macro

Clanker A-3

Security Audit

July 8, 2025
Version 1.0.0

Presented by 0xMacro

Table of Contents

Introduction

This document includes the results of the security audit for Clanker's smart contract code as found in the section titled ‘Source Code’. The security audit was performed by the Macro security team from June 27, 2025 to July 7, 2025.

The purpose of this audit is to review the source code of certain Clanker Solidity contracts, and provide feedback on the design, architecture, and quality of the source code with an emphasis on validating the correctness and security of the software in its entirety.

Disclaimer: While Macro’s review is comprehensive and has surfaced some changes that should be made to the source code, this audit should not solely be relied upon for security, as no single audit is guaranteed to catch all possible bugs.

Overall Assessment

The following is an aggregation of issues found by the Macro Audit team:

Severity	Count	Acknowledged	Won't Do	Addressed
Medium	3	-	-	3
Low	5	-	-	5
Code Quality	3	-	-	3
Gas Optimization	2	-	-	2

Clanker was quick to respond to these issues.

Specification

Our understanding of the specification was based on the following sources:

- Discussions on Telegram with the Clanker team.

Source Code

The following source code was reviewed during the audit:

contracts repository

- **Commit Hash (Initial):** `2699fdfb8a545682a5ea81d8c7141b98d95d9433`
- **Commit Hash (Final):** `fd573064215a55dbba82b9fc0fbe2c97f54a58b6`

lp-lockers-v0-v1 repository

- **Commit Hash (Initial):** `6a9622fd4e1325aadb884594546bc2562704299c`
- **Commit Hash (Final):** `f021bfdcdc03369e3b104d59d5b48edc18d9f62e`

Specifically, we audited the following contracts within these repositories:

Source Code	SHA256
lp-lockers-v0-v1/src/v0/SingeltonLpLocker.sol	`ff04fe3055e1de6da1de6dd5eebcc30d328f0693f00e376d040c9d5bea08396f`
lp-lockers-v0-v1/src/v1/LpLockerV1Owner.sol	`807f540f783ae5f03860999bdec76b70c8ca5f4a9d0176b720678587748cb634`
contracts/src/mev-modules/ClankerSniperAuctionV0.sol	`154c7c21dfbbf2d5402b2af7eafe2c0fc1da16055ecf0bc99eb2c943b8b21082`
contracts/src/mev-modules/interfaces/IClankerSniperAuctionV0.sol	`a6805d2610d08854bdb2c013b15ebc230d2cc806d69749fab9f136dfaefbbb74`
contracts/src/mev-modules/sniper-utils/ClankerSniperUtilV0.sol	`61ea0b8b43998fe2d45d227dbfe571992ea01020b52b8d019327dba35eb64d74`

Source Code	SHA256
contracts/src/lp-lockers/ClankerLpLockerFeeConversion.sol	306b24da384a0359b132a58f46747bff4b619c5e17aafa7c4c2ea535118aa186

Note: This document contains an audit solely of the Solidity contracts listed above. Specifically, the audit pertains only to the contracts themselves, and does not pertain to any other programs or scripts, including deployment scripts.

Issue Descriptions and Recommendations

Click on an issue to jump to it, or scroll down to see them all.

Security Level Reference

We quantify issues in three parts:

1. The high/medium/low/spec-breaking **impact** of the issue:
    - How bad things can get (for a vulnerability)
    - The significance of an improvement (for a code quality issue)
    - The amount of gas saved (for a gas optimization)
2. The high/medium/low **likelihood** of the issue:
    - How likely is the issue to occur (for a vulnerability)
3. The overall critical/high/medium/low **severity** of the issue.

This third part – the severity level – is a summary of how much consideration the client should give to fixing the issue. We assign severity according to the table of guidelines below:

Severity	Description
(C-x) Critical	We recommend the client **must** fix the issue, no matter what, because not fixing would mean **significant funds/assets WILL be lost.**
(H-x) High	We recommend the client **must** address the issue, no matter what, because not fixing would be very bad, *or* some funds/assets will be lost, *or* the code's behavior is against the provided spec.
(M-x) Medium	We recommend the client to **seriously consider** fixing the issue, as the implications of not fixing the issue are severe enough to impact the project significantly, albiet not in an existential manner.
(L-x) Low	The risk is small, unlikely, or may not relevant to the project in a meaningful way. Whether or not the project wants to develop a fix is up to the goals and needs of the project.
(Q-x) Code Quality	The issue identified does not pose any obvious risk, but fixing could improve overall code quality, on-chain composability, developer ergonomics, or even certain aspects of protocol design.
(I-x) Informational	Warnings and things to keep in mind when operating the protocol. No immediate action required.
(G-x) Gas Optimizations	The presented optimization suggestion would save an amount of gas significant enough, in our opinion, to be worth the development cost of implementing it.

Issue Details

~~M-1~~ Incorrect reward distribution when no swaps

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec	Fixed	High	Low

In the `ClankerLpLockerFeeConversion` contract, `_handleFees()` function contains an off-by-one error in the implementation of reward distribution for the edge case when there are no swaps.

The current code handles specifically the last recipient of the reward to be distributed. However, the index with which it is accessed is off-by-one, as it is unnecessarily decremented. Due to this error, when there are multiple recipients, the reward for the last recipient is sent to the recipient who precedes it. When there is a single recipient, the whole transaction is reverted, as the code will try to execute `0 - 1` when evaluating `distributeLoop - 1` expression in the following code:

```
if (toSwapCount == 0) {
    // if there is no bps to swap, distribute the last reward inclusive of the
    SafeERC20.forceApprove(IERC20(rewardToken), address(feeLocker), tokenToSwap
    feeLocker.storeFees(
        tokenRewardInfo.rewardRecipients[toDistributeIndexes[distributeLoop - 1
        address(rewardToken),
        tokenToSwap
    );
    rewardTokenIsToken0
        ? rewards0[toDistributeIndexes[distributeLoop - 1]] += tokenToSwap
        : rewards1[toDistributeIndexes[distributeLoop - 1]] += tokenToSwap;
}
```

~~M-2~~ Missing validation on the auction bid payee allows paying and winning the auction with unauthorized 3rd party assets

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec	Addressed	High	Low

In the `ClankerSniperAuctionV0` caller may bid and pay in the auction with other users' approved tokens, if there is already preexisting approval granted to the `ClankerSniperAuctionV0` . The reason is that there is no validation of what address auctionData encodes.

```
function _pullPayment(PoolId poolId, bytes calldata auctionData)
    internal
    returns (uint256 paymentAmount)
{
    (address payee) = abi.decode(auctionData, (address));

    // calculate the expected payment for the given gas price
    int256 gasSignal = int256(tx.gasprice) - int256(gasPeg[poolId]);

    // shouldn't be negative
    if (gasSignal < 0) {
        revert GasSignalNegative();
    }

    // calculate the expected payment for the given swap params
    paymentAmount = uint256(gasSignal) * PAYMENT_PER_GAS_UNIT;

    // pull payment from the payee
    SafeERC20.safeTransferFrom(IERC20(weth), payee, address(this), paymentAmour

    emit AuctionWon(poolId, payee, paymentAmount, round[poolId]);
}
```

As a result, if other users/snipers have preauthorized a token transfer to `ClankerSniperAuctionV0` in a separate transaction, for example to have a more

efficient bid transaction, they may become vulnerable, and their assets may be used without their authorization.

Consider clearly emphasizing in the documentation the potential risks for the end users if approving `ClankerSniperAuctionV0` for token transfers.

RESPONSE BY CLANKER

> Noted in documentation, it's expected that snipers can handle this complexity and we provided them an example of how to properly use the contract.

~~M-3~~ Incorrect condition check will perform the collection of rewards during the MEV module delay period

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec	Fixed	High	Low

Note: Discovered by the project team

In the `ClankerLpLockerFeeConversion` contract, the `_mevModuleOperating()` function performs several checks to determine if the MEV module is operating. If it is operating, rewards collection should be skipped; otherwise, the rewards collection process should be performed.

When the MEV module is disabled, rewards collection should be allowed. This is implemented with the first check in `_mevModuleOperating()`

```
// if the mev module is disabled, the swap backs cannot be blocked
if (!IClankerHook(address(_tokenRewards[token].poolKey.hooks)).mevModuleEnablec
    return false;
}
```

There is an additional edge case when mevModule is enabled, but the MEV block delay period has expired, which also represents an inactive mevModule.

```
if (
    poolCreationTimestamp
        + IClankerHook(address(_tokenRewards[token].poolKey.hooks)).MAX_MEV_MOI
        **>** block.timestamp
) {
    return false;
    ...
```

However, this check is implemented incorrectly, since the expired MEV block delay period should be checked in the following way:

```
if (
    poolCreationTimestamp
        + IClankerHook(address(_tokenRewards[token].poolKey.hooks)).MAX_MEV_MOI
        **<=** block.timestamp
) {
    return false;
    ...
```

As a result, reward collection will be incorrectly allowed during the MEV block delay period.

Consider updating condition operator to represent properly system intent.

~~L-1~~ Unspent native asset may get locked in ClankerSniperUtilV0

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec	Fixed	High	Low

In the `ClankerSniperUtilV0` contract, ETH/WETH assets that are transferred to the ClankerSniperUtilV0 but not spent during the transaction, will be locked in the contract, and it won't be possible to withdraw these assets as the contract does not feature corresponding functionality for withdrawing native assets (ETH/WETH).

Combined with the M-2 issue, a malicious attacker may use this approval to win the auction and transfer the assets to the reward recipients.

~~L-2~~ Skip zero amount token transfers within _handleFees()

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec	Fixed	High	Low

In the `ClankerLpLockerFeeConversion` contract, `_handleFees()` performs multiple operations for splitting and distributing corresponding shares of rewards.

Due to calculations involving potentially small amounts of rewards and corresponding small percentages for recipients, there is a high probability that some of the calculated results may end up being 0.

Additionally, some of the unusual ERC20 behaviors include reverting 0 token transfers, which may disrupt the entire swap flow, as reward collection is an integral part of that same flow.

Consider adding validation to prevent attempting zero amount token transfers within `_handleFees()` which may fail and block overall system operation.

~~L-3~~ Incorrect event data in the FeesSwapped event

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec	Fixed	Low	Low

In the `ClankerLpLockerFeeConversion` contract, `_handleFees()` reports incorrect data when emitting `FeesSwapped` event in a specific edge case when there were no swaps, but there was reward distribution.

```
emit FeesSwapped(token, rewardToken, tokenToSwap, tokenToSwapInto, swapAmountOu
```

In this edge case `tokenToSwap` amount would be non-zero value because distribution logic does not deduct from `tokenToSwap` the amount of reward distributed to the last reward recipient.

```
if (toSwapCount == 0) {
    // if there is no bps to swap, distribute the last reward inclusive of the
    SafeERC20.forceApprove(IERC20(rewardToken), address(feeLocker), tokenToSwap
    feeLocker.storeFees(
        tokenRewardInfo.rewardRecipients[toDistributeIndexes[distributeLoop - 1
        address(rewardToken),
        tokenToSwap
    );
    rewardTokenIsToken0
        ? rewards0[toDistributeIndexes[distributeLoop - 1]] += tokenToSwap
        : rewards1[toDistributeIndexes[distributeLoop - 1]] += tokenToSwap;
}
```

This deduction is present for non-last recipients, but it is missing from the code block that handles dust defined within `if (toSwapCount == 0)` branch.

Consider emitting `FeesSwapped` event only within the `if (toSwapCount > 0)` block where fee swaps occur, or update `if toSwapCount == 0` block so that proper event argument data is available when the event is emitted.

~~L-4~~ Wrong event data due to incorrect order of event parameters

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec	Fixed	Low	Low

The event declaration and event emission are not aligned within `SingletonLpLocker` and `LpLockerV1Owner` contracts. `ClaimedFee` event has `totalAmount1` first, followed by the `totalAmount0` event argument. However, when the event is emitted, the arguments provided are reversed, first `amount0` followed by the `amount1` .

```
event ClaimedFees(
    address indexed token0,
    address indexed token1,
    uint256 amount0,
    uint256 amount1,
    uint256 **totalAmount1**,
    uint256 **totalAmount0**
);

emit ClaimedFees(token0, token1, recipientFee0, recipientFee1, **amount0**, **a
```

Consider updating the event declaration to match the order of arguments in event emission.

~~L-5~~ Remove unsupported native assets handling

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec	Fixed	Low	Low

In the `ClankerLpLockerFeeConversion` contract, `_uniSwapLocked()` function contains code that incorrectly indicates that it is meant to support native assets handling.

```
universalRouter.execute{
    value: Currency.unwrap(poolKey.currency0) == address(0) ? amountIn : 0
}(commands, inputs, block.timestamp);
```

Consider removing this code, as the specification does not support native assets.

~~Q-1~~ LpLockerV1Owner should use SafeERC20 for token transfers

TOPIC	STATUS	QUALITY IMPACT
Spec	Fixed	Low

In the wrapper contract `LpLockerV1Owner`, tokens are transferred within `_collectFeesPostRelease()` function without checking the return value of the operation, potentially resulting in the success of the overall transaction even when the token transfer has failed, which would violate core system invariants and result in unexpected behavior.

```
...
feeToken0.transfer(creatorFeeRecipient, recipientFee0);
feeToken1.transfer(creatorFeeRecipient, recipientFee1);

feeToken0.transfer(protocolFeeRecipient, protocolFee0);
feeToken1.transfer(protocolFeeRecipient, protocolFee1);

emit ClaimedFees(
    creatorFeeRecipient, token0, token1, recipientFee0, recipientFee1, amount0,
);
```

Consider using SafeERC20 or checking the operation return value to ensure that the token transfer has been successfully completed in all cases, even when underlying transfer functions do not revert but return `false`.

~~Q-2~~ Inconsistent determination of currency order

TOPIC	STATUS	QUALITY IMPACT
Spec	Fixed	Low

In `ClankerLpLockerFeeConversion` contract, determination of currency order in `_uniSwapUnlocked()` and `_uniSwapLocked()` is performed differently.

```
\\ In 1.
bool zeroForOne = tokenIn < tokenOut;

\\ In 2
bool tokenInIsToken0 = Currency.unwrap(poolKey.currency0) == tokenIn;
```

Consider updating the implementation in the `_uniSwapUnlocked()` to match the approach used across other functions in the same contract.

~~Q-3~~ Unused code can be removed

TOPIC	STATUS	QUALITY IMPACT
Spec	Fixed	Low

Unused code is present in several contracts that are in the scope of the audit:

- within `SingletonLpLocker` contract:

```
error NotAuthorized();
event CreatorFeeRecipientUpdated(
    address indexed previousRecipient, address indexed newRecipient
);
event TokenReceived(address indexed from, uint256 id);
```

- within `LpLockerV1Owner` contract:

```
error NotAuthorized();
event TokenReceived(address indexed from, uint256 id);
```

- in `ClankerSniperAuctionV0` contract:

```
import {console} from "forge-std/console.sol";
```

- in `IClankerSniperAuctionV0` contract:

```
error InvalidPayment();
```

- In `ClankerLpLockerFeeConversion` contract:

```
// import
import {TransientStateLibrary} from "@uniswap/v4-core/src/libraries/Transie
// usage
using TransientStateLibrary for IPoolManager;
```

- In `ClankerLpLockerFeeConversion` contract consider replacing

```
// this
zeroForOne: tokenInIsToken0 ? true : false, // swapping tokenIn -> tokenOut
// with
zeroForOne: tokenInIsToken0 // swapping tokenIn -> tokenOut
```

RESPONSE BY CLANKER

> Additional changes performed in the following commits: https://github.com/clanker-devco/lp-lockers-v0-v1/commit/3882ad9fdb85a35c37772995eef90d5a19676105 and https://github.com/clanker-devco/contracts/commit/fd573064215a55dbba82b9fc0fbe2c97f54a58b6

~~G-1~~ Reduce the number of forceApprove operations

TOPIC	STATUS	GAS SAVINGS
Gas Optimization	Fixed	Medium

1. In the `ClankerLpLockerFeeConversion` contract, `_handleFees()` contains two loops in which each iteration executes `forceApprove()` and authorises the same contract `feeLocker`.

   Consider using a single forceApprove per loop with the sum of all token transfers performed within the loop.

2. Similarly, in the `ClankerSniperAuctionV0` there are also multiple approve statements within the loop.

```
 // distribute the rewards
for (uint256 i = 0; i < tokenRewardInfo.rewardBps.length; i++) {
    IERC20(weth).approve(address(feeLocker), rewardsSplit[i]);
    feeLocker.storeFees(tokenRewardInfo.rewardRecipients[i], weth, rewardsS
}
```

   replace with

```
 // distribute the rewards
IERC20(weth).approve(address(feeLocker), lpPayment);
for (uint256 i = 0; i < tokenRewardInfo.rewardBps.length; i++) {
```

```
            feeLocker.storeFees(tokenRewardInfo.rewardRecipients[i], weth, rewardsS
        }
```

RESPONSE BY CLANKER

> Additional changes performed in the following commit: https://github.com/clanker-devco/contracts/commit/fd573064215a55dbba82b9fc0fbe2c97f54a58b6.

~~G-2~~ Avoid repeating calculation operations

TOPIC	STATUS	GAS SAVINGS
Gas Optimization	Fixed	Low

In the `ClankerLpLockerFeeConversion` contract, `_handleFees()` contains the following piece of code, which performs the same calculation three times.

```
SafeERC20.forceApprove(
    IERC20(tokenToSwapInto), address(feeLocker), **swapAmountOut - swapDistri
);
feeLocker.storeFees(
    tokenRewardInfo.rewardRecipients[toSwapIndexes[toSwapCount - 1]],
    address(tokenToSwapInto),
    **swapAmountOut - swapDistributed**
);


rewardTokenIsToken0
    ? rewards1[toSwapIndexes[toSwapCount - 1]] += **swapAmountOut - swapDistr
    : rewards0[toSwapIndexes[toSwapCount - 1]] += **swapAmountOut - swapDistr
```

Consider memoizing the calculation result using a local variable and reusing it instead of repeating the calculation.

Disclaimer

Macro makes no warranties, either express, implied, statutory, or otherwise, with respect to the services or deliverables provided in this report, and Macro specifically disclaims all implied warranties of merchantability, fitness for a particular purpose, noninfringement and those arising from a course of dealing, usage or trade with respect thereto, and all such warranties are hereby excluded to the fullest extent permitted by law.

Macro will not be liable for any lost profits, business, contracts, revenue, goodwill, production, anticipated savings, loss of data, or costs of procurement of substitute goods or services or for any claim or demand by any other party. In no event will Macro be liable for consequential, incidental, special, indirect, or exemplary damages arising out of this agreement or any work statement, however caused and (to the fullest extent permitted by law) under any theory of liability (including negligence), even if Macro has been advised of the possibility of such damages.

The scope of this report and review is limited to a review of only the code presented by the Clanker team and only the source code Macro notes as being within the scope of Macro's review within this report. This report does not include an audit of the deployment scripts used to deploy the Solidity contracts in the repository corresponding to this audit. 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. In this report you may through hypertext or other computer links, gain access to websites operated by persons other than Macro. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such websites' owners. You agree that Macro is not responsible for the content or operation of such websites, and that Macro shall have no liability to your or any other person or entity for the use of third party websites. Macro assumes no responsibility for the use of third party software and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.