macro

Clanker A-2

Security Audit

June 13, 2025
Version 1.0.0

Presented by 0xMacro

Table of Contents

- Introduction
- Overall Assessment
- Specification
- Source Code
- Issue Descriptions and Recommendations
- Security Levels Reference
- Issue Details
- Disclaimer

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 May 19, 2025 to June 12, 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	1	-	-	1
Low	1	-	-	1
Code Quality	13	1	-	12
Informational	3	-	-	-

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:

- **Repository:** contracts
- **Commit Hash:** `f5e83b922844a7fb30f442f6effcd7d6a13561b4`

Specifically, we audited the following contracts within the repository:

Source Code	SHA256
src/Clanker.sol	`0947d31c19c2e403883ff9b0e0dd8d5758f5fc3ada26fdcd5e9e71ee9bd2e3a9`
src/ClankerFeeLocker.sol	`1b9b3dac9cbf425053f34d6032e8924a0bad4e079fd0e5b4ccd754cc6f2c0507`
src/ClankerLpLockerMultiple.sol	`3d9244a291c05150bc7946898d6a81a2e9bda132af3a1abd487f81b227008459`
src/ClankerToken.sol	`2cf930dfa592b3a6fc4166797c8f9cadc2e79691b9c5ba898f8b3a66e2d97646`
src/extensions/ClankerAirdrop.sol	`f00c5451ee9019cbabd301fa282edad59d1d07e36f291446ebd72feec931822f`
src/extensions/ClankerPresaleEthToCreator.sol	`e82b7971e0ac003fb0d75ce5391d65d654e3022acd7f4b5b4862bed44a97d578`
src/extensions/ClankerUniv4EthDevBuy.sol	`205a72d980037b282675e0a3bf9094de1deb503cc79610e5144154097278f150`
src/extensions/ClankerVault.sol	`47a9c9ba746444d6f4cd40debe9bcc91daa63852bb6307a8debb1d4e118cc939`
src/extensions/interfaces/IClankerAirdrop.sol	`3ef48b0be8d2643c4b61ffb6394aea544308a347b3eb80aeefb717d18ef13cb9`

Source Code	SHA256
src/extensions/interfaces/IClankerPresaleEthToCreator.sol	7f5be72652cd0f6640bc5e6fc35da9dfe0749a93948a521a183909cf62b50c41
src/extensions/interfaces/IClankerUniv4EthDevBuy.sol	b97e73497a30a9aba18f6017c9d683fdf201c121e084b54bcc7d546e782a15e1
src/extensions/interfaces/IClankerVault.sol	30eee815a6f574ce1e8a258b3c70fbb6f65825cf8cbe4a801e9a225cfafd29be
src/hooks/ClankerHook.sol	f97ba769c33847e09eea0d0767db4315f095aa2a58842fe371b4aaa28cabe7a8
src/hooks/ClankerHookDynamicFee.sol	2f32974da48ad5353f7032c90f480b48706a8b81475bc7122148980d1be44665
src/hooks/ClankerHookStaticFee.sol	ac1ec39055d02c25b875b1872ab8c90c6227e40d8c649088fc47117c4fd3d651
src/hooks/interfaces/IClankerHookDynamicFee.sol	8b561ea49d0f109854600eb07c9001d74947718d19210407353dc372c10fdcd0
src/hooks/interfaces/IClankerHookStaticFee.sol	f72496f7d50e5eaf07a857943e010da5df36e5cb109f1d2ad068da8d180a9e4a
src/interfaces/IClanker.sol	2b9f5cd46b6f6294f580da5fbba00b0174ca54c7d978bf1057475a0eadd03a1c
src/interfaces/IClankerExtension.sol	2b3eb89e40c86fb0986a5ad85876981e3e87205e7e3d696a522fdb778633f51e
src/interfaces/IClankerFeeLocker.sol	9c3fe07b76cdb7d0b525aa077c8145b6cf80b95f8b055a745577dce2eaff6796
src/interfaces/IClankerHook.sol	813898debf30d2bcaebe049448c0a8fc3eade801678164fc9c93373bd98ddf55
src/interfaces/IClankerLPLocker.sol	6a767b462a0fb04019f820f747f03a01dc0e4fe271ca88a0128d7a80945be2d4

Source Code	SHA256
src/interfaces/IClankerLpLockerMultiple.sol	b3027ae8aeeaa7d7b97d509ae7b1c9c666dd4c8ef913e00aaad4149ef1471ae1
src/interfaces/IClankerMevModule.sol	bf2f969bd46fb1c575ff21751db6deb664c0c1248b70a4c7c881324b97cd88e6
src/interfaces/IOwnerAdmins.sol	8727d451f7a76b3c51cdf4efabbee6d5a3e72a668a0ebe62601e2c527c85e67a
src/mev-modules/ClankerMevBlockDelay.sol	2003122acb446bec4b11bec57a6dbc1aca27189f57f09510ac0a3bdd6b26a768
src/utils/ClankerDeployer.sol	a8f58bb2937a7adae57487e6f3c6457627ede84dd7a0331561c087f30cedc37d
src/utils/OwnerAdmins.sol	362ba39eff554f8a0f45c6bb9fd8e3d9d69f8d843a253c40999d1019fc52fddb

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.

~~M-1~~ Unsafe transfers of ERC20 tokens

~~L-1~~ Missing validation for the implicit requirement that tickLower config values are sorted

Q-1 ClankerMevBlockDelay can be used without initialization

~~Q-2~~ ClankerAirdrop.amountAvailableToClaim()

~~Q-3~~ Inconsistent msg.value validation

~~Q-4~~ Unnecessary cast in _setFee()

~~Q-5~~ Constant value does not match the natspec description

~~Q-6~~ Unused code

~~Q-7~~ Incomplete interface definitions

~~Q-8~~ Improve Event definitions

~~Q-9~~ Update buyIntoPresale implementation to follow CEI pattern

~~Q-10~~ PresaleId set twice unnecessarily

~~Q-11~~ Improve the enabledLockers mapping variable naming

~~Q-12~~ Unnecessary initialize() function in Clanker contract

~~Q-13~~ Make _tokenRewards variable internal

I-1 ClankerAirdrop does not provide guarantees that all allocations set in MerkleTree would be claimable

I-1 ClankerAirdrop does not provide guarantees that all extension supply provided to it may be claimable

I-3 ClankerAirdrop will allow claiming the largest allocation in case the receiver has multiple allocations

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~~ Unsafe transfers of ERC20 tokens

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec	Fixed	High	Low

In multiple system contracts, such as `Clanker`, `ClankerAirdrop`, `ClankerUniv4EthDevBuy`, and `ClankerPresaleEthToCreator`, tokens are transferred without checking the return value of the operation, potentially resulting in the success of the overall transaction even in the case when the token transfer has failed, which would violate core system invariants and result in unexpected behavior.

- `Clanker.claimTeamFees()`

```
IERC20(token).transfer(teamFeeRecipient, balance);
```

- `ClankerAirdrop.receiveTokens()`

```
// pull in token
IERC20(token).transferFrom(msg.sender, address(this), extensionSupply);
```

- `ClankerAirdrop.claim()`

```
// transfer tokens
IERC20(token).transfer(recipient, claimableAmount);
```

- `ClankerUniv4EthDevBuy.receiveTokens()`

```
        // transfer the token to the recipient
        IERC20(token).transfer(devBuyData.recipient, tokenAmount);
```

- `ClankerPresaleEthToCreator.claimTokens()`

```
        // send tokens to user
        IERC20(presale.deployedToken).transfer(msg.sender, tokenAmount);
```

- `ClankerPresaleEthToCreator.receiveTokens()`

```
        // pull in token supply
        IERC20(token).transferFrom(msg.sender, address(this), extensionSupply);
```

Remediations to Consider

- 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` .

RESPONSE BY CLANKER

> Fixed for Clanker.claimTeamFees(), for other places the only token being transferred are ClankerToken instances which fail if the transfers fail.

~~L-1~~ Missing validation for the implicit requirement that tickLower config values are sorted

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec	Fixed	Low	Low

In the `ClankerLpLockerMultiple` contract, `placeLiquidity()` and `_mintLiquidity()` functions are responsible for configuring corresponding liquidity positions in the just-

created pool.

```
    int24 startingTick =
        token0IsClanker ? lockerConfig.tickLower[0] : -lockerConfig.tickLower[0];

    for (uint256 i = 0; i < lockerConfig.tickLower.length; i++) {
        // add mint action
        actions = abi.encodePacked(actions, uint8(Actions.MINT_POSITION));

        // determine token amount for this position
        uint256 tokenAmount = poolSupply * lockerConfig.positionBps[i] / BASIS_PO
        uint256 amount0 = token0IsClanker ? tokenAmount : 0;
        uint256 amount1 = token0IsClanker ? 0 : tokenAmount;

        // determine tick bounds for this position
        int24 tickLower_ =
            token0IsClanker ? lockerConfig.tickLower[i] : -lockerConfig.tickLower
        int24 tickUpper_ =
            token0IsClanker ? lockerConfig.tickUpper[i] : -lockerConfig.tickUpper
        int24 tickLower = token0IsClanker ? tickLower_ : tickUpper_;
        int24 tickUpper = token0IsClanker ? tickUpper_ : tickLower_;
        uint160 lowerSqrtPrice = TickMath.getSqrtPriceAtTick(tickLower);
        uint160 upperSqrtPrice = TickMath.getSqrtPriceAtTick(tickUpper);

        // determine liquidity amount
        uint256 liquidity = LiquidityAmounts.getLiquidityForAmounts(
            startingTick.getSqrtPriceAtTick(), lowerSqrtPrice, upperSqrtPrice, am
        );
    ...
```

When `getLiquidityForAmounts()` is called, the `startingTick` value is based on the value of the first element in `the tickLower[]` array. If values are not sorted, the startingTick value would not be correct, and overall, the operation would result in unexpected reverts.

Consider adding corresponding validation to more explicitly report this configuration error.

```
  for (uint256 i = 1; i < lockerConfig.tickLower.length; i++) {
      if (lockerConfig.tickLower[i] < lockerConfig.tickLower[i-1]) {
          revert TicksNotSorted();
```

```
    }
}
```

RESPONSE BY CLANKER

> Fixed by using the starting price as the lower tick.

Q-1 ClankerMevBlockDelay can be used without initialization

TOPIC	STATUS	QUALITY IMPACT
Spec	Acknowledged	Low

In `ClankerMevBlockDelay`, the `beforeSwap()` function can be executed without first calling `initialize()`.

Currently, `ClankerHook` properly calls `initialize()` first, before invoking `beforeSwap()`. However, initialization is not enforced within the `ClankerMevBlockDelay` contract itself.

Consider updating the implementation to ensure that initialization is completed before executing any operations that rely on it.

RESPONSE BY CLANKER

> This is intentional to allow extensions to perform pool actions before the MevModule potentially locks the pool

---

~~Q-2~~ ClankerAirdrop.amountAvailableToClaim()

TOPIC	STATUS	QUALITY IMPACT
Spec	Fixed	Low

In the `ClankerAirdrop`, `amountAvailableToClaim()` may return a valid value (such as 0) even for tokens that do not have an airdrop yet configured.

Consider generating an error if the airdrop for the token is not configured.

~~Q-3~~ Inconsistent msg.value validation

TOPIC	STATUS	QUALITY IMPACT
Best practices	Fixed	Low

In `ClankerVault`, the check validates that the config value for `msg.value` is not set. However, it does not check that the actual `msg.value` is also not 0.

```
// ensure that the msgValue is zero
if (deploymentConfig.extensionConfigs[extensionIndex].msgValue != 0) {
    revert IClankerExtension.InvalidMsgValue();
}
```

On the other hand, in `ClankerAirdrop.receiveTokens()`, both the config value and the actual `msg.value` are validated,

```
// ensure that the msgValue is zero
if (deploymentConfig.extensionConfigs[extensionIndex].msgValue != 0 || msg.valu
  revert IClankerExtension.InvalidMsgValue();
}
```

Similar implementation is also present in `ClankerPresaleEthToCreator.receiveTokens()`

```
// ensure that the msgValue is zero
if (deploymentConfig.extensionConfigs[extensionIndex].msgValue != 0 || msg.valu
    revert IClankerExtension.InvalidMsgValue();
}
```

Similarly, in `ClankerUniv4EthDevBuy.receiveTokens()`, both the config and actual value are checked.

```
// ensure that the msgValue matches what was requested and is not zero
if (
    deploymentConfig.extensionConfigs[extensionIndex].msgValue != msg.value
        || deploymentConfig.extensionConfigs[extensionIndex].msgValue == 0
) {
    revert IClankerExtension.InvalidMsgValue();
}
```

In addition, the `Clanker` contract (factory) is the only caller of these functions, and the only place where these functions are triggered is `_triggerExtensions()`.

```
// trigger the extension
IClankerExtension(deploymentConfig.extensionConfigs[i].extension).receiveTokens
    value: deploymentConfig.extensionConfigs[i].msgValue
}(deploymentConfig, poolKey, token, extensionSupply, i);
```

Based on how these functions are invoked, `msg.value` is guaranteed to be equal to the `msgValue` config value, and therefore, checking only one variable is enough. Just consider making checks consistent.

~~Q-4~~ Unnecessary cast in _setFee()

TOPIC	STATUS	QUALITY IMPACT
Best practices	Fixed	Low

- In the `ClankerHookDynamicFee`, `_setFee()` function contains unnecessary cast of `lpFee`, which is `uint24` value, to `uint24`.

```
IPoolManager(poolManager).updateDynamicLPFee(poolKey, uint24(lpFee));
```

- In the `ClankerHookDynamicFee`, an unnecessary cast is performed in `_getLpFee()` after the type update of `feeControlNumerator` (from uint24 to uint256).

```
uint256 variableFee = uint256(poolConfigVars_.feeControlNumerator) * (volA
 / FEE_CONTROL_DENOMINATOR;
```

Consider removing unnecessary casts.

~~Q-5~~ Constant value does not match the natspec description

TOPIC	STATUS	QUALITY IMPACT
Best practices	Fixed	Low

In the `ClankerHookDynamicFee`, `MIN_BASE_FEE` is set to 2500. Correspondingly, the natspec comment describing this constant indicates this value represents 0.025% of the unit represented in 1_000_000 basis points.

```
uint24 public constant MIN_BASE_FEE = 2500; // 0.025%;
```

However, comment and value do not match as the set value actually represents 0.25% and not 0.025% of the FEE unit.

Consider updating the constant value or changing the corresponding code comment.

~~Q-6~~ Unused code

TOPIC	STATUS	QUALITY IMPACT
Best practices	Fixed	Low

- Unused imports In `ClankerHookStaticFee` .

```
import {BeforeSwapDelta} from "@uniswap/v4-core/src/types/BeforeSwapDelta.
import {console} from "forge-std/console.sol";
```

- Duplicate import in `ClankerUniv4EthDevBuy` .

```
import {PoolKey} from "@uniswap/v4-core/src/types/PoolKey.sol";
import {PoolKey} from "@uniswap/v4-core/src/types/PoolKey.sol";
```

- The following errors are defined in `IClankerFeeLocker` but are never used.

```
error ClaimAmountTooHigh();
error InvalidRecipient();
```

- `IClankerExtension` is unnecessary in the contract inheritance declaration within `ClankerPresaleEthToCreator` , as it is already present in the `IClankerPresaleEthToCreator` .

~~Q-7~~ Incomplete interface definitions

TOPIC	STATUS	QUALITY IMPACT
Best practices	Fixed	Low

Multiple interfaces across the codebase do not contain all the public methods of the underlying contracts.

- `IClankerHookStaticFee` does not contain a function declaration for functions corresponding to the public variables, such as accessors for `clankerFee` and `pairedFee` mappings.
- `IClankerHookDynamicFee` does not contain a function declaration for functions corresponding to the public variables, such as accessors for `poolFeeVars` and `poolConfigVars` mappings or any of the public constants.
- Missing function declarations in `IClankerMevModule` interface for public variables `poolUnlockTime` and `blockDelay` .
- Missing function declarations in `IClanker` interface for public variables.
- Multiple important public and contract functions are not declared in the `IClankerPresaleEthToCreator` interface.
- The same applies to other interfaces too.

RESPONSE BY CLANKER

> Added missing definitions for `IClankerHookStaticFee` and `IClankerHookDynamicFee` .

~~Q-8~~ Improve Event definitions

TOPIC	STATUS	QUALITY IMPACT
Best practices	Fixed	Low

- `ClankerPresaleEthToCreator` - does not feature any events. Currently, events are missing for important state changes, such as changes in presale status.
- `IClankerLpLockerMultiple` - missing indexed attribute for Events
  - `ClaimedRewards` - token
  - `RewardRecipientUpdated` - token, oldRecipient, newRecipient
  - `RewardAdminUpdated` - token, oldAdmin, newAdmin
- `ClankerAirdrop` - consider emitting an AirdropFullyClaimed(address token) event in the claim function when airdrop.totalClaimed becomes equal to airdrop.totalSupply. Off-chain systems might find it useful to know when an airdrop is fully depleted.

RESPONSE BY CLANKER

> Fixed `IClankerLpLockerMultiple` .

~~Q-9~~ Update buyIntoPresale implementation to follow CEI pattern

TOPIC	STATUS	QUALITY IMPACT
Best practices	Fixed	Low

In the `ClankerPresaleEthToCreator` contract, the `buyIntoPresale()` function implementation currently does not follow the Check-Effects-Interaction pattern, as an external call to refund eth to `msg.sender` is performed before important state updates.

```
// refund excess eth
if (msg.value > ethToUse) {
    // refund excess eth
    payable(msg.sender).transfer(msg.value - ethToUse);
}

// record a user's eth contribution
presaleBuys[presaleId][msg.sender] += ethToUse;

// update eth raised
presale.ethRaised += ethToUse;
```

The risk of reentrancy in this case is limited by using a `transfer` that has a limited gas stipend. However, best practice is to avoid having potentially reentrant code in case of future code updates.

Consider updating `buyIntoPresale()` implementation and moving the refund piece of logic to the end of the function.

~~Q-10~~ PresaleId set twice unnecessarily

TOPIC	STATUS	QUALITY IMPACT
Best practices	Fixed	Low

In the `ClankerPresaleEthToCreator`, the `startPresale()` function encodes the `presaleId` value and sets it on the corresponding extension config from the stored deploymentConfig. Immediately after this, the deployment config is stored as part of the overall presale struct record.

```
deploymentConfig.extensionConfigs[deploymentConfig.extensionConfigs.length - 1]
  .extensionData = abi.encode(presaleId);
```

In addition, in the `endPresale()` function, where the presale struct record is loaded, presaleId is again set to the same value.

```
// encode presale id into extension config data
presale.deploymentConfig.extensionConfigs[presale.deploymentConfig.extensionCon
  - 1].extensionData = abi.encode(presaleId);
```

Consider removing the unnecessary `presaleId` update in the `endPresale()` method.

~~Q-11~~ Improve the enabledLockers mapping variable naming

TOPIC	STATUS	QUALITY IMPACT
Best practices	Fixed	Low

In the `Clanker` contract, the `enabledLockers` mapping variable is defined in the following way:

```
mapping(address locker => mapping(address pool => bool enabled)) public enabled
```

However, when used, it isn't clear if the `pool` variable within the mapping represents the actual Uniswap V4 pool or the associated `hook`.

In `setLocker()`, this mapping is used in the following way.

```
enabledLockers[locker][pool] = enabled;
```

While in `_initializeLiquidity()` it is used with a different variable naming, notice `poolConfig.hook`

```
enabledLockers[lockerConfig.locker][poolConfig.hook]
```

Consider revising the definition of this mapping and using it consistently throughout the contract.

~~Q-12~~ Unnecessary initialize() function in Clanker contract

TOPIC	STATUS	QUALITY IMPACT
Best practices	Fixed	Low

In the `Clanker` contract, the `initialize()` function's only role is to be a wrapper for calling `setDeprecated()` and `setTeamFeeRecipient()` underlying methods. Additionally, following recent changes and the removal of the locker check, the `initialize()` function may be called multiple times, which is not expected.

Consider removing the `initialize()` function and relying on direct calls to the underlying `setDeprecated()` and `setTeamFeeRecipient()` functions.

~~Q-13~~ Make _tokenRewards variable internal

TOPIC	STATUS	QUALITY IMPACT
Best practices	Fixed	Low

In the `ClankerLpLockerMultiple`, the `tokenRewards` public variable has been renamed to `_tokenRewards`. Additionally, the custom `tokenRewards()` function has been introduced. However, since public variables obtain automatically generated getters that means `_tokenRewards()` function is also present which might be confusing and unnecessary.

Consider making the `_tokenRewards` variable internal to prevent direct external access to it.

I-1 ClankerAirdrop does not provide guarantees that all allocations set in MerkleTree would be claimable

TOPIC
Spec

IMPACT
Informational ✳

In the `ClankerAirdrop` contract, there is no validation that the sum of allocations, which are set as leaves of the MerkleTree, is equal to the configured extension supply. If this supply is smaller than the sum of allocations, `ClankerAirdrop` will act according to the First In First Out (FIFO) principle, and the first users to claim will receive tokens, while later users may not be able to claim the allocation in full or partially.

RESPONSE BY CLANKER

> Acknowledged - Noted in documentation.

I-1 ClankerAirdrop does not provide guarantees that all extension supply provided to it may be claimable

TOPIC	IMPACT
Spec	Informational ✳

In the `ClankerAirdrop` contract, there is no validation that sum of allocations, that are set as leafs of the MerkleTree, is equal to the configured extension supply. If this supply exceeds the sum of allocations, `ClankerAirdrop` will lock the surplus extension supply.

RESPONSE BY CLANKER

> Acknowledged - Noted in documentation.

I-3 ClankerAirdrop will allow claiming the largest allocation in case the receiver has multiple allocations

TOPIC	IMPACT
Spec	Informational ✳

In the `ClankerAirdrop` contract, in situations where MerkleTree has multiple entries for the user with a particular address, the user will be able to claim the maximum out of all his allocations.

RESPONSE BY CLANKER

> Acknowledged - Noted in documentation.

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.