBaseProxy` Minor ✓ Fixed

**Resolution**

In the `2b508ff772206751317e8b0c6f5f70d4987a2b5e` commit provided for the fix review the finding has been fixed using the provided recommendation.

### Description

The `removeStarBaseProxy` function does not validate whether the `oldStarBaseProxy` address is actually in the `_IS_ALLOWED_PROXY_` list before attempting to remove it. This lack of validation can lead to potential misuse, as it may be mistakenly called on an address that was never part of the list. Additionally, this function does not clean up the `_PENDING_ADD_StarBase_PROXY_` variable, which can be unexpected to the caller, especially since the function lacks comments.

### Examples

**starbase-limitorder/src/StarBaseApproveProxy.sol:L64-L66**

```
function removeStarBaseProxy (address oldStarBaseProxy) public onlyOwner {
    _IS_ALLOWED_PROXY_[oldStarBaseProxy] = false;
}
```

### Recommendation

We recommend adding validation to check whether the `oldStarBaseProxy` address is actually in the `_IS_ALLOWED_PROXY_` list before attempting to remove it. This will prevent potential misuse and maintain the integrity of the proxy management process:

```
require(_IS_ALLOWED_PROXY_[oldStarBaseProxy], "Address is not an allowed proxy");
_IS_ALLOWED_PROXY_[oldStarBaseProxy] = false;
```

## 5.14 Missing Contract Code Check in `_callOptionalReturn` Function Minor ✓ Fixed

**Resolution**

In the `2b508ff772206751317e8b0c6f5f70d4987a2b5e` commit provided for the fix review the finding has been fixed using the provided recommendation.

### Description

The `_callOptionalReturn` function performs a low-level call to the `token` contract without first checking if the target address contains contract code. This omission can lead to unexpected behavior if the `token` address is not a contract, potentially causing the function to return `success` as `true` if the call is made to an externally owned account (EOA) address, leading to unintended behavior.

### Examples

**starbase-limitorder/src/lib/SafeERC20.sol:L67-L86**

```
function _callOptionalReturn(IERC20 token, bytes memory data) private {
    // We need to perform a low level call here, to bypass Solidity's return data size checking mechanism, since
    // we're implementing it ourselves.

    // A Solidity high level call has three parts:
    //  1. The target address is checked to verify it contains contract code
    //  2. The call itself is made, and success asserted
    //  3. The return value is decoded, which in turn checks the size of the returned data.
    // solhint-disable-next-line max-line-length

    // solhint-disable-next-line avoid-low-level-calls
    (bool success, bytes memory returndata) = address(token).call(data);
    require(success, "SafeERC20: low-level call failed");

    if (returndata.length > 0) {
        // Return data is optional
        // solhint-disable-next-line max-line-length
        require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
    }
}
```

### Recommendation

We recommend adding a check to ensure that the `token` address is indeed a contract before performing the low-level call. This can be done by verifying that the address has contract code, as suggested in the OpenZeppelin implementation. This check will prevent unexpected behavior and improve the security of the function:

```
require(Address.isContract(address(token)), "SafeERC20: call to non-contract");

(bool success, bytes memory returndata) = address(token).call(data);
require(success, "SafeERC20: low-level call failed");
```

## 5.15 Missing Validations in `constructor`, `initializer` and Setter Functions Minor ✓ Fixed

**Resolution**

In the `2b508ff772206751317e8b0c6f5f70d4987a2b5e` commit provided for the fix review, the finding has been partially fixed using the provided recommendation.

**Update (commit hash `be86d6b0940556113cf04f0298c868502a58926a`):** More checks are added.

## Description

In the code repository, there is a lack of validation for input variables in `constructor` and `initializer` functions, as well as in various setter functions. Specifically, there is no validation to ensure that the provided addresses implement the correct interface using `ERC165Checker`. Additionally, there is no validation to ensure that the input variable is not a zero address. Without these validations, there is a risk that incorrect or incompatible contracts could be assigned to these variables, potentially leading to unexpected behavior or contract failures.

## Examples

### starbase_swap/contracts/AggregatedSwapRouter.sol:L14-L17

```
constructor(address CallSwapTool_, address IWETH_) {
    _CallSwapTool = CallSwapTool_;
    _IWETH = IWETH_;
}
```

### starbase-limitorder/src/StarBaseApprove.sol:L49-L56

```
constructor(address permit2){
    PERMIT2 = IAllowanceTransfer(permit2);
}

function init(address owner, address initProxyAddress) external {
    initOwner(owner);
    _StarBase_PROXY_ = initProxyAddress;
}
```

### starbase-limitorder/src/StarBaseApprove.sol:L58-L64

```
function unlockSetProxy(address newStarBaseProxy) public onlyOwner {
    if(_StarBase_PROXY_ == address(0))
        _TIMELOCK_ = block.timestamp + _TIMELOCK_EMERGENCY_DURATION_;
    else
        _TIMELOCK_ = block.timestamp + _TIMELOCK_DURATION_;
    _PENDING_StarBase_PROXY_ = newStarBaseProxy;
}
```

### starbase-limitorder/src/StarBaseApproveProxy.sol:L38-L51

```
constructor(address StarBaseApporve) {
    _StarBase_APPROVE_ = StarBaseApporve;
}

function init(address owner, address[] memory proxies) external {
    initOwner(owner);
    for(uint i = 0; i < proxies.length; i++)
        _IS_ALLOWED_PROXY_[proxies[i]] = true;
}

function unlockAddProxy(address newStarBaseProxy) public onlyOwner {
    _TIMELOCK_ = block.timestamp + _TIMELOCK_DURATION_;
    _PENDING_ADD_StarBase_PROXY_ = newStarBaseProxy;
}
```

### starbase-limitorder/src/StarBaseDCA.sol:L69-L74

```
function init(address owner, address StarBaseApproveProxy, address feeReciver, uint160 feeRate) external {
    initOwner(owner);
    _StarBase_APPROVE_PROXY_ = StarBaseApproveProxy;
    _FEE_RECEIVER_ = feeReciver;
    _FEE_RATE_ = feeRate;
}
```

### starbase-limitorder/src/StarBaseDCA.sol:L176-L179

```
function addWhiteList(address contractAddr) public onlyOwner {
    isWhiteListed[contractAddr] = true;
    emit AddWhiteList(contractAddr);
}
```

### starbase-limitorder/src/StarBaseDCA.sol:L186-L188

```
function changeFeeReceiver(address newFeeReceiver) public onlyOwner {
    _FEE_RECEIVER_ = newFeeReceiver;
    emit ChangeFeeReceiver(newFeeReceiver);
```

**starbase-limitorder/src/StarBaseDCABot.sol:L33-L43**

```
function init(
    address owner,
    address StarBaseDCA,
    address tokenReceiver,
    address StarBaseApprove
) external {
    initOwner(owner);
    _StarBase_DCA_ = StarBaseDCA;
    _TOKEN_RECEIVER_ = tokenReceiver;
    _StarBase_APPROVE_ = StarBaseApprove;
}
```

**starbase-limitorder/src/StarBaseDCABot.sol:L106-L109**

```
function addAdminList (address userAddr) external onlyOwner {
    isAdminListed[userAddr] = true;
    emit addAdmin(userAddr);
}
```

**starbase-limitorder/src/StarBaseDCABot.sol:L116-L119**

```
function changeTokenReceiver(address newTokenReceiver) external onlyOwner {
    _TOKEN_RECEIVER_ = newTokenReceiver;
    emit changeReceiver(newTokenReceiver);
}
```

**starbase-limitorder/src/StarBaseLimitOrder.sol:L56-L61**

```
function init(address owner, address StarBaseApproveProxy, address feeReciver,uint160 feeRate) external {
    initOwner(owner);
    _StarBase_APPROVE_PROXY_ = StarBaseApproveProxy;
    _FEE_RECEIVER_ = feeReciver;
    _FEE_RATE_ = feeRate;
}
```

**starbase-limitorder/src/StarBaseLimitOrder.sol:L168-L171**

```
function addWhiteList (address contractAddr) public onlyOwner {
    isWhiteListed[contractAddr] = true;
    emit AddWhiteList(contractAddr);
}
```

**starbase-limitorder/src/StarBaseLimitOrder.sol:L178-L181**

```
function changeFeeReceiver (address newFeeReceiver) public onlyOwner {
    _FEE_RECEIVER_ = newFeeReceiver;
    emit ChangeFeeReceiver(newFeeReceiver);
}
```

**starbase-limitorder/src/StarBaseLimitOrderBot.sol:L35-L45**

```
function init(
    address owner,
    address StarBaseLimitOrder,
    address tokenReceiver,
    address StarBaseApprove
) external {
    initOwner(owner);
    _StarBase_LIMIT_ORDER_ = StarBaseLimitOrder;
    _TOKEN_RECEIVER_ = tokenReceiver;
    _StarBase_APPROVE_ = StarBaseApprove;
}
```

**starbase-limitorder/src/StarBaseLimitOrderBot.sol:L104-L107**

```
function addAdminList (address userAddr) external onlyOwner {
    isAdminListed[userAddr] = true;
    emit addAdmin(userAddr);
}
```

**starbase-limitorder/src/StarBaseLimitOrderBot.sol:L114-L117**

```
function changeTokenReceiver(address newTokenReceiver) external onlyOwner {
    _TOKEN_RECEIVER_ = newTokenReceiver;
    emit changeReceiver(newTokenReceiver);
}
```

### Recommendation

We recommend adding `ERC165Checker` interface validation in the constructor and initializer functions to ensure that the provided addresses implement the required interfaces and are valid. Additionally, ensure that input variables are not zero addresses. This will help ensure that only compatible contracts are used, reducing the risk of errors or unexpected behavior.

## 5.16 Redundant `add` Operation in Assembly Code Minor ✓ Fixed

| Resolution |
| --- |
| The finding has been acknowledged. |

### Description

In the assembly block, the `add( ..., 0)` operation is redundant and does not contribute to the calculation. This inefficiency could be avoided, as adding zero does not change the value. These unnecessary operations should be optimized for cleaner and more gas-efficient code.

### Examples

**starbase_swap/contracts/AggregatedSwapRouter.sol:L131**

```
remain := mload(add(add(_assetTransfer, 0x20), 0))
```

### Recommendation

We recommend removing the unnecessary `add 0` operation in the assembly code to optimize the code's gas efficiency and keep the codebase clean.

## 5.17 Redundant and Unvalidated `maxOutAmount` in `StarBaseDCA` Minor ✓ Fixed

| Resolution |
| --- |
| In the `2b508ff7722067513l7e8b0c6f5f70d4987a2b5e` commit provided for the fix review the finding has been fixed using the provided recommendation with the only difference that `maxOutAmount` can be equal `minOutAmount`. |

### Description

In the `Order` struct, there are two variables that can be specified by the user: `minOutAmountPerCycle` and `maxOutAmountPerCycle`. During the `fillDCA` call, these variables are never validated to ensure that `maxOutAmountPerCycle` is greater than `minOutAmountPerCycle`. For example, in the `doDCASwap` function of `StarBaseDCABot`, if `maxOutAmountPerCycle` is smaller than `minOutAmountPerCycle`, any additional tokens swapped and received by the taker will remain at the taker's address as an additional fee. This is especially dangerous when the user or frontend doesn't specify `maxOutAmountPerCycle` explicitly and passes this variable as `0`, while correctly specifying `minOutAmountPerCycle`. In such cases, all tokens received after the trade will go to the taker's address.

The existence of the `maxOutAmountPerCycle` variable in the `Order` struct is not recommended. The `minOutAmountPerCycle` variable plays a crucial role in protecting the maker's funds from receiving nothing after the trade. However, the `maxOutAmountPerCycle` variable does not benefit the maker, and all maker addresses will likely specify this variable as `type(uint256).max`. As a result, the check in `doDCASwap` will never be used and will only consume excessive gas.

### Examples

**starbase-limitorder/src/StarBaseDCA.sol:L26-L37**

```
struct Order {
    uint16 cycleSecondsApart; // executed per minute
    uint16 numberOfTrade; // executed 5 times
    address inputToken; // sell
    address outputToken; // buy
    address maker;
    uint160 inAmount; // total principal
    uint256 minOutAmountPerCycle; //min out amount
    uint256 maxOutAmountPerCycle; //max out amount
    uint256 expiration;
    uint256 salt;
}
```

**starbase-limitorder/src/StarBaseDCABot.sol:L95-L98**

```
require(returnTakerAmount >= minOutAmount, "SWAP_TAKER_AMOUNT_NOT_ENOUGH");
if(returnTakerAmount > maxOutAmount){
    returnTakerAmount = maxOutAmount;
}
```

### Recommendation

We recommend fully removing the `maxOutAmountPerCycle` variable from the `Order` struct. However, if it is crucial for the protocol design, we recommend adding a validation check at the beginning of the `fillDCA` function to ensure that `maxOutAmountPerCycle` is greater than `minOutAmountPerCycle`:

```
require(maxOutAmount > minOutAmount, "maxOutAmount must be greater than minOutAmount");
```

## 5.18 Missing Validation of `receiver` in `AggregatedSwapRouter` Minor ✓ Fixed

### Resolution

In the `d81b6f90d52b12dcfd6f05f023b19ca6e9a8c9e2` commit provided for the fix review the finding has been partially fixed using the provided recommendation, the check hasn't been added to `swapFromEth`, `defiSwapFromEth` functions, also, the check should be done as a modifier to make the code cleaner and more gas efficient.

**Update (commit hash `7bd9750abbf283970167a4b9b475633481a38d50`):** Fixed.

### Description

In the following functions:

- `swap`
- `defiSwap`
- `defiSwapForEth`
- `swapForEth`
- `defiSwapFromEth`
- `swapFromEth`

there is no validation to ensure that the `receiver` address is not the zero address, the `_CallSwapTool` address, or the `AggregatedSwapRouter` address. Allowing a transaction to proceed with a zero address as the `receiver` can lead to unintended behavior, such as tokens being irretrievably lost and burned at the zero address, resulting in a loss for the original caller.

### Examples

**starbase_swap/contracts/AggregatedSwapRouter.sol:L19-L24**

```
function swap(
    uint amountIn,
    uint amountOutMin,
    address tokenIn,
    address tokenOut,
    address receiver,
```

**starbase_swap/contracts/AggregatedSwapRouter.sol:L48-L53**

```
function defiSwap(
    uint amountIn,
    uint amountOutMin,
    address tokenIn,
    address tokenOut,
    address receiver,
```

**starbase_swap/contracts/AggregatedSwapRouter.sol:L70-L74**

```
function defiSwapForEth(
    uint amountIn,
    uint amountOutMin,
    address tokenIn,
    address payable receiver,
```

**starbase_swap/contracts/AggregatedSwapRouter.sol:L87-L91**

```
function swapForEth(
    uint amountIn,
    uint amountOutMin,
    address tokenIn,
    address payable receiver,
```

**starbase_swap/contracts/AggregatedSwapRouter.sol:L111-L114**

```
function defiSwapFromEth(
    uint amountOutMin,
    address tokenOut,
    address receiver,
```

**starbase_swap/contracts/AggregatedSwapRouter.sol:L142-L145**

```
function swapFromEth(
    uint amountOutMin,
    address tokenOut,
    address receiver,
```

### Recommendation

We recommend adding a modifier with validation to ensure that the `receiver` is not a zero address.

## 5.19 Infinite Allowance Risks ✓ Fixed

| Resolution |
| --- |
| **Update (commit hash `1693a92cc15b71f848e9ceb1d30bfa6d699b0ed3`):** Mitigated by only giving the approval when it's needed. |

### Description

In the `doLimitOrderSwap` and `doDCASwap` functions of the `StarBaseLimitOrderBot` and `StarBaseDCABot` contracts, there is an approval for the max value to the `_StarBase_APPROVE_` address. However, based on the protocol's flow, this address might not be the `StarBaseApprove` contract, but rather a contract that will execute the swap. Due to the lack of tests and the uncertainty of the actual implementation of the `_StarBase_APPROVE_` address, there is a risk that tokens on the `StarBaseLimitOrderBot` could be stolen because of the infinite allowance granted to the `_StarBase_APPROVE_` address. Additionally, in the `doLimitOrderSwap` function of the `StarBaseLimitOrderBot` contract, tokens left on the balance as fees that could be stolen, depending on the implementation of the `_StarBase_APPROVE_` contract.

### Examples

**starbase-limitorder/src/StarBaseLimitOrderBot.sol:L73-L84**

```
function doLimitOrderSwap(
    uint256 curTakerFillAmount,
    uint256 curMakerFillAmount,
    address makerToken, //fromToken
    address takerToken, //toToken
    address StarBaseRouteProxy,
    bytes memory StarBaseApiData
) external {
    require(msg.sender == _StarBase_LIMIT_ORDER_, "ACCESS_NENIED");
    uint256 originTakerBalance = IERC20(takerToken).balanceOf(address(this));

    _approveMax(IERC20(makerToken), _StarBase_APPROVE_, curMakerFillAmount);
```

**starbase-limitorder/src/StarBaseDCABot.sol:L71-L83**

```
function doDCASwap(
    uint256 inAmount,
    uint256 minOutAmount,
    uint256 maxOutAmount,
    address inputToken, //fromToken
    address outputToken, //toToken
    address StarBaseRouteProxy,
    bytes memory StarBaseApiData
) external returns (uint256 returnTakerAmount){
    require(msg.sender == _StarBase_DCA_, "ACCESS_NENIED");
    uint256 originTakerBalance = IERC20(outputToken).balanceOf(address(this));

    _approveMax(IERC20(inputToken), _StarBase_APPROVE_, inAmount);
```

**starbase-limitorder/src/StarBaseLimitOrderBot.sol:L93-L99**

```
uint256 takerBalance = IERC20(takerToken).balanceOf(address(this));
uint256 returnTakerAmount = takerBalance - originTakerBalance;
uint256 fee = curTakerFillAmount * _FEE_RATE_ / 10000;

require(returnTakerAmount >= curTakerFillAmount + fee, "SWAP_TAKER_AMOUNT_NOT_ENOUGH");

_approveMax(IERC20(takerToken), _StarBase_LIMIT_ORDER_, curTakerFillAmount);
```

### Recommendation

We recommend reviewing the implementations of all addresses where infinite allowance is granted. Additionally, add tests to clarify which address is expected to be used as the `_StarBase_APPROVE_` address. To minimize the risk of token loss, we also recommend using finite allowances for external contracts instead of granting infinite allowances.

## 5.20 Unfulfillable Orders Due to Mismatched Expiration Times Partially Addressed

| Resolution |
| --- |
| **Update (commit hash `1693a92cc15b71f848e9ceb1d30bfa6d699b0ed3`):** Partially mitigated by removing permit mechanics. It is still possible that an order can have an expiration time that is too low to be fully executed. This part is hard to mitigate as the order can become unfulfillable over time, which is natural for the expiration mechanism. |

### Description

In the current implementation, there is no validation to ensure that the order's `expiration` aligns with the product of `cycleSecondsApart * numberOfTrade`, which can result in the latter being greater than the order's expiration time. Additionally, the permit expiration might be smaller than the order's `expiration`, leading to situations where an order cannot be fully executed. This discrepancy could result in orders that are technically valid but cannot be completed as expected.

### Examples

**starbase-limitorder/src/StarBaseDCA.sol:L26-L37**

```
struct Order {
    uint16 cycleSecondsApart; // executed per minute
    uint16 numberOfTrade; // executed 5 times
    address inputToken; // sell
    address outputToken; // buy
    address maker;
    uint160 inAmount; // total principal
    uint256 minOutAmountPerCycle; //min out amount
    uint256 maxOutAmountPerCycle; //max out amount
    uint256 expiration;
    uint256 salt;
}
```

**starbase-limitorder/lib/permit2/src/interfaces/IAllowanceTransfer.sol:L32-L39**

```
event Permit(
    address indexed owner,
    address indexed token,
    address indexed spender,
    uint160 amount,
    uint48 expiration,
    uint48 nonce
);
```

**starbase-limitorder/src/StarBaseLimitOrder.sol:L26-L34**

```
struct Order {
    address makerToken;
    address takerToken;
    uint160 makerAmount;
    uint160 takerAmount;
    address maker;
    uint256 expiration;
    uint256 salt;
}
```

### Recommendation

We recommend implementing validation logic to invalidate orders that cannot be fulfilled due to mismatched expiration times. Specifically:

1. Validate `cycleSecondsApart * numberOfTrade`: Ensure that this product is always smaller than the order expiration time. If not, the order should be marked as invalid.
2. Check permit `expiration`: Ensure that the permit expiration is not smaller than the order expiration. If the permit expires before the order, it should be invalidated to prevent unfulfillable orders.

## 5.21 Unnecessary Variable Initialization ✓ Fixed

**Resolution**

In the `2b508ff772206751317e8b0c6f5f70d4987a2b5e` commit provided for the fix review the finding has been fixed using the provided recommendation, however in the `StarBaseLimitOrderBot` contract the new problem has occurred as the validation for the return taker amount has been fully removed. In the `042349cc0c0bfbde14456366465c6a98bdc14787` commit the check has been returned, but without the `fee` validation.

### Description

In the code, the `takerBalance` variable is declared immediately after a successful swap operation by calling `IERC20(takerToken).balanceOf(address(this))`. However, this initialization can be optimized out, as it is simply being used to calculate the difference from the `originTakerBalance`. The initialization of the `takerBalance` variable is unnecessary and adds an extra step that can be avoided to save gas.

### Examples

**starbase-limitorder/src/StarBaseLimitOrderBot.sol:L93-L94**

```
uint256 takerBalance = IERC20(takerToken).balanceOf(address(this));
uint256 returnTakerAmount = takerBalance - originTakerBalance;
```

**starbase-limitorder/src/StarBaseLimitOrderBot.sol:L62-L63**

```
uint256 takerBalance = IERC20(takerToken).balanceOf(address(this));
uint256 leftTakerAmount = takerBalance - originTakerBalance;
```

**starbase-limitorder/src/StarBaseDCABot.sol:L60-L61**

```
uint256 takerBalance = IERC20(outputToken).balanceOf(address(this));
uint256 leftTakerAmount = takerBalance - originTakerBalance;
```

**starbase-limitorder/src/StarBaseDCABot.sol:L92-L93**

```
uint256 takerBalance = IERC20(outputToken).balanceOf(address(this));
returnTakerAmount = takerBalance - originTakerBalance;
```

### Recommendation

We recommend optimizing the code by eliminating the initialization of the `takerBalance` variable. Instead, directly calculate the difference between the new balance and `originTakerBalance`:

```
uint256 returnTakerAmount = IERC20(takerToken).balanceOf(address(this)) - originTakerBalance;
```

## 5.22 Lack of Testing for the Codebase ✓ Fixed

| Resolution |
| --- |
| **Update (commit hash `1693a92cc15b71f848e9ceb1d30bfa6d699b0ed3`):** The test suite significantly improved. |

### Description

The codebase lacks sufficient testing, making it difficult to understand how the contracts are expected to operate and leading to issues where some functions does not work as intended or completely doesn't work. Without comprehensive tests, it is challenging to verify that the contracts behave as expected under various scenarios, including edge cases and potential vulnerabilities. The absence of tests significantly increases the risk of deploying contracts with hidden bugs or flaws.

### Recommendation

We strongly recommend implementing a thorough testing suite for the entire codebase. Aim to achieve at least 80% test coverage, ensuring that all major functionalities are covered. Include tests for edge cases, as well as multiple expected scenarios. Ensure that test descriptions accurately reflect the functionality being tested. This will improve the reliability and security of the contracts before they are deployed in a production environment.

## 5.23 Unbounded Gas Consumption in `_verifyERC1271WalletSignature` ✓ Fixed

| Resolution |
| --- |
| **Update (commit hash `1693a92cc15b71f848e9ceb1d30bfa6d699b0ed3`):** Fixed by adding the limit. |

### Description

The `_verifyERC1271WalletSignature` function includes an external call to the `isValidSignature` method of an ERC1271 wallet. This external call is unbounded in terms of gas consumption, meaning that a malicious maker could exploit this to drain gas from the taker during the execution of their order. Since the gas cost of this external call is not controlled, it could potentially lead to denial of service by exhausting the gas provided for the transaction.

### Examples

**starbase-limitorder/src/StarBaseDCA.sol:L218-L221**

```
function _verifyERC1271WalletSignature(address _addr, bytes32 _hash, bytes memory _signature) internal view {
    bytes4 result = IERC1271Wallet(_addr).isValidSignature(_hash, _signature);
    require(result == 0x1626ba7e, "INVALID_SIGNATURE");
}
```

### Recommendation

We recommend that the `taker` always validate the gas sent before executing transactions involving the `_verifyERC1271WalletSignature` function.

## 5.24 Missing Balance Check Before Token Transfer in `IERC20.safeTransferFrom` Call ✓ Fixed

| Resolution |
| --- |
| Starbase acknowledged the issue with the comment: "There is no need to check itself internally already". |

**Update (commit hash `1693a92cc15b71f848e9ceb1d30bfa6d699b0ed3`):** Fixed by adding the required check.

### Description

In the `fillDCA` function, there is no check to ensure that the `msg.sender` has enough tokens before attempting to transfer `curTakerFillAmount` from the taker to the maker, while this check is present in the very similar `fillLimitOrder` function.

### Examples

**starbase-limitorder/src/StarBaseLimitOrder.sol:L113-L117**

```
require(IERC20(order.takerToken).balanceOf(msg.sender) > fee + curTakerFillAmount,"SLOP: INFINCIENT_BALANCE");
sendTaker(order, msg.sender,  curTakerFillAmount);
sendFee(order, msg.sender, _FEE_RECEIVER_, fee);

emit LimitOrderFilled(order.maker, msg.sender, orderHash, curTakerFillAmount, curMakerFillAmount);
```

**starbase-limitorder/src/StarBaseDCA.sol:L141-L144**

```
//Taker => Maker
IERC20(order.outputToken).safeTransferFrom(msg.sender, order.maker, curTakerFillAmount);

emit DCAFilled(order.maker, msg.sender, orderHash, curTakerFillAmount, order.inAmount);
```

### Recommendation

We recommend reviewing the code and adding a check to ensure that `msg.sender` has a sufficient balance of `order.outputToken` before attempting the transfer in the `fillDCA` function. Alternatively, consider removing the same check in the `fillLimitOrder` function, as the function will still revert if the taker doesn't have enough tokens, but with a different error message.

## 5.25 `package.json` Has Incorrect Name and Doesn't Lock Versions ✓ Fixed

**Resolution**

In the `2b508ff772206751317e8b0c6f5f70d4987a2b5e` commit provided for the fix review the finding has been fixed using the provided recommendation.

### Description

In the `package.json` file, the OpenZeppelin library is not version-locked, which can lead to compatibility issues with newer versions of the library. For example, the code may not compile with OpenZeppelin version `4.9.0`. During the audit, the package was compiled with OpenZeppelin version `4.8.0`. Different versions of OpenZeppelin can have different vulnerabilities, some of which may be publicly disclosed.

Additionally, the name of the package is set to `combination-pizza-hut-and-taco-bell`, which is inappropriate for a production environment and should be changed to something more relevant.

### Examples

**starbase-limitorder/package.json:L2**

```
"name": "combination-pizza-hut-and-taco-bell",
```

**starbase-limitorder/package.json:L36**

```
"@openzeppelin/contracts": "^4.8.0",
```

### Recommendation

We recommend locking the OpenZeppelin package version in the `package.json` to `4.8.0` to ensure consistent builds and compatibility. Additionally, consider locking versions of other packages to prevent potential issues during deployment. Finally, rename the package to something more appropriate for a production environment.

## 5.26 Redundant Function `getStarBaseProxy`, `isAllowedProxy` ✓ Fixed

**Resolution**

In the `2b508ff772206751317e8b0c6f5f70d4987a2b5e` commit provided for the fix review the finding has been fixed using the provided recommendation, however we also recommend removing the `getStarBaseProxy` function from the `IStarBaseApprove` interface.

### Description

The `getStarBaseProxy` function is unnecessary because the `_StarBase_PROXY_` variable is already public, meaning it can be accessed directly without the need for a getter function. Including such a redundant function adds unnecessary complexity and clutters the codebase.

Similarly, the `isAllowedProxy` function is redundant because the `_IS_ALLOWED_PROXY_` mapping is already declared as public. In Solidity, public mappings automatically have a getter function generated for them, allowing external contracts and users to query their values directly. Including this additional function adds unnecessary code without providing any extra functionality.

### Examples

**starbase-limitorder/src/StarBaseApproveProxy.sol:L85-L87**

```
function isAllowedProxy(address _proxy) external view returns (bool) {
    return _IS_ALLOWED_PROXY_[_proxy];
}
```

**starbase-limitorder/src/StarBaseApprove.sol:L98-L100**

```
function getStarBaseProxy() public view returns (address) {
    return _StarBase_PROXY_;
}
```

### Recommendation

We recommend removing the `getStarBaseProxy` and `isAllowedProxy` functions from the contracts to simplify the code. Since `_StarBase_PROXY_` is public, it can be accessed directly without needing a separate function. This will reduce code redundancy and improve the clarity and maintainability of the contract.

## 5.27 Dead Code in Contracts ✓ Fixed

**Resolution**

In the `2b508ff772206751317e8b0c6f5f70d4987a2b5e` commit provided for the fix review the finding has been partially fixed using the provided recommendation, but the dead code in the `AggregatedSwapRouter` contract hasn't been removed.

**Update (commit hash `7bd9750abbf283970167a4b9b475633481a38d50`):** Removed.

### Description

The code contains commented-out lines that serve no functional purpose and are considered dead code. Dead code can clutter the codebase, reduce readability, and potentially cause confusion for developers and auditors reviewing the contract.

### Examples

**starbase-limitorder/src/StarBaseApprove.sol:L90**

```
//IERC20(token).safeTransferFrom(who, dest, amount);
```

**starbase-limitorder/src/StarBaseLimitOrder.sol:L125**

```
//IERC20(order.takerToken).safeTransferFrom(msg.sender, _FEE_RECEIVER_ , fee);
```

**starbase-limitorder/src/StarBaseLimitOrder.sol:L134**

```
//IERC20(order.takerToken).safeTransferFrom(msg.sender, _FEE_RECEIVER_ , fee);
```

**starbase_swap/contracts/AggregatedSwapRouter.sol:L159**

```
//0x94D020E1AE0c95d574a43aD8E4A197cf2c2eCc5F
```

### Recommendation

We recommend removing this dead code to improve the readability and maintainability of the codebase. Cleaning up unnecessary code helps to prevent misunderstandings and makes the code easier to audit and maintain. If the functionality represented by this code is no longer needed, it should be removed entirely.

## 5.28 Functions Can Be Marked as `external` Instead of `public` ✓ Fixed

**Resolution**

In the `2b508ff772206751317e8b0c6f5f70d4987a2b5e` commit provided for the fix review the finding has been partially fixed using the provided recommendation.

**Update (commit hash `1693a92cc15b71f848e9ceb1d30bfa6d699b0ed3`):** Fixed.

### Description

Several functions in the codebase are marked as `public` but can be changed to `external` to optimize gas usage and contract bytecode size. The following functions can be updated to `external`:

- `addWhiteList`
- `removeWhiteList`
- `changeFeeReceiver` (both overloads)
- `changeFeeRate`
- `unlockSetProxy`
- `unlockAddProxy`
- `removeStarBaseProxy`
- `fillDCA`
- `fillLimitOrder`

## Examples

**starbase-limitorder/src/StarBaseDCA.sol:L77-L83**

```
function fillDCA(
    Order memory order,
    bytes memory signature,
    bytes memory takerInteraction,
    IAllowanceTransfer.PermitSingle calldata permitSingle,
    bytes calldata permitSignature
) public nonReentrant returns (uint256 curTakerFillAmount) {
```

**starbase-limitorder/src/StarBaseLimitOrder.sol:L64-L72**

```
function fillLimitOrder(
    Order calldata order,
    bytes memory signature,
    uint160 takerFillAmount,
    uint160 thresholdTakerAmount,
    bytes memory takerInteraction,
    IAllowanceTransfer.PermitSingle calldata permitSingle,
    bytes calldata permitSignature
) public nonReentrant returns(uint160 curTakerFillAmount, uint160 curMakerFillAmount) {
```

**starbase-limitorder/src/StarBaseApprove.sol:L58**

```
function unlockSetProxy(address newStarBaseProxy) public onlyOwner {
```

**starbase-limitorder/src/StarBaseApproveProxy.sol:L48**

```
function unlockAddProxy(address newStarBaseProxy) public onlyOwner {
```

**starbase-limitorder/src/StarBaseApproveProxy.sol:L64**

```
function removeStarBaseProxy (address oldStarBaseProxy) public onlyOwner {
```

**starbase-limitorder/src/StarBaseDCA.sol:L176-L194**

```
function addWhiteList(address contractAddr) public onlyOwner {
    isWhiteListed[contractAddr] = true;
    emit AddWhiteList(contractAddr);
}

function removeWhiteList(address contractAddr) public onlyOwner {
    isWhiteListed[contractAddr] = false;
    emit RemoveWhiteList(contractAddr);
}

function changeFeeReceiver(address newFeeReceiver) public onlyOwner {
    _FEE_RECEIVER_ = newFeeReceiver;
    emit ChangeFeeReceiver(newFeeReceiver);
}

function changeFeeReceiver(uint160 feeRate) public onlyOwner {
    _FEE_RATE_ = feeRate;
    emit ChangeFeeRate(feeRate);
}
```

**starbase-limitorder/src/StarBaseLimitOrder.sol:L168-L186**

```
function addWhiteList (address contractAddr) public onlyOwner {
    isWhiteListed[contractAddr] = true;
    emit AddWhiteList(contractAddr);
}

function removeWhiteList (address contractAddr) public onlyOwner {
    isWhiteListed[contractAddr] = false;
    emit RemoveWhiteList(contractAddr);
}

function changeFeeReceiver (address newFeeReceiver) public onlyOwner {
    _FEE_RECEIVER_ = newFeeReceiver;
    emit ChangeFeeReceiver(newFeeReceiver);
}

function changeFeeRate (uint160 feeRate) public onlyOwner {
    _FEE_RATE_ = feeRate;
    emit ChangeFeeRate(feeRate);
}
```

### Recommendation

We recommend changing the visibility of these functions from `public` to `external` to optimize gas usage, as they are only called externally and not within the contract itself.

## 5.29 Compromised Developer Addresses Acknowledged

### Description

When the team sent the repository, they accidentally included a `.env` file containing the private keys of the addresses `0x0A64Bc73793FAf399Adb51EBAd204Acb11F0ae64` and `0x1ce00e6895e44d2e8d17d96c65bf46f679053731`. These private keys should be considered compromised and should never be used in the future. It is crucial to handle these addresses with extreme caution, especially when dealing with private keys or any funds associated with them.

### Recommendation

We recommend immediately marking these addresses as compromised and refraining from any further use of these addresses for transactions or development purposes. Additionally, ensure that private keys for any developer addresses containing funds are stored securely and not shared carelessly. Consider rotating the keys or creating new addresses for future use and removing any compromised keys from all environments and repositories.

## 5.30 Lack of NatSpec Documentation and Comments Throughout the Codebase ✓ Fixed

**Resolution**

**Update (commit hash `d323fe3cc9c939518cd631d63a8952bf4465ba16`):** The team has done a lot of work adding comments and documentation.

### Description

Most of the contracts and functions in the audited codebase lack sufficient comments throughout the code. Well-commented code improves both the speed and depth of an audit, helping to clarify developer intentions and identify discrepancies between intended and actual implementation. The absence of comments makes it much more difficult to detect potential issues. Beyond aiding auditors, comments are invaluable to future developers and users by clearly defining code functionality and reducing the likelihood of bugs. Additionally, the contracts lack proper documentation in the Ethereum Natural Specification Format (NatSpec).

### Recommendation

We strongly recommend thoroughly commenting the existing code and incorporating regular commenting into the development process. In particular, we advise commenting on every line of assembly code and providing detailed explanations for complex mathematical operations. Additionally, we recommend using NatSpec for documenting functions, parameters, return values, and events, ensuring that the code is easier to understand and maintain.

## 5.31 Unused Imports and Files ✓ Fixed

**Resolution**

In the `2b508ff7722067513l7e8b0c6f5f70d4987a2b5e` and `d81b6f90d52b12dcfd6f05f023b19ca6e9a8c9e2` commits provided for the fix review the finding has been partially fixed using the provided recommendation, the `ReentrancyGuard` contract in the `AggregatedSwapRouter` contract was left untouched.

**Update (commit hash `7bd9750abbf283970167a4b9b475633481a38d50`):** Fixed.

### Description

In the codebase, there are numerous unused imports of contracts, as well as unused files:

- The import statement for `ReentrancyGuard` from `@openzeppelin/contracts/security/ReentrancyGuard.sol` is not utilized in the code, although it should be used in all of the swap functions.
- The import of the `IERC20` interface from `./intf/IERC20.sol` is unused and should be removed.

- The import of the `SafeERC20` library from `./lib/SafeERC20.sol` is unused and should be removed.
- The import of the `IStarBaseApprove` interface from `./intf/IStarBaseApprove.sol` is unused. The contract should inherit this interface.
- The `SafeMath` library is unused and should be removed.
- The `StarBaseApproveProxy` contract is missing the inheritance of the `IStarBaseApproveProxy` interface.
- In the `IStarBaseApproveProxy` interface, the `IStarBaseApprove` import is not used and should be removed.

### Examples

**starbase_swap/contracts/AggregatedSwapRouter.sol:L8**

```
import {ReentrancyGuard} from "@openzeppelin/contracts/security/ReentrancyGuard.sol";
```

**starbase-limitorder/src/StarBaseApprove.sol:L10-L11**

```
import {IERC20} from "./intf/IERC20.sol";
import {SafeERC20} from "./lib/SafeERC20.sol";
```

**starbase-limitorder/src/StarBaseApprove.sol:L13**

```
import {IAllowanceTransfer,IStarBaseApprove} from "./intf/IStarBaseApprove.sol";
```

**starbase-limitorder/src/lib/SafeMath.sol:L11-L17**

```
/**
 * @title SafeMath
 * @author StarBase  Simon
 *
 * @notice Math operations with safety checks that revert on error
 */
library SafeMath {
```

**starbase-limitorder/src/intf/IStarBaseApproveProxy.sol:L9**

```
import {IAllowanceTransfer, IStarBaseApprove} from "./IStarBaseApprove.sol";
```

### Recommendation

We recommend removing the unused imports to improve the readability and efficiency of the code. Removing unnecessary code can also help in avoiding potential conflicts or issues during compilation and deployment.

## 5.32 Typos in the Code ✓ Fixed

**Resolution**

In the `2b508ff772206751317e8b0c6f5f70d4987a2b5e` commit provided for the fix review the finding has been partially fixed using the provided recommendation by fixing `changeFeeReceiver` name, while all other typos were left untouched.

**Update (commit hash `bc717783e5dbea806457456d9985db73cd0728c5`):** Fixed.

### Description

In the repository, there are numerous typos:

- In the variable name `blanceBefore`, which should be corrected to `balanceBefore`.
- In the error message string `DCAP: INFINCIENT_BALANCE`, which should be corrected to `INSUFFICIENT_BALANCE`.
- In the constructor parameter name `StarBaseApporve`, which should be corrected to `StarBaseApprove`.
- In the function parameter name `feeReciver`, which should be corrected to `feeReceiver`.
- In the error message string `ACCESS_NENIED`, which should be corrected to `ACCESS_DENIED`.
- The function name `sendTaker` should be corrected to `sendMaker` to accurately reflect its purpose, which is to transfer tokens from the taker to the maker.
- The function name `changeFeeReceiver` should be corrected to `changeFeeRate` to accurately reflect its purpose, which is to change the fee rate rather than the fee receiver.

### Examples

**starbase_swap/contracts/AggregatedSwapRouter.sol:L39-L43**

```
uint blanceBefore = IERC20(tokenOut).balanceOf(receiver);
CallSwapTool(_CallSwapTool).callswap(callSwapAddr, datas, "E"); //SWAP ERROR
require(
    IERC20(tokenOut).balanceOf(receiver) >=
        (blanceBefore + amountOutMin),
```

**starbase_swap/contracts/AggregatedSwapRouter.sol:L58-L62**

```
uint blanceBefore = IERC20(tokenOut).balanceOf(receiver);
CallSwapTool(_CallSwapTool).callswap(callSwapAddr, datas, "E"); //SWAP ERROR
require(
    IERC20(tokenOut).balanceOf(receiver) >=
        (blanceBefore + amountOutMin),
```

### starbase_swap/contracts/AggregatedSwapRouter.sol:L79-L81

```
uint blanceBefore = receiver.balance;
CallSwapTool(_CallSwapTool).callswap(callSwapAddr, datas, "FE"); //SWAP ERROR
require(receiver.balance >= (blanceBefore + amountOutMin), "FOT"); //INSUFFICIENT_OUTPUT_AMOUNT
```

### starbase_swap/contracts/AggregatedSwapRouter.sol:L106-L108

```
uint blanceBefore = receiver.balance;
CallSwapTool(_CallSwapTool).callswap(callSwapAddr, datas, "FE"); //SWAP ERROR
require(receiver.balance >= (blanceBefore + amountOutMin), "FOT"); //INSUFFICIENT_OUTPUT_AMOUNT
```

### starbase_swap/contracts/AggregatedSwapRouter.sol:L120-L124

```
uint blanceBefore = IERC20(tokenOut).balanceOf(receiver);
CallSwapTool(_CallSwapTool).callswap(callSwapAddr, datas, "FRE"); //SWAP ERROR
require(
    IERC20(tokenOut).balanceOf(receiver) >=
        (blanceBefore + amountOutMin),
```

### starbase_swap/contracts/AggregatedSwapRouter.sol:L150-L155

```
uint blanceBefore = IERC20(tokenOut).balanceOf(receiver);
CallSwapTool(_CallSwapTool).callswap(callSwapAddr, datas, "FRE"); //SWAP ERROR
require(
    IERC20(tokenOut).balanceOf(receiver) >=
        (blanceBefore + amountOutMin),
    "FROT"
```

### starbase-limitorder/src/StarBaseDCA.sol:L101

```
require(IERC20(order.inputToken).balanceOf(order.maker) > fee + order.inAmount,"DCAP: INFINCIENT_BALANCE");
```

### starbase-limitorder/src/StarBaseLimitOrder.sol:L113

```
require(IERC20(order.takerToken).balanceOf(msg.sender) > fee + curTakerFillAmount,"SLOP: INFINCIENT_BALANCE");
```

### starbase-limitorder/src/StarBaseApproveProxy.sol:L38-L39

```
constructor(address StarBaseApporve) {
    _StarBase_APPROVE_ = StarBaseApporve;
```

### starbase-limitorder/src/StarBaseLimitOrder.sol:L56-L59

```
function init(address owner, address StarBaseApproveProxy, address feeReciver,uint160 feeRate) external {
    initOwner(owner);
    _StarBase_APPROVE_PROXY_ = StarBaseApproveProxy;
    _FEE_RECEIVER_ = feeReciver;
```

### starbase-limitorder/src/StarBaseDCA.sol:L69-L72

```
function init(address owner, address StarBaseApproveProxy, address feeReciver, uint160 feeRate) external {
    initOwner(owner);
    _StarBase_APPROVE_PROXY_ = StarBaseApproveProxy;
    _FEE_RECEIVER_ = feeReciver;
```

### starbase-limitorder/src/StarBaseDCABot.sol:L80

```
require(msg.sender == _StarBase_DCA_, "ACCESS_NENIED");
```

### starbase-limitorder/src/StarBaseLimitOrderBot.sol:L81

```
require(msg.sender == _StarBase_LIMIT_ORDER_, "ACCESS_NENIED");
```

### starbase-limitorder/src/StarBaseLimitOrder.sol:L114-L127

```
    sendTaker(order, msg.sender,  curTakerFillAmount);
    sendFee(order, msg.sender, _FEE_RECEIVER_, fee);

    emit LimitOrderFilled(order.maker, msg.sender, orderHash, curTakerFillAmount, curMakerFillAmount);
}

function sendTaker(Order calldata order,address from, uint160 curTakerFillAmount) internal{

    //Taker => Maker
    IERC20(order.takerToken).safeTransferFrom(from, order.maker, curTakerFillAmount);
    //Taker => Fee
    //IERC20(order.takerToken).safeTransferFrom(msg.sender, _FEE_RECEIVER_ , fee);

}
```

**starbase-limitorder/src/StarBaseDCA.sol:L186-L194**

```
function changeFeeReceiver(address newFeeReceiver) public onlyOwner {
    _FEE_RECEIVER_ = newFeeReceiver;
    emit ChangeFeeReceiver(newFeeReceiver);
}

function changeFeeReceiver(uint160 feeRate) public onlyOwner {
    _FEE_RATE_ = feeRate;
    emit ChangeFeeRate(feeRate);
}
```

### Recommendation

We recommend correcting typos to improve code clarity and maintain consistency in naming conventions.

## 5.33 Incorrect Function Selector Used in Comment Within `swap` Function ✓ Fixed

| Resolution |
| --- |
| **Update (commit hash `1693a92cc15b71f848e9ceb1d30bfa6d699b0ed3`):** Fixed. |

### Description

In the `swap` function, the comment incorrectly references the function selector `0xa9059cbb` instead of `0x23b872dd`. The correct selector, `0x23b872dd`, corresponds to the `transferFrom(address,address,uint256)` function, whereas `0xa9059cbb` is the selector for `transfer(address,uint256)`. This discrepancy in the comment may cause confusion for developers and auditors reviewing the code.

### Examples

**starbase_swap/contracts/AggregatedSwapRouter.sol:L28**

```
//0xa9059cbb=bytes4(keccak256(bytes('transfer(address,uint256)')));
```

### Recommendation

We recommend updating the comment to accurately reflect the intended function selector and to keep the codebase clear and accurate.

## 5.34 Deflationary Tokens Are Not Supported in `StarBaseDCA` and `StarBaseLimitOrder`

Acknowledged

| Resolution |
| --- |
| Starbase team has acknowledged the finding with the comment: “The first official version of LO and DCA bot will not support the deflationary token.” |

### Description

The `StarBaseLimitOrder` and `StarBaseDCA` contracts do not appear to support rebasing, deflationary, or inflationary tokens, where the balance changes during transfers or over time. The necessary checks include verifying the amount of tokens transferred to contracts before and after the actual transfer.

Specifically, the calculation of `_FILLED_TAKER_AMOUNT_` does not account for deflationary tokens. When dealing with deflationary tokens, the actual amount received after a transfer might be less than the specified amount due to token burn mechanisms or transfer fees. The current logic assumes a 1:1 transfer ratio, which is not the case for deflationary tokens. Even though the `_FILLED_TAKER_AMOUNT_` has been increased by the `curTakerFillAmount` value, the maker will receive fewer tokens than `curTakerFillAmount` after the `sendTaker` function when deflationary tokens are involved.

It’s important to note that the USDT token has a fee mechanism, which is currently disabled. However, the USDT owner can enable it at any time, making it deflationary.

### Examples

**starbase-limitorder/src/StarBaseDCA.sol:L142**

```
IERC20(order.outputToken).safeTransferFrom(msg.sender, order.maker, curTakerFillAmount);
```

**starbase-limitorder/src/StarBaseLimitOrder.sol:L94**

```
_FILLED_TAKER_AMOUNT_[orderHash] = filledTakerAmount + curTakerFillAmount;
```

### Recommendation

We recommend reviewing the protocol logic and adding balance validations if deflationary tokens are expected to be supported.

## 5.35 Potential Precision Loss in Calculations Partially Addressed

### Resolution

Starbase team has acknowledged the issue with the comment: “There is no need to modify, the common problem”.

**Update (commit hash `1693a92cc15b71f848e9ceb1d30bfa6d699b0ed3`):** The issue was partially mitigated by adding a requirement that fees should be higher than zero. The solution excludes this particular edge case, but there can still be rounding errors even with non-zero fees. There is no easy fix to that. Usually, the value of the rounding errors should be very low so most protocols ignore them. It’s just important to be aware of this property of the protocol.

### Description

In the `fillLimitOrder` function, some orders may become unexecutable due to the calculation of the `curMakerFillAmount` variable and precision loss when tokens have different decimal amounts, as well as when the `leftTakerAmount` or `takerFillAmount` are small values. A similar issue occurs with the `fee` variable calculations. If the `curTakerFillAmount` or `inAmount` are small variables, even after being multiplied by the `_FEE_RATE_` variable, the result may still be less than the denominator, leading to a 0 fee, even when the `curTakerFillAmount` is a valid non-zero variable. This creates a potential risk that any order can be filled multiple times for very small `curTakerFillAmount` amounts, executing the order fully without paying any fee to the protocol.

### Examples

**starbase-limitorder/src/StarBaseDCA.sol:L99**

```
uint160 fee = (order.inAmount * _FEE_RATE_) / 10000;
```

**starbase-limitorder/src/StarBaseLimitOrder.sol:L88-L91**

```
curMakerFillAmount = curTakerFillAmount * order.makerAmount / order.takerAmount;
uint160 fee = curTakerFillAmount * _FEE_RATE_ / 10000;

require(curTakerFillAmount > 0 && curMakerFillAmount > 0, "SLOP: ZERO_FILL_INVALID");
```

### Recommendation

We recommend being aware that in some trades, the maker may receive more tokens through multiple small orders instead of one large order due to precision loss in fee calculations.

## 5.36 Multiple `tokenOut` Can’t Be Used in `AggregatedSwapRouter` Contract Acknowledged

### Resolution

The team acknowledged the issue with the following response:

**Pre-transaction Transparency**: Users can fully view the transaction route (datas) and expected output before submitting their swap, ensuring clarity on how their tokens will be exchanged.

**Post-transaction Verification and On-chain Transparency**: Backend systems record the route, input, and output tokens, and all transaction details are fully visible on-chain. Users can independently verify transaction results, including token balances and routes, to ensure no unexpected tokens are generated or transferred.

**Economic Disincentives for Malicious Behavior**: Any deviation from expected behavior would be immediately detectable on-chain, and malicious actions would severely harm the protocol’s reputation, outweighing any potential short-term gains.

### Description

In the functions of `AggregatedSwapRouter`, the `tokenIn` token can be swapped into two different tokens during the swap operation. One of these tokens could match the exact `amountOutMin` and be validated, while the other portion could be sent to a malicious address as a second token, since multiple `tokenOut` is not supported. This could result in the user receiving the minimum required output in one token but losing a portion of their input to a swap into a different token, which lacks validation for the minimum amount out.

### Examples

**starbase_swap/contracts/AggregatedSwapRouter.sol:L19**

```
function swap(
```

**starbase_swap/contracts/AggregatedSwapRouter.sol:L48**

```
function defiSwap(
```

**starbase_swap/contracts/AggregatedSwapRouter.sol:L70**

```
function defiSwapForEth(
```

**starbase_swap/contracts/AggregatedSwapRouter.sol:L87**

```
function swapForEth(
```

**starbase_swap/contracts/AggregatedSwapRouter.sol:L111**

```
function defiSwapFromEth(
```

**starbase_swap/contracts/AggregatedSwapRouter.sol:L142**

```
function swapFromEth(
```

### Recommendation

We recommend avoiding the formation of data where the receiver address is supposed to receive multiple tokens, as the contract lacks sufficient checks for such scenarios. All swaps into multiple tokens should be done in separate transactions.

# Appendix 1 - Files in Scope

This audit covered the following files:

| File | SHA-1 hash |
| --- | --- |
| ./StarBaseDCABot.sol | ff0182c431ae9273e78d852a9f93103f1443c44e |
| ./StarBaseApproveProxy.sol | b7db266e8774566a41b9620981d146de55d288b0 |
| ./StarBaseLimitOrderBot.sol | eb6c9e3b5515792aacf785fe9bb2034d5cae5f5b |
| ./StarBaseDCA.sol | 316803d8a022442e12c85f53ce3ea49646b51d3a |
| ./intf/IERC1271Wallet.sol | 9cccdad46a0021403c6678f63ad0903711302ad2 |
| ./intf/IERC20.sol | a5823f30a1ef7d36c6f4648e53e7f083318dedbd |
| ./intf/IStarBaseApproveProxy.sol | c7c337d3d84ca0f43f239a80c31b68e827d00af8 |
| ./intf/IStarBaseApprove.sol | 387da7c814c0e1f4e3d112d8b70653b0ff7992fb |
| ./lib/SafeERC20.sol | b252129cb230428c24ac0d6898f99bf01593b3cd |
| ./lib/SafeMath.sol | 7917f19a36ee62a34cf7530bb0e9ebef782f1f89 |
| ./lib/InitializableOwnable.sol | 2dc42c67e70b79e64c223298a67680e97c4ef185 |
| ./lib/ArgumentsDecoder.sol | 1790cdbd28f6f3078bc01ac894ae1f84a832c3dd |
| ./StarBaseLimitOrder.sol | 7a4aa4e3aa8b6152af4e7e97164553efa4876eb9 |
| ./StarBaseApprove.sol | d89bebee16b29847cea62bc24c5787420617d964 |

| File | SHA-1 hash |
| --- | --- |
| ./AggregatedSwapRouter.sol | 003b71fb27488ff5c56de30fac4a4e16b9789b12 |

# Appendix 2 - Disclosure

Consensys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via Consensys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any third party in any respect, including regarding the bug-free nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any third party by virtue of publishing these Reports.

### A.2.1 Purpose of Reports

The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of code and only the code we note as being within the scope of our review within this report. Any Solidity code itself presents unique and unquantifiable risks as the Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond specified code that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. In some instances, we may perform penetration testing or infrastructure assessments depending on the scope of the particular engagement.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

### A.2.2 Links to Other Web Sites from This Web Site

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Consensys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that Consensys and CD are not responsible for the content or operation of such Web sites, and that Consensys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that Consensys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. Consensys and CD assumes no responsibility for the use of third-party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

### A.2.3 Timeliness of Content

The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice unless indicated otherwise, by Consensys and CD.