ChainLight

Blaster Swap Security Audit

: Blaster Swap v2 / v3

May 1, 2024

Revision 1.1

ChainLight@Theori

Table of Contents

Executive Summary

Starting on Apr 12, 2024, ChainLight of Theori audited the smart contract of Blaster Swap v2 / v3 for a week. Blaster Swap is a fork of Uniswap v2 / v3 (the largest and time-tested AMM) with minor modifications for Blast. Therefore, modifications from upstream and its side-effect were the main concern of the audit. We primarily considered the issues/impacts listed below.

- Theft of funds
- Violation of liquidity invariant
- Incorrect accounting when handling claimed yields
- Any modification to core logic where not expected

As a result, we identified the issues listed below.

- Total: 7
- High: 1 (Permanent freeze of yield)
- Medium: 1
- Low: 3 (Liquidity providers may lose part of yield, etc.)
- Informational: 2

Audit Overview

Scope

Name	Blaster Swap Security Audit
Target / Version	• Git Repository (`blasterswap/blasterswap-core-v2`): commit `01ab3d58f13466fcb8a8adb8f3c573f9c4144d5a` • Git Repository (`blasterswap/blasterswap-periphery-v3`): commit `17c8b8c4f43ea0a3d672febf28f9cf7890c05dbb` • Git Repository (`blasterswap/blasterswap-v3-core`): commit `6b66b1897f482ba345a1e8e9053497eb9fdc78e1`
Application Type	Smart contracts
Lang. / Platforms	Smart contracts [Solidity]

Code Revision

N/A

Severity Categories

Severity	Description
Critical	The attack cost is low (not requiring much time or effort to succeed in the actual attack), and the vulnerability causes a high-impact issue. (e.g., Effect on service availability, Attacker taking financial gain)
High	An attacker can succeed in an attack which clearly causes problems in the service's operation. Even when the attack cost is high, the severity of the issue is considered "high" if the impact of the attack is remarkably high.
Medium	An attacker may perform an unintended action in the service, and the action may impact service operation. However, there are some restrictions for the actual attack to succeed.
Low	An attacker can perform an unintended action in the service, but the action does not cause significant impact or the success rate of the attack is remarkably low.
Informational	Any informational findings that do not directly impact the user or the protocol.
Note	Neutral information about the target that is not directly related to the project's safety and security.

Status Categories

Status	Description
Reported	ChainLight reported the issue to the client.
WIP	The client is working on the patch.
Patched	The client fully resolved the issue by patching the root cause.
Mitigated	The client resolved the issue by reducing the risk to an acceptable level by introducing mitigations.
Acknowledged	The client acknowledged the potential risk, but they will resolve it later.
Won't Fix	The client acknowledged the potential risk, but they decided to accept the risk.

Finding Breakdown by Severity

Category	Count	Findings
Critical	**0**	• N/A
High	**1**	• `BLASTERSWAP-002`
Medium	**1**	• `BLASTERSWAP-005`
Low	**3**	• `BLASTERSWAP-001` • `BLASTERSWAP-004` • `BLASTERSWAP-006`
Informational	**2**	• `BLASTERSWAP-003` • `BLASTERSWAP-007`
Note	**0**	• N/A

Findings

Summary

#	ID	Title	Severity	Status
1	BLASTERSWAP-001	`BlasterswapV2Router02._addLiquidity()` must consider the unclaimed yield	Low	Mitigated
2	BLASTERSWAP-002	`BlasterswapV3Pool._updatePosition()` incorrectly updates the fee growth from the rebasing token's yield	High	Patched
3	BLASTERSWAP-003	Hardcoded chain-dependent values in `NonfungibleTokenPositionDescriptor.tokenRatioPriority()` should be updated	Informational	Patched
4	BLASTERSWAP-004	Significant accumulation of unclaimed yield may incur losses to liquidity providers of `BlasterswapV2Pair`	Low	Mitigated
5	BLASTERSWAP-005	`BlasterswapV3Pool._updatePosition()` must claim the yield from rebase tokens first	Medium	Patched
6	BLASTERSWAP-006	Missing gas mode settings	Low	Patched

#	ID	Title	Severity	Status
7	BLASTERSWAP-007	Minor suggestions	Informational	Patched

#1 BLASTERSWAP-001

`BlasterswapV2Router02._addLiquidity()` must consider the unclaimed yield

ID	Summary	Severity
`BLASTERSWAP-001`	`BlasterswapV2Router02._addLiquidity()` does not consider the yield from rebasing tokens that the pair has not yet claimed. As a result, liquidity providers may oversupply one side (token) and incur losses.	Low

Description

In BlasterSwap, liquidity providers can receive yield from `WETHRebasing` and `USDB`, rebasing tokens operated by Blast. `BlasterswapV2Router02._addLiquidity()` calculates the optimal token amount when adding liquidity and checks whether liquidity can be supplied in the price range expected by the liquidity provider.

However, the optimal token amount/rate calculation implemented in `_addLiquidity()` does not consider the yield from rebasing tokens that the pair has not yet claimed. Since a pair containing a rebasing token would claim the yield before the provision/removal of liquidity, the calculation result will be incorrect.

Suppose liquidity is supplied at a ratio other than the optimal token ratio. In that case, tokens on one side will be oversupplied, and the excess amount will be divided with existing liquidity providers. Additionally, due to the yield, the token price in the pair may diverge from the market price. In this circumstance, liquidity providers can incur impermanent losses when adding liquidity.

Impact

Low

Liquidity providers may incur losses when adding liquidity to a pool with unclaimed rebasing token yield. However, considering the frequency of the yield claim and the pool's liquidity, the loss will be tiny in the current environment (deployed pools).

Recommendation

(NOTE: Fixing the `BLASTERSWAP-004` issue through primary recommendation would invalidate this issue.)

`BlastswapV2Library.getReserves()` must add the unclaimed yield of the pair to the reserve value by querying `USDB.getClaimableAmount(pair)` and `WETH.getClaimableAmount(pair)` when `tokenA` or `tokenB` is `USDB` or `WETH`.

For instance:

```
function getReserves(
    address factory,
    address tokenA,
    address tokenB
) internal view returns (uint reserveA, uint reserveB) {
    (address token0, ) = sortTokens(tokenA, tokenB);
    (uint reserve0, uint reserve1, ) = IBlasterswapV2Pair(
        pairFor(factory, tokenA, tokenB)
    ).getReserves();
    (reserveA, reserveB) = tokenA == token0
        ? (reserve0, reserve1)
        : (reserve1, reserve0);

    if (tokenA == address(USDB) || tokenB == address(USDB)) {
        if (tokenA == address(USDB)) {
          reserveA += USDB.getClaimableAmount(pairFor(factory, tokenA, tok
enB));
        }
        else {
          reserveB += USDB.getClaimableAmount(pairFor(factory, tokenA, tok
enB));
        }
    }

    if (tokenA == address(WETH) || tokenB == address(WETH)) {
        if (tokenB == address(WETH)) {
          reserveA += WETH.getClaimableAmount(pairFor(factory, tokenA, tok
enB));
        }
```

```
        else {
          reserveB += WETH.getClaimableAmount(pairFor(factory, tokenA, tok
enB));
        }
    }
}
```

Remediation

Mitigated

Since v2 contracts are immutable and already deployed, this will be mitigated by running a script that periodically triggers the yield claim.

#2 `BLASTERSWAP-002` `BlasterswapV3Pool._updatePosition()` incorrectly updates the fee growth from the rebasing token's yield

ID	Summary	Severity
`BLASTERSWAP-002`	`BlasterswapV3Pool._updatePosition()` adds the raw amount of rebasing token yield claimed to the `feeGrowthGlobal0X128` and `feeGrowthGlobal1X128` variables. However, it should be scaled since these variables are in the `Q128.128` format.	High

Description

`feeGrowthGlobal0X128` and `feeGrowthGlobal1X128` are the total fees collected in the pool per unit of `liquidity`. However, `BlasterswapV3Pool._updatePosition()` adds the raw amount of rebasing token (`USDB`, `WETHRebasing`) yield claimed to the `feeGrowthGlobal0X128` and `feeGrowthGlobal1X128` variables. Before added to the fee growth variables, the amount must be multiplied by `FixedPoint128.Q128` (since fee variables are in the `Q128.128` format) and divided by liquidity.

`Liquidity` is an `int128` type variable and is thus always smaller than the value of `FixedPoint128.Q128`. As a result, the `_updatePosition()` always distributes less to liquidity providers than the correct amount.

Impact

High

A lesser amount of rebasing token yield than they should get is distributed to liquidity providers. The remainder are permanently frozen in the contract.

Recommendation

In `BlasterswapV3Pool._updatePosition()`, multiply the claimed amount by `FixedPoint128.Q128`, and then divide by `liquidity`, before adding to the `feeGrowthGlobal0X128` or `feeGrowthGlobal1X128` variable as shown in the code example below.

```
if(liquidity > 0) feeGrowthGlobal0X128 += FullMath.mulDiv(amountClaimedUSD
B, FixedPoint128.Q128, liquidity);
```

Remediation

Patched

It is patched as recommended.

#3 BLASTERSWAP-003 Hardcoded chain-dependent values in `NonfungibleTokenPositionDescriptor.tokenRatioPriority()` should be updated

ID	Summary	Severity
BLASTERSWAP-003	In the `NonfungibleTokenPositionDescriptor` contract, the `tokenRatioPriority()` has no effect except for `WETH` since it does not have hardcoded addresses for Blast.	Informational

Description

In `NonfungibleTokenPositionDescriptor`, since the `tokenRatioPriority()` is only implemented for Ethereum (`chainId` 1), it fails to assign priorities to major tokens on Blast, such as `USDB`. When the `tokenURI()` is called to fetch metadata for a position, it uses `tokenRatioPriority()`.

For instance, in the `USDB/ORBIT` pair where `token0=USDB` and `token1=ORBIT`, `USDB` should be the `quoteToken`. However, due to the absence of the priority designation, it incorrectly assigns the value `quoteTokenAddress` as `token1` and `baseTokenAddress` as `token0`.

Impact

Informational

While there are no direct impacts on the actual positions, this issue leads to the creation of position NFTs with the reversed `quoteToken` and `baseToken` in the metadata.

Recommendation

Modify `tokenRatioPriority()` to return `TokenRatioSortOrder.NUMERATOR_MOST` for `USDB` when `chainId=0x13e31`. And ensure that `WETH` address is correct when deploying `NonfungibleTokenPositionDescriptor`.

Remediation

Patched

It is patched as recommended.

#4 BLASTERSWAP-004 Significant accumulation of unclaimed yield may incur losses to liquidity providers of BlasterswapV2Pair

ID	Summary	Severity
BLASTERSWAP-004	When a significant amount of unclaimed yield is accumulated, existing liquidity providers of a BlasterswapV2Pair containing rebasing tokens may not receive the yields from rebasing tokens in full. And new liquidity providers may suffer unexpected impermanent loss.	Low

Description

When liquidity is added or removed, BlasterswapV2Pair claims yield from rebasing tokens. The yield remains in the pool, increasing liquidity and the value of LP tokens. If liquidity addition/removal does not occur for a while, a large amount of unclaimed yield may accumulate.

The problem is that the increase in balance lowers the token's value in the pool, so the rebasing token's price in the pair may fall below the market price. In this case, an arbitrage opportunity may open up when the difference is significant, and arbitrageurs can take a portion of the yield virtually. In other words, liquidity providers of a pair containing rebasing tokens may not receive the yields from rebasing tokens in full.

Moreover, when adding liquidity, the pair's token ratio may differ from the market price due to the claim of accumulated rebasing token yield. This would expose liquidity providers to impermanent losses by providing liquidity at an incorrect price. However, the loss can be recouped by resetting the pair's token price to the same as the market price through swap(). In arbitrageurs' perspective, they can arbitrage the price discrepancy caused by the claim of yield by triggering the claim first by providing liquidity in small amounts. A liquidity provider adding liquidity after this action will not incur impermanent loss from yield.

Impact

Low

A third party can take part of the yield from a rebasing token, not liquidity providers. However, considering the frequency of the yield claim and the pool's liquidity, the loss will be tiny in the current environment (deployed pools).

Remediation

Mitigated

Since v2 contracts are immutable and already deployed, this will be mitigated by running a script that periodically triggers the yield claim.

#5 BLASTERSWAP-005 BlasterswapV3Pool._updatePosition() must claim the yield from rebase tokens first

ID	Summary	Severity
BLASTERSWAP-005	`BlasterswapV3Pool._updatePosition()` must first call `claim()` of rebase token and update variables `feeGrowthGlobal0X128`, `feeGrowthGlobal1X128`.	Medium

Description

`BlasterswapV3Pool._updatePosition()` adjusts the liquidity of LP positions by calling `position.update(liquidityDelta, feeGrowthInside0X128, feeGrowthInside1X128)`. When the liquidity of the position is modified, `feeGrowthInside0X128` and `feeGrowthInside1X128` variables must be the latest values to enable accurate fee distribution. However, `_updatePosition()` calculates `feeGrowthInside0X128`, `feeGrowthInside1X128` before updating `feeGrowthGlobal0X128`, `feeGrowthGlobal1X128` value from the `claim()`'s result. Thus, fees may be incorrectly distributed since the position liquidity is updated without the yield claimed from rebasing tokens reflected in the `feeGrowthInside0X128` and `feeGrowthInside1X128` variables.

For instance, assume that an attacker provides liquidity, including the current token price, when the `claimableAmount` of rebase tokens is not zero. In `_updatePosition()`, `feeGrowthInside0X128` and `feeGrowthInside1X128` of the new position are updated using the values before the claim, and later, `feeGrowthGlobal0X128` and `feeGrowthGlobal1X128` are updated using the result of the `claim()`. Then the attacker will immediately call `burn` to remove the liquidity, and `position.update()` will calculate `tokensOwed0` by multiplying the position's liquidity for the difference between the `feeGrowthInside0X128` value (that is affected by the claim amount) and the `self.feeGrowthInside0LastX128` value (that is not affected), as shown in the code below. Namely, an attacker can receive abnormally inflated fees just by adding and removing liquidity when the `claimableAmount` is not zero.

```
// calculate accumulated fees
uint128 tokensOwed0 =
    uint128(
```

```
        FullMath.mulDiv(
            feeGrowthInside0X128 - _self.feeGrowthInside0LastX128,
            _self.liquidity,
            FixedPoint128.Q128
        )
    );
```

Conversely, suppose a benign user calls `burn` when `claimableAmount` is not zero. In that case, a fee smaller than the amount actually due by using the `feeGrowthInside0X128` (or `feeGrowthInside1X128`) value that is not affected by the `claimableAmount`.

Impact

Medium

Where there is an unclaimed yield, an attacker can receive more fees that they are not entitled to by exploiting this issue, and benign users may receive less. Moreover, if `feeGrowthGlobal0X128` and `feeGrowthGlobal1X128` are updated correctly by patching the `BLASTERSWAP-002` issue (or the issue did not exist), the loss will become much larger.

Recommendation

`BlasterswapV3Pool._updatePosition()` must first (at the beginning of the function) call `claim()` and update variables `feeGrowthGlobal0X128`, `feeGrowthGlobal1X128`.

Remediation

Patched

It is patched as recommended.

#6 BLASTERSWAP-006 Missing gas mode settings

ID	Summary	Severity
BLASTERSWAP-006	Some contracts lack a gas claim configuration, so the team cannot claim the gas incurred by those.	Low

Description

The BlasterSwap team cannot claim gas fees incurred in some contracts of BlasterSwap since the `BlasterswapV2ERC20`, `BlasterswapV3Factory`, `V3Migrator`, `NonfungiblePositionManager`, and `SwapRouter` contracts lack relevant configuration.

Impact

Low

Part of the gas fees incurred in the protocol goes to the Blast sequencer instead of the team.

Recommendation

Add the missing `configureClaimableGas()` and `configureGovernor()` calls to `BlasterswapV2ERC20`, `BlasterswapV3Factory`, `V3Migrator`, `NonfungiblePositionManager`, and `SwapRouter`.

Remediation

Patched

The fix is applied to contracts in the hot path only.

#7 BLASTERSWAP-007 Minor suggestions

ID	Summary	Severity
BLASTERSWAP-007	The description includes multiple suggestions for preventing incorrect settings caused by operational mistakes, mitigating potential issues, improving code maturity and readability, and other minor issues.	Informational

Description

Code Readability

- `UniswapInterfaceMulticall.sol` file should be renamed to match the contract name, `BlasterswapInterfaceMulticall`.

Typo

- The string "Uniswap" of `generateDescriptionPartOne()` and `generateName()` in `NFTDescriptor` should be changed to "BlasterSwap" to match the service name.

Gas Optimization

- In `BlasterswapV3Pool._updatePosition()`, respectively copying the storage variables `token0` and `token1` to memory variables `_token0` and `_token1`, can reduce gas usage by avoiding repetitive access to storage variables.
- In `BlasterswapV2Pair.claimYieldAndUpdate()`, adding an early return statement when both `amountClaimedUSDB` and `amountClaimedWETH` are zero can reduce gas usage by avoiding unnecessary computations.

Other

- Since yields from rebasing tokens are treated as fees in V3, they are only allocated to positions that have the price at the time of claim in their range; this seems to be a desirable behavior. However, it should be documented since it may differ from users' expectations.

Impact

Informational

Recommendation

Consider applying the suggestions in the description above.

Remediation

Patched

Selected suggestions are applied.

Revision History

Version	Date	Description
1.0	Apr 29, 2024	Initial version
1.1	May 1, 2024	Revised severity and remediation status

Theori, Inc. (“We”) is acting solely for the client and is not responsible to any other party. Deliverables are valid for and should be used solely in connection with the purpose for which they were prepared as set out in our engagement agreement. You should not refer to or use our name or advice for any other purpose. The information (where appropriate) has not been verified. No representation or warranty is given as to accuracy, completeness or correctness of information in the Deliverables, any document, or any other information made available. Deliverables are for the internal use of the client and may not be used or relied upon by any person or entity other than the client. Deliverables are confidential and are not to be provided, without our authorization (preferably written), to entities or representatives of entities (including employees) that are not the client, including affiliates or representatives of affiliates of the client.