



PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Pangolin

08 April 2025



paladinsec.co



info@paladinsec.co

Table of Contents

Table of Contents	2
Disclaimer	3
1 Overview	4
1.1 Summary	4
1.2 Contracts Assessed	5
1.3 Findings Summary	6
1.3.1 Global	7
1.3.2 PangolinV3Factory	7
1.3.3 PangolinV3Pool	7
1.3.4 NonfungiblePositionManager	8
2 Findings	9
2.1 Global Issues	9
2.1.1 Issues & Recommendations	10
2.2 PangolinV3Factory	12
2.2.1 Privileged Functions	12
2.2.2 Issues & Recommendations	13
2.3 PangolinV3Pool	14
2.3.1 Privileged Functions	14
2.3.2 Issues & Recommendations	15
2.4 NonfungiblePositionManager	20
2.4.1 Privileged Functions	20
2.4.2 Issues & Recommendations	21

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocacy for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. Paladin is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. Paladin is furthermore allowed to claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for Pangolin Exchange on the Avalanche network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	Pangolin Exchange
URL	https://www.pangolin.exchange/
Platform	Avalanche
Language	Solidity
Preliminary Contracts	https://github.com/pangolindex/exchange-contracts/commit/a3c771f6846d381b84b4b09e0efe374f51c136dc
Resolution #1	https://github.com/pangolindex/exchange-contracts/commit/352178b687ac7e75dfe430635036e431e48262fc
Resolution #2	https://github.com/pangolindex/exchange-contracts/commit/2a6f9e49268417f09c2845c080f060a98ee547ba
Resolution #3	https://github.com/pangolindex/exchange-contracts/commit/88a657259b07d37d35125ee711fa1808b40c964c

1.2 Contracts Assessed

Name	Contract	Live Code Match
PangolinV3Factory	0x1128F23D0bc0A8396E9FBC3c0c68f5EA228B8256	✓ MATCH
PangolinV3Pool	Deployed by PangolinV3Factory	✓ MATCH
NonfungiblePositionMa nager	0xf40937279F38D0c1f97aFA5919F1cB3cB7f06A7F	✓ MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● Governance	1	1	-	-
● High	1	1	-	-
● Medium	3	1	-	2
● Low	2	1	-	1
● Informational	4	1	-	3
Total	11	5	-	6

Classification of Issues

Severity	Description
● Governance	Issues under this category are where the governance or owners of the protocol have certain privileges that users need to be aware of, some of which can result in the loss of user funds if the governance's private keys are lost or if they turn malicious, for example.
● High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
● Medium	Bugs or issues that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
● Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
● Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 Global

ID	Severity	Summary	Status
01	MEDIUM	The fee parameter used in the swap path and for gathering specific pools no longer guarantees returning a pool which uses that fee	ACKNOWLEDGED

1.3.2 PangolinV3Factory

ID	Severity	Summary	Status
02	GOV	No limitations placed on altering the fees of any pool	✓ RESOLVED
03	INFO	Use of immutable clones rather than deploying pools as separate contracts will result in higher gas costs over time	ACKNOWLEDGED

1.3.3 PangolinV3Pool

ID	Severity	Summary	Status
04	HIGH	Changing the tickSpacing when the fee for a pool is changed will break core invariants, and can lead to lost funds	✓ RESOLVED
05	MEDIUM	Users who mint directly from the PangolinV3Pool contract will not receive any token rewards	ACKNOWLEDGED
06	LOW	Reward rate delta can be incorrect when the timestamp has surpassed <code>type(uint32).max</code>	ACKNOWLEDGED
07	LOW	<code>setFee</code> does not check that <code>tickSpacing</code> is registered in the factory	✓ RESOLVED
08	INFO	<code>initialize()</code> is not protected for the <code>PangolinV3Pool</code> implementation contract	ACKNOWLEDGED

1.3.4 NonfungiblePositionManager

ID	Severity	Summary	Status
09	MEDIUM	Users who <code>mint()</code> a new position and never alter their liquidity will never receive any token rewards	✓ RESOLVED
10	INFO	Lack of an external function to trigger <code>_updateReward()</code> without altering liquidity	✓ RESOLVED
11	INFO	Gas optimizations	ACKNOWLEDGED

2 Findings

2.1 Global Issues

The issues listed in this section apply to the protocol as a whole. Please read through them carefully and take care to apply the fixes across the relevant contracts.

Comparison with UniswapV3

Aside from the custom changes to the contracts mentioned in their respective sections, the rest of the contracts were adjusted to match the Pangolin name and pragma version was set to be between $\geq 0.5.0 < 0.8.0$.



2.1.1 Issues & Recommendations

Issue #01	The fee parameter used in the swap path and for gathering specific pools no longer guarantees returning a pool which uses that fee
-----------	------------------------------------------------------------------------------------------------------------------------------------

Severity

 MEDIUM SEVERITY

Description

In the original Uniswap V3 implementation, each pool has a fixed fee, meaning that if you look up a pool in the `PangolinV3Factory` contract based on a specific fee with `getPool()`, it is guaranteed to return a pool which has this fee set. This is no longer the case for Pangolin V3, where the fee can be altered at any time. This changes a lot of assumptions about external interactions with this contract, where the new design is highly unintuitive.

As another example, the `SwapRouter` contract generally takes in a path where a fee value is specified to denote which pool to swap through. With this change, the value of this fee does not necessarily correspond with the actual fee charged in that pool.

One more example — `computeAddress()` generally uses the fee to compute the address.

```
function computeAddress(address factory, PoolKey memory key)
internal pure returns (address pool) {
    require(key.token0 < key.token1);
    pool = address(
        uint256(
            keccak256(
                abi.encodePacked(
                    hex'ff',
                    factory,
                    keccak256(abi.encode(key.token0,
key.token1, key.fee)),
                    POOL_INIT_CODE_HASH
                )
            )
        )
    );
}
```

This means that if a pool is deployed with a fee of X and then the fee is changed to another value (e.g. Y), poolKey would change even though the pool's address remains the same based on the fee at the point of deployment.

If the pool key now uses Y as the fee and not X, it would result in an incorrect computeAddress being returned, and thus verifyCallback would fail.

Recommendation At a minimum, consider making it much clearer that the fee value passed in these functions does not correspond to the current fee of the pool in the documentation and the codebase. However, we also believe the functionality for looking up pools should be altered to make it more clear that pools can have altered fees.

Users may implicitly be expecting a similar experience with Uniswap V3 given the foundation of this codebase.

Resolution

 ACKNOWLEDGED



2.2 PangolinV3Factory



PangolinV3Factory is a fork of the UniswapV3Factory contract with the following differences:



1. Different initial pool tick spacing / fee amount:
 - 100 fee 1 spacing
 - 2500 fee 60 spacing
 - 8000 fee 200 spacing
2. Deterministic deployment of pools using clones rather than having a separate contract deployed per pool.
3. New interface functions for collecting protocol fees (`collectProtocol()`) and setting both the admin and pool fees (`setFeeProtocol()`, `setFee()`). This effectively enabled dynamic fee adjustments, wherein the admin can alter fees at any point in time for any pool.

2.2.1 Privileged Functions

- `collectProtocol`
- `setFeeProtocol`
- `setFee`
- `enableFeeAmount`

2.2.2 Issues & Recommendations

Issue #02	No limitations placed on altering the fees of any pool
Severity	 GOVERNANCE
Description	There are currently no limitations placed on the admin to restrict them from being able to freely alter the fee. This means that the admin can update the fee for any pool in perpetuity, which may not be in the best interest or experience of users.
Recommendation	Consider implementing certain limitations on setting the fee, such as allowing for a reasonable time lock prior to the value being updated, or having functionality in which the admin can revoke their ability to update the fee for pools.
Resolution	 RESOLVED The ownership has been transferred to a multi-signature contract.

Issue #03	Use of immutable clones rather than deploying pools as separate contracts will result in higher gas costs over time
Severity	 INFORMATIONAL
Description	<p>The use of the proxy pattern instead of the original method of deploying each pool as a separate contract will result in additional gas costs over time as each (for example) swap call will require two hops rather than one compared to a normal smart contract.</p> <p>While it is true that deploying pools require less gas, but over time, each operation for the pool will require additional gas.</p>
Recommendation	We recommend deploying pools using the same method as the original Uniswap V3 implementation.
Resolution	 ACKNOWLEDGED

2.3 PangolinV3Pool

PangolinV3Pool is a fork of the UniswapV3Pool contract with the following differences:


1. PangolinV3Pool is an implementation contract for the deterministic clones proxy with an `initialize()` function.
2. New `setFee()` function which allows the admin to alter the fee on swaps for the pool.
3. New reward functionality which provides rewards per user using similar math as the fee growth calculation.

2.3.1 Privileged Functions

- `setFeeProtocol`
- `collectProtocol`
- `setRewardRate`
- `setFee`



2.3.2 Issues & Recommendations

Issue #04	Changing the tickSpacing when the fee for a pool is changed will break core invariants, and can lead to lost funds
Severity	 HIGH SEVERITY
Description	<p>The current implementation changes a pool's fee when its tickSpacing is changed. This can be seen in the <code>setFee()</code> implementation of <code>PangolinV3Factory</code>. This breaks core invariants of the <code>UniswapV3</code> implementation, in which we will provide an example.</p> <p>Consider that there is a pool in which the tickSpacing for the initial fee is 10. There are two positions with liquidity—one from tick -10 to 0 and one from tick -60 to 0. Now assume that based on the fee change for this pool, the new tickSpacing is 60. The position covers tick -60 to 0. We assume that some swaps have occurred in this range, and now the current tick is > 0.</p> <p>This means that tick 0 will have up to date <code>feeGrowthOutside0X128</code> and <code>feeGrowthOutside1X128</code> values. Notice that since tick -10 is no longer accessed based on the tickSpacing of 60, its values for <code>feeGrowthOutside0X128</code> and <code>feeGrowthOutside1X128</code> are smaller than they should be, given they have not been updated even though in practice they have been crossed.</p> <p>When the current tick is greater than the upper tick for a position, we can calculate the fees owed to that position as <code>UpperTick.feeGrowthOutside1X128 - LowerTick.feeGrowthOutside1X128</code>. Intuitively, we can see now that the -10 to 0 tick position will have inflated fee accrual since the lower tick at -10 have not had their <code>feeGrowthOutside0X128</code> or <code>feeGrowthOutside1X128</code> values updated.</p>
Recommendation	Remove all logic for changing the tickSpacing of already initialized pools.

Resolution



`setFee()` of `PangolinV3Factory` was modified so that `tickSpacing` is not altered anymore. However, there is no input validation for the `setFee()` function — values greater than `1e6` can break core math.

In the second resolution round, an error was introduced where a pool can still be deployed with more than `setFee` limit only when `setFee` has that limitation.

This issue was resolved in the third resolution round.

Issue #05

Users who mint directly from the `PangolinV3Pool` contract will not receive any token rewards

Severity



Description

Only users who use the `NonfungiblePositionManager` contract for minting and managing their liquidity will be able to receive token rewards. This is because reward calculation and credit are handled exclusively by this contract.

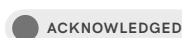
This differs from the behavior for collecting swap fees, where users who mint either from the pool directly or the `NonfungiblePositionManager` contract will still be able to collect their fees.


There is also a side effect of this in which minting directly from the pool still contributes to the total liquidity, meaning it would affect `rewardPerLiquidityCumulativeX64`, specifically diluting its value.

Recommendation

Assuming this is the intended logic, make sure to properly document this behavior. Otherwise, a fix for this is not exactly trivial and might involve either only allowing minting vs the `NonfungiblePositionManager` contract or somehow taking into account only liquidity created via the manager.

Resolution



Issue #06**Reward rate delta can be incorrect when the timestamp has surpassed `type(uint32).max`****Severity** LOW SEVERITY**Description**


The logic for setting the cumulative rewards per liquidity in part occurs in the `Oracle.transform()` call, which is triggered from this contract. This function has the following logic:

```
uint32 rewardRateEffectiveDelta = block.timestamp >
type(uint32).max
    ? 0
    : rewardRateEffectiveUntil < block.timestamp
    ? rewardRateEffectiveUntil <= last.blockTimestamp
        ? 0
        : rewardRateEffectiveUntil - last.blockTimestamp
    : delta;
```

This implementation currently does not correctly handle the edge case when the current timestamp is greater than `type(uint32).max` and `rewardRateEffectiveUntil > last.blockTimestamp`. This is because it defaults to setting the delta equal to 0 just based on the current `block.timestamp`.

Recommendation

Consider adding this edge case in this ternary operation.

Resolution ACKNOWLEDGED

Issue #07**setFee does not check that tickSpacing is registered in the factory****Severity** LOW SEVERITY**Description**

v3 factory's setFee checks that the tickSpacing is registered in the factory:

```
function setFee(address _pool, uint24 _fee) external {
    require(msg.sender == owner, "AUTH");
    int24 tickSpacing = feeAmountTickSpacing[_fee];
    require(tickSpacing != 0, "ERRTICK");
    IPangolinV3Pool(_pool).setFee(_fee, tickSpacing);
}
```

However, the same check is not done in v3 pool. It is fine if the call is only allowed through the factory, but onlyFactoryOwner does not only allow the factory as the caller, but also the factory's owner.

```
function setFee(uint24 _fee, int24 _tickSpacing) external
override onlyFactoryOwner {
    uint24 oldfee = fee;
    int24 oldtickspacing = tickSpacing;
    fee = _fee;
    tickSpacing = _tickSpacing;
    emit SetFee(oldfee, fee, oldtickspacing, tickSpacing);
}
```

```
modifier onlyFactoryOwner() {
    require(msg.sender == factory || msg.sender ==
    IPangolinV3Factory(factory).owner());
    _;
}
```



Thus, it is possible for the owner to call pool.setFee(fee, wrongTickSpacing).

Recommendation

Consider checking that the tick spacing is registered in the factory in the pool itself.

Resolution RESOLVED

tickSpacing is no longer altered in the setFee() function.

Issue #08	<code>initialize()</code> is not protected for the PangolinV3Pool implementation contract
Severity	 INFORMATIONAL
Description	It is good practice to prevent <code>initialize()</code> or equivalent functions from being called for the implementation contract. There is nothing preventing them from being called here.
Recommendation	Consider adding a modifier which prevents calling these functions for the implementation contract.
Resolution	 ACKNOWLEDGED



2.4 NonfungiblePositionManager

NonfungiblePositionManager is a fork of the NonfungiblePositionManager with the added functionality to handle the rewards calculation and distribution for users. The rewards distribution involves an external integration / call with the PangolinRewarder to actually collect the rewards.

2.4.1 Privileged Functions

- updateTokenDescriptor
- updateRewardManager



2.4.2 Issues & Recommendations

Issue #09	Users who <code>mint()</code> a new position and never alter their liquidity will never receive any token rewards
------------------	--------------------------------------------------------------------------------------------------------------------------

Severity

 MEDIUM SEVERITY

Description

When a user creates a new position, it will create and store their `Position` struct with `rewardPerLiquidityInsideLastX64`, `rewardLastUpdated`, `rewardLastCollected`, and `rewardOwed` equal to 0. This creates a major issue when we look at the implementation of `_updateReward()` which defines the logic for determining how much rewards a user is owed:

```
if (position.rewardLastUpdated != 0) {  
    position.rewardOwed += FullMath.mulDiv(  
        rewardPerLiquidityInsideCurrentX64 -  
        position.rewardPerLiquidityInsideLastX64,  
        position.liquidity,  
        2**64  
    );  
}  
position.rewardPerLiquidityInsideLastX64 =  
rewardPerLiquidityInsideCurrentX64;
```

When the `rewardLastUpdated` value is 0, any rewards owed to the user will never be given to that user even if their liquidity has been active for some period of time.

`_updateReward()` is only triggered by `decreaseLiquidity()` or `increaseLiquidity()`, meaning you have to explicitly update the liquidity of your position prior to being able to accrue any fees. If you never do this then you will never accrue any fees.

Recommendation

Consider calling `_updateReward()` on the new position in order to set its `rewardPerLiquidityInsideLastX64` value.

Resolution



`_updateReward()` was added to `mint()`. However, there is another issue where `mint()` is called prior to there being any rewards set for the pool. In such a scenario, `rewardPerLiquidityInsideCurrentX64` inside the `this` check will be 0:

```
if (position.rewardPerLiquidityInsideLastX64 >=
    rewardPerLiquidityInsideCurrentX64) {
    return;
}
```

The `_updateReward()` call ends up returning prematurely and `position.rewardLastUpdated` never being set to non-zero, resulting in the same issue of the user not accumulating rewards once rewards are initialized for the pool.

In the second resolution round, a critical error was introduced where rewards were not being distributed correctly.

This appears to have been fixed in the third resolution round but we advise the team to perform more testing for edge cases.

Issue #10

Lack of an external function to trigger `_updateReward()` without altering liquidity

Severity



Description

`_updateReward()` is only called if you alter a position using `decreaseLiquidity()` or `increaseLiquidity()`. This is not ideal because this call needs to be triggered prior to calling `claimReward()`.

Recommendation

Consider adding an external function for triggering `_updateReward()` or adding it to the `claimReward()` call.

Resolution



Issue #11 Gas optimizations	
Severity	<div><div></div> INFORMATIONAL</div>
Description	It makes more sense to not load the entire Position struct into memory for the positions() and positionReward() function calls as not all the fields are used. Instead, use a storage pointer.
Recommendation	Consider implementing the above mentioned recommendations.
Resolution	<div><div></div> ACKNOWLEDGED</div>





PALADIN
BLOCKCHAIN SECURITY