Clanker Contracts

Security Review

Cantina Managed review by:

Noah Marconi, Lead Security Researcher
Cccz, Security Researcher

June 11, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Clanker is a set of smart contracts that create token markets which reward token creators.

From May 20th to Jun 4th the Cantina team conducted a review of clanker-contracts on commit hash f5e83b92. The team identified a total of **26** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	6	5	1
Low Risk	8	7	1
Gas Optimizations	4	1	3
Informational	8	3	5
Total	**26**	**16**	**10**

3 Findings

3.1 Medium Risk

3.1.1 Precision loss in presale may result in asset loss

Severity: Medium Risk

Context: ClankerPresaleEthToCreator.sol#L192-L205

Description: In presale extension, when the maximum raised amount is not reached, the protocol adjusts the extensionBps that are distributed to presale:

```
if (presale.ethRaised != presale.maxEthGoal) {
    uint256 originalBps = uint256(
        presale.deploymentConfig.extensionConfigs[presale
            .deploymentConfig
            .extensionConfigs
            .length - 1].extensionBps
    );

    uint256 newBps = (originalBps * presale.ethRaised) / presale.maxEthGoal;
    presale.deploymentConfig.extensionConfigs[presale
        .deploymentConfig
        .extensionConfigs
        .length - 1].extensionBps = uint16(newBps);
}
```

Since the extensionBps base is 10000 and TOKEN_SUPPLY is 100_000_000_000e18, so precision loss can result in a large token loss.

```
uint256 public constant TOKEN_SUPPLY = 100_000_000_000e18; // 100b with 18 decimals
uint256 public constant BPS = 10_000;
```

Consider extensionBps == 10%, extensionSupply is 10_000_000_000e18, the precision loss would result in a maximum token loss of `1_000_000e18`.

Proof of Concept: The following proof of concept shows that raising `99_999` or `99_990` ether gets the same amount of tokens.

```
function test_poc() public {
    vm.deal(alice, 200_000 ether);
    // start presale
    vm.prank(clankerTeamEOA);
    uint256 presaleId = presaleExtension.startPresale(
        baseDeploymentConfig, 1000 ether, 100_000 ether, 1 days, bob, 0 days, 0 days
    );

    // buy into presale
    vm.prank(alice);
    presaleExtension.buyIntoPresale{value: 99_999 ether}(presaleId);

    vm.warp(block.timestamp + 1 days);
    // trigger presale end
    vm.prank(clankerTeamEOA);
    address token = presaleExtension.endPresale(presaleId, bytes32(0));

    console.log(IERC20(token).balanceOf(address(presaleExtension))); // 49990000000e18

    vm.prank(clankerTeamEOA);
    presaleId = presaleExtension.startPresale(
        baseDeploymentConfig, 100 ether, 100_000 ether, 1 days, bob, 0 days, 0 days
    );

    // buy into presale
    vm.prank(alice);
    presaleExtension.buyIntoPresale{value: 99_990 ether}(presaleId);

    vm.warp(block.timestamp + 2 days);
    // trigger presale end
    vm.prank(clankerTeamEOA);
    token = presaleExtension.endPresale(presaleId, bytes32("1"));

    console.log(IERC20(token).balanceOf(address(presaleExtension))); // 49990000000e18
}
```

Recommendation: Since changing the precision involves changes to other components, the following options are provided for fixes beyond precision.

- Option 1: Refunds in excess of precision during purchase, but this would be a bit of a blow to the presale progress.
- Option 2: Refund proportionally in excess of precision after the presale is ended, which would require additional implementation.

Clanker: Acknowledged.

Cantina Managed: Acknowledged.

3.1.2 Incorrect protocolFee adjustment in `_beforeSwap()`

Severity: Medium Risk

Context: ClankerHook.sol#L289-L313

Description: ClankerHook charges protocol fees when users swap. In order to make exactIn and `exactOut` charge the same fee for the same direction, ClankerHook adjusts the protocol fee in `_beforeSwap()` to make them consistent.

For example, when a user swaps WETH for Clanker, when taking `exactOut`, the protocol calculates the protocol fee based on the amount of WETH gotten after the swap. When taking `exactIn`, the protocol calculates the protocol fee based on the amount of WETH provided by the user before the swap. Since the amount of WETH before and after the swap is different, the protocol slightly decreases the protocol fee in `_beforeSwap()` in order to make the fees charged consistent. But the problem here is that the protocol uses the incorrect formula. It should use `protocolFee / (1 - protocolFee)` instead of `1 * (1 + protocolFee)`.

```
protocolFee =
    uint24(uint256(protocolFee) * (10_000 - (uint256(protocolFee) / 100)) / 10_000);
```

Similarly, in the other path, it should use `protocolFee / (1+protocolFee)` instead of `1 * (1-protocolFee)`.

```
protocolFee =
    uint24(uint256(protocolFee) * (10_000 + (uint256(protocolFee) / 100)) / 10_000);
```

Proof of Concept: In the following test, making the LPFee 50% and the protocol fee 10%. For `exactIn`, the protocol fee should be 10%/(1 - 10%) = 0.0909, while the current calculation is 10% * (1 - 10%) = 0.09, i.e. the fee is calculated incorrectly.

```
function test_fee() public {
    baseDeploymentConfig.poolConfig.poolData = abi.encode(
        IClankerHookStaticFee.PoolStaticConfigVars({clankerFee: 500_000, pairedFee: 500_000}) // 50% ,
        ↪  protocolFee = 10%
    );
    (bytes32 tokenLowerSalt,) = Utils.generateSalt(
        address(clanker),
        baseDeploymentConfig.tokenConfig,
        baseDeploymentConfig.poolConfig.pairedToken,
        false
    );
    baseDeploymentConfig.tokenConfig.salt = tokenLowerSalt;

    tokenLowerBase = clanker.deployToken(baseDeploymentConfig);

    token0IsClankerPoolKeyBase = PoolKey({
        currency0: Currency.wrap(tokenLowerBase),
        currency1: Currency.wrap(weth),
        fee: LPFeeLibrary.DYNAMIC_FEE_FLAG,
        tickSpacing: 200,
        hooks: IHooks(address(hook))
    });

    vm.roll(block.number + 2);

    approveTokens(alice, tokenLowerBase);

    uint128 swapAmount = 1 ether;

    // base clanker pool
    uint256 protocolFeeBefore =
        IPoolManager(poolManager).balanceOf(address(hook), Currency.wrap(weth).toId());
    uint256 amountClankerOut =
        swapExactInputSingle(alice, token0IsClankerPoolKeyBase, weth, swapAmount);

    console.log("amountClankerOut: ", amountClankerOut); // 4_406_408_578_168_640_794_759_221_031

    uint256 protocolFeeAfter =
        IPoolManager(poolManager).balanceOf(address(hook), Currency.wrap(weth).toId());
    console.log("actual protocol fee: ", protocolFeeAfter - protocolFeeBefore); // 90000000000000000
}

function test_fee1() public {
    baseDeploymentConfig.poolConfig.poolData = abi.encode(
        IClankerHookStaticFee.PoolStaticConfigVars({clankerFee: 500_000, pairedFee: 500_000}) // 50% ,
        ↪  protocolFee = 10%
    );
    (bytes32 tokenLowerSalt,) = Utils.generateSalt(
        address(clanker),
        baseDeploymentConfig.tokenConfig,
        baseDeploymentConfig.poolConfig.pairedToken,
        false
    );
    baseDeploymentConfig.tokenConfig.salt = tokenLowerSalt;

    tokenLowerBase = clanker.deployToken(baseDeploymentConfig);

    token0IsClankerPoolKeyBase = PoolKey({
        currency0: Currency.wrap(tokenLowerBase),
        currency1: Currency.wrap(weth),
        fee: LPFeeLibrary.DYNAMIC_FEE_FLAG,
        tickSpacing: 200,
        hooks: IHooks(address(hook))
    });
```

```
    vm.roll(block.number + 2);

    approveTokens(alice, tokenLowerBase);

    uint128 swapAmount = 1 ether;

    // base clanker pool
    uint256 protocolFeeBefore =
        IPoolManager(poolManager).balanceOf(address(hook), Currency.wrap(weth).toId());
    uint256 amountEthIn = swapExactOutputSingle(
        alice, tokenOIsClankerPoolKeyBase, tokenLowerBase, 4_406_408_578_168_640_794_759_221_031
    );

    console.log("amountEthIn: ", amountEthIn); // 100100000000000000

    uint256 protocolFeeAfter =
        IPoolManager(poolManager).balanceOf(address(hook), Currency.wrap(weth).toId());
    console.log("actual protocol fee: ", protocolFeeAfter - protocolFeeBefore); // 91000000000000000
}
```

Recommendation: It is recommended to change the fee adjustment formula, and also to reduce the precision loss, it is recommended to scale the protocol fee precision:

```
--- a/src/hooks/ClankerHook.sol
+++ b/src/hooks/ClankerHook.sol
@@ -290,10 +290,8 @@ abstract contract ClankerHook is BaseHook, Ownable, IClankerHook {
             // since we're taking the protocol fee before the LP swap, we want to
             // take a slightly smaller amount to keep the taken LP/protocol fee at the 20% ratio,
             // this also helps us match the ExactOutput swappingForClanker scenario
-            protocolFee =
-                uint24(uint256(protocolFee) * (10_000 - (uint256(protocolFee) / 100)) / 10_000);
-
-            int128 fee = int128(swapParams.amountSpecified * -int24(protocolFee)) / FEE_DENOMINATOR;
+            uint128 scaledProtocolFee = uint128(protocolFee) * 1e18 / (1_000_000 + protocolFee);
+            int128 fee = int128(swapParams.amountSpecified * -int128(scaledProtocolFee) / 1e18);
             delta = toBeforeSwapDelta(fee, 0);
             poolManager.mint(
                 address(this),
@@ -309,10 +307,8 @@ abstract contract ClankerHook is BaseHook, Ownable, IClankerHook {
         if (!isExactInput && !swappingForClanker) {
             // we increase the protocol fee here because we want to better match
             // the ExactOutput !swappingForClanker scenario
-            protocolFee =
-                uint24(uint256(protocolFee) * (10_000 + (uint256(protocolFee) / 100)) / 10_000);
-
-            int128 fee = int128(swapParams.amountSpecified * int24(protocolFee)) / FEE_DENOMINATOR;
+            uint128 scaledProtocolFee = uint128(protocolFee) * 1e18 / (1_000_000 - protocolFee);
+            int128 fee = int128(swapParams.amountSpecified * int128(scaledProtocolFee) / 1e18);
```

Clanker: Fixed in PR 40.

Cantina Managed: Fix verified.

3.1.3 Dynamic fee mechanisms may take higher volatility to calculate the fee

Severity: Medium Risk

Context: ClankerHookDynamicFee.sol#L158

Description: The Dynamic Fee Mechanism performs a simulated swap before the swap and determines the protocol fee and LP fee based on the tick counts that are crossed during the simulated swap as volatility.

```
try this.simulateSwap(poolKey, swapParams) {
    revert("simulate swap should fail");
```

Since the simulated swap is initiated by hook, `hook._beforeSwap()/_afterSwap()` will not be called in `beforeSwap()/afterSwap()` hook, that is, no protocol fee will be deducted in `hook._beforeSwap()`, which causes the simulation result to be inconsistent with the actual result.

```
function beforeSwap(IHooks self, PoolKey memory key, SwapParams memory params, bytes calldata hookData)
    internal
    returns (int256 amountToSwap, BeforeSwapDelta hookReturn, uint24 lpFeeOverride)
{
    amountToSwap = params.amountSpecified;
    if (msg.sender == address(self)) return (amountToSwap, BeforeSwapDeltaLibrary.ZERO_DELTA, lpFeeOverride);
```

Proof of Concept: Taking `test_SameFeeOk` as the example:

```
function test_SameFeeOk() public {
    // setup dynamic hook config
    baseDeploymentConfig.poolConfig.poolData = abi.encode(
        IClankerHookDynamicFee.PoolDynamicConfigVars({
            baseFee: 100_000,
            maxLpFee: 100_000,
            referenceTickFilterPeriod: 2 seconds,
            resetPeriod: 5 seconds,
            resetTickFilter: 200,
            feeControlNumerator: 5000,
            decayFilterBps: 5000
        })
    );

    (address token, PoolKey memory poolKey) =
        deployTokenGeneratePoolKey(baseDeploymentConfig, false);
    vm.roll(block.number + 2);
    approveTokens(alice, token);

    // swaps works fine (even tho is dumb)
    swapExactInputSingle(alice, poolKey, weth, 100_000 ether);

    // grap lp fee
    (,,, uint24 lpFee) = poolStateView.getSlot0(poolKey.toId());
    console.log("lpFee", lpFee);
}
```

The `afterTick` of the simulated swap is `-38431`, and the `afterTick` of the actual swap is `-38827`.

```
[PASS] test_SameFeeOk() (gas: 4601889)
Logs:
  sim tick -38431
  act tick -38827
  lpFee 100000
```

After removing the protocol fee logic in ClankerHook, the output is:

```
[PASS] test_SameFeeOk() (gas: 4574549)
Logs:
  sim tick -38431
  act tick -38431
  lpFee 100000
```

From this, it can be concluded that the simulated swap does not take into account the protocol fee in `ClankerHook`.

Recommendation: The recommendation is to deduct the fee before simulating the swap, and only the fee in `_beforeSwap()` needs to be applied, the fee in `_afterSwap()` will not affect the simulation results:

```
@@ -175,7 +176,7 @@ contract ClankerHookDynamicFee is ClankerHook, IClankerHookDynamicFee {
         emit EstimatedTickDifference(tickBefore, tickAfter);
     }

-    function _getTicks(PoolKey calldata poolKey, IPoolManager.SwapParams calldata swapParams)
+    function _getTicks(PoolKey calldata poolKey, IPoolManager.SwapParams memory swapParams)
         internal
         returns (int24 tickAfter)
     {
@@ -210,18 +211,33 @@ contract ClankerHookDynamicFee is ClankerHook, IClankerHookDynamicFee {
         }
     }

-    function simulateSwap(PoolKey calldata poolKey, IPoolManager.SwapParams calldata swapParams)
+    function simulateSwap(PoolKey calldata poolKey, IPoolManager.SwapParams memory swapParams)
         external
     {
         if (msg.sender != address(this)) {
             revert("simulateSwap can only be called by the hook");
         }
+        bool token0IsClanker = clankerIsToken0[poolKey.toId()];
+        bool swappingForClanker = swapParams.zeroForOne != token0IsClanker;
+        bool isExactInput = swapParams.amountSpecified < 0;
+        if (isExactInput && swappingForClanker) {
+            uint128 scaledProtocolFee = uint128(protocolFee) * 1e18 / (1_000_000 + protocolFee);
+            int128 fee = int128(swapParams.amountSpecified * -int128(scaledProtocolFee) / 1e18);
+            swapParams.amountSpecified = swapParams.amountSpecified + fee;
+        }
+        if (!isExactInput && !swappingForClanker) {
+            uint128 scaledProtocolFee = uint128(protocolFee) * 1e18 / (1_000_000 - protocolFee);
+            int128 fee = int128(swapParams.amountSpecified * int128(scaledProtocolFee) / 1e18);
+            swapParams.amountSpecified = swapParams.amountSpecified + fee;
+        }
         // run the swap
         poolManager.swap(poolKey, swapParams, abi.encode());
```

Clanker: Fixed in PR 40.

Cantina Managed: Fix verified.

3.1.4 `ClankerAirdrop` extension is not compatible with `Presale`

Severity: Medium Risk

Context: ClankerAirdrop.sol#L101-L107

Description: When the presale starts, the deployment configuration for the tokens needs to be provided.

```
function startPresale(
    IClanker.DeploymentConfig memory deploymentConfig,
    uint256 minEthGoal,
    uint256 maxEthGoal,
    uint256 presaleDuration,
    address recipient,
    uint256 lockupDuration,
    uint256 vestingDuration
) external onlyAdmin returns (uint256 presaleId) {
```

And when the presale ends, user need to provide salt to end the presale and create the token. So the token address is determined until the presale ends.

```
function endPresale(uint256 presaleId, bytes32 salt)
    external
    presaleExists(presaleId)
    updatePresaleState(presaleId)
    returns (address token)
{
    Presale storage presale = presaleState[presaleId];
    // ...
    // deploy token
    token = factory.deployToken(presale.deploymentConfig);
}
```

But in the `ClankerAirdrop` extension, the leaf nodes of the Merkle tree will contain the token address, which means that at the start of the presale, the Merkle root included in the configuration is deterministic and the token address is deterministic.

```
if (
    !MerkleProof.verifyCalldata(
        proof,
        airdrop.merkleRoot,
        keccak256(bytes.concat(keccak256(abi.encode(recipient, token, allocatedAmount))))
    )
) {
    revert InvalidProof();
}
```

Consider the token launcher presales tokens and uses the `ClankerAirdrop` extension, which means that the token launcher has to provide the root of the Merkel tree that contains the token address, once another user deploys this token first, then when the presale is ended, a different salt will be used to create a different token address, thus making the `ClankerAirdrop` extension invalid and the airdropped tokens cannot be claimed.

Recommendation: Since token address in leaf nodes is redundant, it is recommended not to include token address in leaf nodes:

```
  if (
      !MerkleProof.verifyCalldata(
          proof,
          airdrop.merkleRoot,
-         keccak256(bytes.concat(keccak256(abi.encode(recipient, token, allocatedAmount))))
+         keccak256(bytes.concat(keccak256(abi.encode(recipient, allocatedAmount))))
      )
  ) {
      revert InvalidProof();
  }
```

Clanker: Fixed in PR 39.

Cantina Managed: Fix verified.

3.1.5 Fees rounding to 0 can cause a DoS when tokens disallow 0 value transfers/approves

Severity: Medium Risk

Context: ClankerLpLockerMultiple.sol#L292-L293, ClankerLpLockerMultiple.sol#L302-L303

Finding Description: When small fees combined with small `rewardBps` values round to 0, the full transaction can end up reverting if one of the tokens is a token that reverts on 0 value transfers. And for revert on 0 approve tokens (BNB), subsequent approve calls will also revert.

Impact Explanation: High due to DoS of core functionality such as swaps.

Likelihood Explanation: Low likelihood due to only affecting particular tokens and requiring rounding to 0.

Recommendation: Check for 0 values before attempting transfers/approves.

Clanker: Fixed in PR 43.

Cantina Managed: Fix verified.

3.1.6 Use of the `IERC20` interface means tokens with no return value, such as USDT, are not supported

Severity: Medium Risk

Context: ClankerFeeLocker.sol#L24

Finding Description: The IERC20 interface reverts at the language level when return values, or lack thereof, do not conform to the interface. Tokens, such as USDT, where there is no return value are therefore not supported by the protocol.

Likelihood Explanation: Given the marketcap and widespread usage of of USDT, it would be at least somewhat likely that a project will attempt to use this token at some point.

Recommendation: Prefer `safeTransfer` and related approval functions for interactions with ERC20 token implementations.

Clanker: Fixed in PR 43.

Cantina Managed: Fix verified.

3.2 Low Risk

3.2.1 Incorrect `startingTick` when placing initial liquidity

Severity: Low Risk

Context: ClankerLpLockerMultiple.sol#L202-L227

Description: When placing initial liquidity, the `startingTick` is `tickLower[0]`.

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

There is an implicit assumption that `tickLower[0]` must be less than or equal to all `tickLower[i]`, otherwise `getLiquidityForAmounts()` will return 0.

```
uint256 liquidity = LiquidityAmounts.getLiquidityForAmounts(
    startingTick.getSqrtPriceAtTick(), lowerSqrtPrice, upperSqrtPrice, amount0, amount1
);
```

Since Uniswap V4 does not allow adding 0 liquidity, this will cause the deployment to fail:

```
if (liquidityDelta == 0) {
    // disallow pokes for 0 liquidity positions
    if (liquidity == 0) CannotUpdateEmptyPosition.selector.revertWith();
```

Proof of Concept:

```
function test_poc() public {
    baseDeploymentConfig.lockerConfig.tickLower = new int24[](3);
    baseDeploymentConfig.lockerConfig.positionBps = new uint16[](3);
    baseDeploymentConfig.lockerConfig.tickUpper = new int24[](3);

    baseDeploymentConfig.lockerConfig.tickLower[0] = -60_000;
    baseDeploymentConfig.lockerConfig.tickUpper[0] = 80_000;
    baseDeploymentConfig.lockerConfig.positionBps[0] = 500;

    baseDeploymentConfig.lockerConfig.tickLower[1] = -230_400;
    baseDeploymentConfig.lockerConfig.tickUpper[1] = -60_000;
    baseDeploymentConfig.lockerConfig.positionBps[1] = 9000;

    baseDeploymentConfig.lockerConfig.tickLower[2] = 80_000;
    baseDeploymentConfig.lockerConfig.tickUpper[2] = 430_400;
    baseDeploymentConfig.lockerConfig.positionBps[2] = 500;

    // grab higher than paired deployment salt
    (address tokenA, PoolKey memory tokenAPoolKey) =
        deployTokenGeneratePoolKey(baseDeploymentConfig, true); // revert with CannotUpdateEmptyPosition
}
```

Recommendation: Since `lockerConfig.tickLower[i] >= poolConfig.tickIfToken0IsClanker` was checked earlier, so to ensure that `getLiquidityForAmounts()` returns the correct result later, `startingTick` should be `poolConfig.tickIfToken0IsClanker` instead of `lockerConfig.tickLower[0]`.

```
- int24 startingTick =
-     tokenOIsClanker ? lockerConfig.tickLower[0] : -lockerConfig.tickLower[0];
+ int24 startingTick =
+     tokenOIsClanker ? poolConfig.tickIfToken0IsClanker : -poolConfig.tickIfToken0IsClanker;
```

Clanker: Fixed in PR 41.

Cantina Managed: Fix verified.

3.2.2 Updates to `protocolFee` in `_beforeSwap()` affect simulated swaps in the dynamic fee mechanism

Severity: Low Risk

Context: ClankerHook.sol#L293-L313

Description: When the protocol adjusted protocolFee in `_beforeSwap()`, protocolFee was incorrectly updated:

```
if (isExactInput && swappingForClanker) {
    // since we're taking the protocol fee before the LP swap, we want to
    // take a slightly smaller amount to keep the taken LP/protocol fee at the 20% ratio,
    // this also helps us match the ExactOutput swappingForClanker scenario
    protocolFee =
        uint24(uint256(protocolFee) * (10_000 - (uint256(protocolFee) / 100)) / 10_000);
```

In the dynamic fee mechanism, the protocol simulates swaps to determine the dynamic fee. Under some conditions, the protocol directly uses the protocol fee after the last swap to simulate the swap. Since protocolFee is incorrectly updated, this will affect the result of the simulated swap.

```
if (resetTickDifference > poolCVars.resetTickFilter) {
    // the tick difference is large enough, don't kill the reference tick
    poolFVars.resetTick = tickBefore;
    poolFVars.resetTickTimestamp = block.timestamp;
} else {
```

Recommendation: The recommendation in "Incorrect protocolFee adjustment in `_beforeSwap()`" will also address this issue.

```
--- a/src/hooks/ClankerHook.sol
+++ b/src/hooks/ClankerHook.sol
@@ -290,10 +290,8 @@ abstract contract ClankerHook is BaseHook, Ownable, IClankerHook {
             // since we're taking the protocol fee before the LP swap, we want to
             // take a slightly smaller amount to keep the taken LP/protocol fee at the 20% ratio,
             // this also helps us match the ExactOutput swappingForClanker scenario
-            protocolFee =
-                uint24(uint256(protocolFee) * (10_000 - (uint256(protocolFee) / 100)) / 10_000);
-
-            int128 fee = int128(swapParams.amountSpecified * -int24(protocolFee)) / FEE_DENOMINATOR;
+            uint128 scaledProtocolFee = uint128(protocolFee) * 1e18 / (1_000_000 + protocolFee);
+            int128 fee = int128(swapParams.amountSpecified * -int128(scaledProtocolFee) / 1e18);
             delta = toBeforeSwapDelta(fee, 0);
             poolManager.mint(
                 address(this),
@@ -309,10 +307,8 @@ abstract contract ClankerHook is BaseHook, Ownable, IClankerHook {
         if (!isExactInput && !swappingForClanker) {
             // we increase the protocol fee here because we want to better match
             // the ExactOutput !swappingForClanker scenario
-            protocolFee =
-                uint24(uint256(protocolFee) * (10_000 + (uint256(protocolFee) / 100)) / 10_000);
-
-            int128 fee = int128(swapParams.amountSpecified * int24(protocolFee)) / FEE_DENOMINATOR;
+            uint128 scaledProtocolFee = uint128(protocolFee) * 1e18 / (1_000_000 - protocolFee);
+            int128 fee = int128(swapParams.amountSpecified * int128(scaledProtocolFee) / 1e18);
```

Clanker: Fixed in PR 40.

Cantina Managed: Fix verified.

3.2.3 Use of `.transfer` to refund excess causes revert when contracts include receive logic or state updates

Severity: Low Risk

Context: ClankerPresaleEthToCreator.sol#L238

Finding Description: The gas stipend of 2300 disallows writes post Istanbul hardfork (see evm.codes `When the amount of gas left to the transaction is less than or equal 2300`) which will cause the transaction to revert for any contract or smart account updating storage on receipt of native currency.

This branch of the code is only reached as the presale is ending, making the likelihood of occurring quite low.

Recommendation: Prefer `.call` as used elsewhere in the codebase to allow contracts to not run out of gas here. Importantly, switching from `transfer` means that reentrancy must be prevented through other means. Following checks/effects/interactions ordering and moving the `.call` to the end of the function is needed if implementing the first part of the recommendation.

Clanker: Fixed in PR 42.

Cantina Managed: Fix verified.

3.2.4 Differing endTime handling between `buyIntoPresale` and `updatePresaleState`

Severity: Low Risk

Context: ClankerPresaleEthToCreator.sol#L226

Description: The check in `buyIntoPresale` reverts when `endTime` is before the current block. `updatePresaleState` on the other hand allows the state to transition from active to `PresaleStatus.SuccessfulMinimumHi/PresaleStatus.Failed` when `presale.endTime == block.timestamp`. Attempts to buy in the final second may be prevented when frontrun with a call to `withdrawFromPresale` with amount of 1 or more which will trigger the update causing the sale to close. Likelihood and impact are both low.

Recommendation: Use consistent handling for the final second of the sale.

Clanker: Fixed in PR 42.

Cantina Managed: Fix verified.

3.2.5 Tokens that transfer of less than amount allow fees to be drained

Severity: Low Risk

Context: ClankerFeeLocker.sol#L22

Finding Description: Malicious `allowedDepositor` can extract fees in tokens like cUSDCv3 where an amount of max uint resolves to the current balance (see address 0xaec1954467b6d823a9042e9e9d6e4f40111069a9):

```
if (amount == type(uint256).max) {
    amount = balanceOf(src);
}
```

Calling the function with `type(uint256).max` credits the caller with this amount when the token contract will only be transferring their current balance. Importantly, having `type(uint256).max` recorded in `feesToClaim` means that transfers out of the contract will transfer out the contract's entire balance.

Impact Explanation: Impact is only in tokens with this behavior.

Likelihood Explanation: Low requires access to be given to malicious party.

Recommendation: Consider balance before/after deltas to conform the actual amount received.

Clanker: Fixed in PR 43.

Cantina Managed: Fix verified.

3.2.6 Owner may skim fees for pools using a token with a malicious callback

Severity: Low Risk

Context: ClankerLpLockerMultiple.sol#L350

Finding Description: `owner` can skim fees by:

- Using pool where the paired token has a malicious transfer callback.
- Using the callback to call the `withdrawERC20` before the balance diff takes place.

Impact Explanation: Low impact as only possible for pools with the malicious pair.

Likelihood Explanation: Extremely low likelihood as requires the owner to be malicious.

Recommendation: Add reentrancy mutex to owner functions as well.

Clanker: Fixed in PR 43.

Cantina Managed: Fix verified.

3.2.7 Fee on transfer tokens are not supported

Severity: Low Risk

Context: ClankerLpLockerMultiple.sol#L304-L309

Finding Description: Fee on transfer tokens not fully supported by the protocol. Bringing fees into the system with `_bringFeesIntoContract` relies on balance diffs, meaning any fees taken prior to the tokens entering the contracts are not problematic. When moving the rewards into the locker, *expected* amounts are accounted for, rather than *actual* amounts:

```
function storeFees(address feeOwner, address token, uint256 amount) external nonReentrant {

    if (!allowedDepositors[msg.sender]) revert Unauthorized();


    bool success = IERC20(token).transferFrom(msg.sender, address(this), amount);
    if (!success) revert TransferFailed();

    feesToClaim[feeOwner][token] += amount;  // <- expected amount may not match actual amount for fee on
    ↪  transfer tokens
    emit StoreTokens(feeOwner, token, feesToClaim[feeOwner][token], amount);
}
```

Likelihood Explanation: Mixed likelihood as large marketcap tokens such as USDT have the ability for owner to turn on transfer fees. For the most part, currently activated fee on transfer tokens are out of scope for this project.

Recommendation: Consider explicit handling of these tokens.

Clanker: Fixed in PR 43.

Cantina Managed: Fix verified.

3.2.8 Temporary DoS when salts are maliciously reused

Severity: Low Risk

Context: ClankerHook.sol#L125

Finding Description: Calling `initializePoolOpen` ahead of a token deploy causes `deployToken` to revert due to the pool already being initialized. May cause temporary griefing of ending presales. The protocol has already handled the edge case by allowing a custom salt to be passed in at the last minute.

Proof of Concept:

```
function test_poc() public {
    // start presale
    vm.prank(clankerTeamEOA);
    uint256 presaleId = presaleExtension.startPresale(
        baseDeploymentConfig, 10 ether, 1000 ether, 1 days, bob, 0 days, 0 days
    );

    // buy into presale
    vm.prank(alice);
    presaleExtension.buyIntoPresale{value: 10 ether}(presaleId);
    IClanker.TokenConfig memory tokenConfig = baseDeploymentConfig.tokenConfig;
    address predictedAddress = address(
        uint160(
            uint256(
                keccak256(
                    abi.encodePacked(
                        bytes1(0xff),
```

```
                        address(clanker),
                        keccak256(abi.encode(tokenConfig.tokenAdmin, bytes32(0))),
                        keccak256(
                            abi.encodePacked(
                                type(ClankerToken).creationCode,
                                abi.encode(
                                    tokenConfig.name,
                                    tokenConfig.symbol,
                                    TOKEN_SUPPLY,
                                    tokenConfig.tokenAdmin,
                                    tokenConfig.image,
                                    tokenConfig.metadata,
                                    tokenConfig.context,
                                    tokenConfig.originatingChainId
                                )
                            )
                        )
                    )
                )
            )
        )
    );
    console.log(predictedAddress);
    hook.initializePoolOpen(
        predictedAddress,
        baseDeploymentConfig.poolConfig.pairedToken,
        baseDeploymentConfig.poolConfig.tickIfToken0IsClanker,
        baseDeploymentConfig.poolConfig.tickSpacing,
        baseDeploymentConfig.poolConfig.poolData
    );
    vm.warp(block.timestamp + 1 days);
    // trigger presale end
    vm.prank(clankerTeamEOA);
    address token = presaleExtension.endPresale(presaleId, bytes32(0)); // [FAIL: custom error 0x7983c051]
    console.log(token);
}
```

Recommendation: Consider reserving addresses for a presale, blocking other methods of deploy for the same predicted address.

Clanker: Acknowledged.

Cantina Managed: Acknowledged.

3.3 Gas Optimization

3.3.1 Skip fee handling when there are 0 fees

Severity: Gas Optimization

Context: ClankerHook.sol#L254-L257

Description/Recommendation: When fees are 0, gas can be saved by avoiding PoolManager burning/taking related to fee amount.

Clanker: Acknowledged.

Cantina Managed: Acknowledged.

3.3.2 Factory address does not change and can be immutable

Severity: Gas Optimization

Context: ClankerPresaleEthToCreator.sol#L26

Description/Recommendation: Consider making the value an immutable one as the contracts are not upgradeable and there is not functionality to change set a new factory address. Gas can be saved by eliminating the sload.

Clanker: Fixed in PR 42.

Cantina Managed: Fix verified.

3.3.3 Consider fusing multiple loops into one to eliminate duplicate operations

Severity: Gas Optimization

Context: Clanker.sol#L253-L278

Description/Recommendation: When all loops are ececuted successfully, gas can be saved by combing them into one, instead of 3 separate iterations.

Note: doing does does eliminate saving for early reverting e.g. when `extensionSupplyPercentage > MAX_EXTENSION_BPS` the revert occurs before summing the `expectedExtensionEth` saving gas only in the reverting case.

Inspecting the generated `yul` reveals an additional micro savings when loading length from memory 1 time rather than 3 if the length is cached.

Clanker: Acknowledged.

Cantina Managed: Acknowledged.

3.3.4 Eliminate division operation by using a constant

Severity: Gas Optimization

Context: Clanker.sol#L281

Description/Recommendation: Given both `TOKEN_SUPPLY` and `BPS` are constants, the result of `TOKEN_SUPPLY / BPS` can be a constant: `10_000_000e18`. Further, the multiplication can be unchecked as well since `MAX_EXTENSION_BPS * 10_000_000e18` doesn't overflow.

Clanker: Acknowledged.

Cantina Managed: Acknowledged.

3.4 Informational

3.4.1 Typo in `IClankerLpLocker` import

Severity: Informational

Context: Clanker.sol#L10

Description/Recommendation: `ICLankerLPLocker.sol` is the correct file name.

Clanker: Acknowledged.

Cantina Managed: Acknowledged.

3.4.2 Unresolved TODO for failed fee transfer to `withdrawFeeRecipient`

Severity: Informational

Context: ClankerPresaleEthToCreator.sol#L292

Description/Recommendation: To address the TODO a pull payment or force transfer could both work: see solady for example.

Clanker: Fixed in PR 42.

Cantina Managed: Fix verified.

3.4.3 Presale status getter returns `Active` before a presale is created

Severity: Informational

Context: IClankerPresaleEthToCreator.sol#L31

Description/Recommendation: Consider adding to the enum a default value in the first position `Default` or `NotCreated`.

Clanker: Fixed in PR 42.

Cantina Managed: Fix verified.

3.4.4 Consider using OZ's `Ownable2Step`

Severity: Informational

Context: OwnerAdmins.sol#L7

Description/Recommendation: Consider `Ownable2Step` to prevent missteps when transferring to a new owner.

Clanker: Acknowledged.

Cantina Managed: Acknowledged.

3.4.5 Elevate comments to labelling the mappings directly

Severity: Informational

Context: ClankerPresaleEthToCreator.sol#L29-L33

Description/Recommendation: Later solidity versions allow for language level labelling to replace the use of comments explaining the mappings. Consider the following edit:

```
- mapping(uint256 => Presale) public presaleState; // presaleId -> presale state
+ mapping(uint256 presaleId => Presale) public presaleState;

  // presale user buy and claim amounts
- mapping(uint256 => mapping(address => uint256)) public presaleBuys; // presaleId -> address -> amount
- mapping(uint256 => mapping(address => uint256)) public presaleClaimed; // presaleId -> address -> amount
+ mapping(uint256 presaleId => mapping(address account => uint256 amount)) public presaleBuys;
+ mapping(uint256 presaleId => mapping(address account => uint256 amount)) public presaleClaimed;
```

Clanker: Fixed in PR 42.

Cantina Managed: Fix verified.

3.4.6 Airdrop assumptions

Severity: Informational

Context: ClankerAirdrop.sol#L62-L64

Description/Recommendation: Some properties about the airdrops worth documenting, no code changes recommended.

- No restriction on duration. Caution advised to avoid setting unreasonably long durations.
- Roots cannot be republished. Prevents later malicious behavior, however, makes it important to validate to avoid publishing errors.
- Unclaimed effectively burned. There is no means of recovering unclaimed amounts, they should be considered burned.
- Total is not certain to be sum of leaves. Offchain validation is assumed to handle validation of the published amounts; a discrepancy where leaf nodes sum to a value greater than the total mean the airdrop is first come first serve.

Clanker: Acknowledged.

Cantina Managed: Acknowledged.

3.4.7 `ClankerFeeLocker` depositors can only be allowed and not disallowed

Severity: Informational

Context: ClankerFeeLocker.sol#L16-L19

Description/Recommendation: Recommend adding the ability for owner to disallow depositors. Especially relevant if an upgradeable contract is ever added as the implementation cannot be permanently trused.

Clanker: Acknowledged.

Cantina Managed: Acknowledged.

3.4.8 Malicious token pairs are problematic

Severity: Informational

Context: ClankerHook.sol#L275

Description/Recommendation: A malicious token in a pair is able to create a token with callbacks or hooks embeded to interact with an unlocked pool. The calls to a token originating from both `_lpLockerFeeClaim` and a later `settle` open a sandwiching opportunity. This particular scenario would only affect pools with the malicious token. Severity is informational as the project team is already aware of the vector and odd token pairs are considered out of scope. Issue created for documentation purposes.

Recommendation: Controlling what is presented in the UI along with cautioning about untrusted pools is recommended. The project team indicated this strategy is already in use.

Clanker: Acknowledged.

Cantina Managed: Acknowledged.