Omniscia

SMART CONTRACT AUDITS

Smart Contract Security Assessment

08-1-2022

Prepared for
pangolindex

Online Report
pangolin-fee-based-exchange-router-specialized-token

Fee Based Exchange Router Security Audit

Audit Overview

We were tasked with performing an audit of the Pangolin Exchange codebase and in particular their new Uniswap-V2 compliant Pangolin Router supporting a specialized partner-fee structure for all trades performed via it.

Over the course of the audit, we identified a potential mis-assessment of how the fee structure is imposed that we have classified with `unknown` severity as it may ultimately be desirable functionality by the Pangolin team.

We advise the Pangolin team to closely evaluate all minor-and-above findings identified in the report and promptly remediate them as well as consider all optimizational exhibits identified in the report.

Post-Audit Conclusion

The Pangolin team evaluated all exhibits identified in the report and provided us with either acknowledgement or changes in the form of a commit in the GitHub repository within scope of the audit.

The fee-related finding has been addressed as "acknowledged" by the Pangolin team given that they wish to retain the current behaviour in place. We advise them to monitor fluctuations in fees to ensure that they fit their business purpose in the long run.

With regards to one of the gas optimization exhibits, the Pangolin team mis-assessed our advice and we advise them to reconsider PRS-01C as they can potentially optimize the codebase one step further.

Contracts Assessed

Files in Scope	Repository	Commit(s)
PangolinLibrary8.sol (PL8)	exchange-contracts	f24bb065a4, 93b3e0115a
PangolinRouterSupportingFees.sol (PRS)	exchange-contracts	f24bb065a4, 93b3e0115a

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	1	0	0	1
Informational	3	2	0	1
Minor	1	1	0	0
Medium	0	0	0	0
Major	0	0	0	0

During the audit, we filtered and validated a total of **1 findings utilizing static analysis** tools as well as identified a total of **4 findings during the manual review** of the codebase. We strongly recommend that any minor severity or higher findings are dealt with promptly prior to the project's launch as they introduce potential misbehaviours of the system as well as exploits.

The list below covers each segment of the audit in depth and links to the respective chapter of the report:

Compilation

The project utilizes `hardhat` as its development pipeline tool, containing an array of tests and scripts coded in JavaScript.

To compile the project, the `compile` command needs to be issued via the `npx` CLI tool to `hardhat`:

```bash
npx hardhat compile
```

The `hardhat` tool automatically selects Solidity version `0.8.13` based on the version specified within the `hardhat.config.js` file in conjunction with the `pragma` statements of the contracts in scope.

The project contains discrepancies with regards to the Solidity version used, however, they are solely contained in dependencies and can thus be safely ignored.

The `pragma` statements have been locked to `0.8.13` (`=0.8.13`), the same version utilized for our static analysis as well as optimizational review of the codebase.

During compilation with the `hardhat` pipeline, no errors were identified that relate to the syntax or bytecode size of the contracts.

Static Analysis

The execution of our static analysis toolkit identified **30 potential issues** within the codebase of which **27 were ruled out to be false positives** or negligible findings.

The remaining **3 issues** were validated and grouped and formalized into the **1 exhibits** that follow:

ID	Severity	Addressed	Title
PRS-01S	Minor	Yes	Inexistent Sanitization of Input Addresses

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Pangolin's new exchange router.

As the project at hand implements an exchange router, intricate care was put into ensuring that the **flow of funds within the system conforms to the specifications and restrictions** laid forth within the protocol's specification.

We validated that **all state transitions of the system occur within sane criteria** and that all rudimentary formulas within the system execute as expected. We **pinpointed a potential discrepancy in the fee system** which can have **minor ramifications to user experience** that we advise the Pangolin team to evaluate.

Additionally, the system was investigated for any other commonly present attack vectors such as re-entrancy attacks, mathematical truncations, logical flaws and **ERC / EIP** standard inconsistencies. The documentation of the project was satisfactory to the extent it need be.

A total of **4 findings** were identified over the course of the manual review of which **1 findings** concerned the behaviour and security of the system. The non-security related findings, such as optimizations, are included in the separate **Code Style** chapter.

The finding table below enumerates all these security / behavioural findings:

ID	Severity	Addressed	Title
PRS-01M	Unknown	Acknowledged	Inconsistent Fee Application

PangolinRouterSupportingFees Static Analysis Findings

PRS-01S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Minor	PangolinRouterSupportingFees.sol:L46-L50

Description:

The linked function(s) accept `address` arguments yet do not properly sanitize them.

Impact:

The presence of zero-value addresses, especially in `constructor` implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

Example:

contracts/pangolin-periphery/PangolinRouterSupportingFees.sol

SOL

```sol
46 constructor(address _FACTORY, address _WAVAX, address firstOwner) public {
47     FACTORY = _FACTORY; // 0xefa94DE7a4656D787667C749f7E1223D71E9FD88
48     WAVAX = _WAVAX; // 0xB31f66AA3C1e785363F0875A1B74E27b85FD66c7
49     transferOwnership(firstOwner);
50 }
```

Recommendation:

We advise some basic sanitization to be put in place by ensuring that each `address` specified is non-zero.

Alleviation:

All input addresses are now properly sanitized as advised, alleviating this exhibit.

PangolinRouterSupportingFees Manual Review Findings

PRS-01M: Inconsistent Fee Application

Type	Severity	Location
Mathematical Operations	Unknown	PangolinRouterSupportingFees.sol:L129, L153, L182, L206, L240, L265, L322, L349, L376

Description:

The `PangolinRouterSupportingFees` implementation attempts to come up with an exchange fee model that is applied on each swap operation either applied on the final output based on a fixed input or "added" to the expected output to offset the input the user needs to provide. This type of operation is discrepant, however, as a Uniswap V2 exchange pair represents a curve whereby an upward movement in the curve (increasing the total output) is not identical to a downward movement in the curve (decreasing the total output) depending on which portion of the curve's slope the pair's reserves are currently at.

Example:

contracts/pangolin-periphery/PangolinRouterSupportingFees.sol

SOL

```
111 function swapExactTokensForTokens(
112     uint256 amountIn,
113     uint256 amountOutMin,
114     address[] calldata path,
115     address to,
116     uint256 deadline,
117     address feeTo
118 ) external ensure(deadline) returns (uint256[] memory amounts) {
119     FeeInfo storage feeInfo = feeInfos[feeTo];
120     require(feeInfo.initialized, "Invalid partner");
121 
122     amounts = PangolinLibrary8.getAmountsOut(FACTORY, amountIn, path);
123 
124     uint256 feeTotalAmount;
125     uint256 userAmountOut;
```

```
    uint256 userAmountOut;

    { // Scope amountOut locally
        uint256 amountOut = amounts[amounts.length - 1];
        feeTotalAmount = amountOut * feeInfo.feeTotal / BIPS;
        userAmountOut = amountOut - feeTotalAmount;
    }

    require(userAmountOut >= amountOutMin, "INSUFFICIENT_OUTPUT_AMOUNT");

    TransferHelper.safeTransferFrom(
        path[0], msg.sender, PangolinLibrary8.pairFor(FACTORY, path[0], pat
    );

    _swap(amounts, path);
    _distribute(userAmountOut, path[path.length - 1], to, feeTo, feeInfo.fe
}
function swapTokensForExactTokens(
    uint256 amountOut,
    uint256 amountInMax,
    address[] calldata path,
    address to,
    uint256 deadline,
    address feeTo
) external ensure(deadline) returns (uint256[] memory amounts) {
    FeeInfo storage feeInfo = feeInfos[feeTo];
    require(feeInfo.initialized, "Invalid partner");

    uint256 feeTotalAmount = amountOut * feeInfo.feeTotal / BIPS;

    // Adjust amountOut to include fee
    amounts = PangolinLibrary8.getAmountsIn(FACTORY, amountOut + feeTotalAm
    uint256 amountIn = amounts[0];
    require(amountIn <= amountInMax, "EXCESSIVE_INPUT_AMOUNT");

    TransferHelper.safeTransferFrom(
        path[0], msg.sender, PangolinLibrary8.pairFor(FACTORY, path[0], pat
    );

    _swap(amounts, path);
    _distribute(amountOut, path[path.length - 1], to, feeTo, feeInfo.feeCut
}
```

Recommendation:

This behaviour will naturally increase the fees that are imposed for identical amounts depending on the current reserves and can be classified as an acknowledged trade-off of the currently simplified fee structure implementation. Alternatively, a fee could be applied on the input token rather than the output amount instead ensuring the same movement on the slope and thus identical fees regardless of the pair's reserves. Either an acknowledgement or a different fee structure can be considered valid alleviations to this exhibit.

Alleviation:

The Pangolin team assessed the impact of this exhibit and ultimately decided to acknowledge this behaviour as desirable and will monitor it for potential future adjustments.

Finding Types

A description of each finding type included in the report can be found below and is linked by each respective finding. A full list of finding types Omniscia has defined will be viewable at the central audit methodology we will publish soon.

External Call Validation

Many contracts that interact with DeFi contain a set of complex external call executions that need to happen in a particular sequence and whose execution is usually taken for granted whereby it is not always the case. External calls should always be validated, either in the form of `require` checks imposed at the contract-level or via more intricate mechanisms such as invoking an external getter-variable and ensuring that it has been properly updated.

Input Sanitization

As there are no inherent guarantees to the inputs a function accepts, a set of guards should always be in place to sanitize the values passed in to a particular function.

Indeterminate Code

These types of issues arise when a linked code segment may not behave as expected, either due to mistyped code, convoluted `if` blocks, overlapping functions / variable names and other ambiguous statements.

Language Specific

Language specific issues arise from certain peculiarities that the Solidity language boasts that discerns it from other conventional programming languages. For example, the EVM is a 256-bit machine meaning that operations on less-than-256-bit types are more costly for the EVM in terms of gas costs, meaning that loops utilizing a `uint8` variable because their limit will never exceed the 8-bit range actually cost more than redundantly using a `uint256` variable.

Code Style

An official Solidity style guide exists that is constantly under development and is adjusted on each new Solidity release, designating how the overall look and feel of a codebase should be. In these types of findings, we identify whether a project conforms to a particular naming convention and whether that convention is consistent within the codebase and legible. In case of inconsistencies, we point them out under this category. Additionally, variable shadowing falls under this category as well which is identified when a local-level variable contains the same name as a contract-level variable that is present in the inheritance chain of the local execution level's context.

Gas Optimization

Gas optimization findings relate to ways the codebase can be optimized to reduce the gas cost involved with interacting with it to various degrees. These types of findings are completely optional and are pointed out for the benefit of the project's developers.

Standard Conformity

These types of findings relate to incompatibility between a particular standard's implementation and the project's implementation, oftentimes causing significant issues in the usability of the contracts.

Mathematical Operations

In Solidity, math generally behaves differently than other programming languages due to the constraints of the EVM. A prime example of this difference is the truncation of values during a division which in turn leads to loss of precision and can cause systems to behave incorrectly when dealing with percentages and proportion calculations.

Logical Fault

This category is a bit broad and is meant to cover implementations that contain flaws in the way they are implemented, either due to unimplemented functionality, unaccounted-for edge cases or similar extraordinary scenarios.

Centralization Concern

This category covers all findings that relate to a significant degree of centralization present in the project and as such the potential of a Single-Point-of-Failure (SPoF) for the project that we urge them to re-consider and potentially omit.

Reentrant Call

This category relates to findings that arise from re-entrant external calls (such as EIP-721 minting operations) and revolve around the inapplicacy of the Checks-Effects-Interactions (CEI) pattern, a pattern that dictates checks (`require` statements etc.) should occur before effects (local storage updates) and interactions (external calls) should be performed last.

Disclaimer

The following disclaimer applies to all versions of the audit report produced (preliminary / public / private) and is in effect for all past, current, and future audit reports that are produced and hosted under Omniscia:

IMPORTANT TERMS & CONDITIONS REGARDING OUR SECURITY AUDITS/REVIEWS/REPORTS AND ALL PUBLIC/PRIVATE CONTENT/DELIVERABLES

Omniscia ("Omniscia") has conducted an independent security review to verify the integrity of and highlight any vulnerabilities, bugs or errors, intentional or unintentional, that may be present in the codebase that were provided for the scope of this Engagement.

Blockchain technology and the cryptographic assets it supports are nascent technologies. This makes them extremely volatile assets. Any assessment report obtained on such volatile and nascent assets may include unpredictable results which may lead to positive or negative outcomes.

In some cases, services provided may be reliant on a variety of third parties. This security review does not constitute endorsement, agreement or acceptance for the Project and technology that was reviewed. Users relying on this security review should not consider this as having any merit for financial advice or technological due diligence in any shape, form or nature.

The veracity and accuracy of the findings presented in this report relate solely to the proficiency, competence, aptitude and discretion of our auditors. Omniscia and its employees make no guarantees, nor assurance that the contracts are free of exploits, bugs, vulnerabilities, deprecation of technologies or any system / economical / mathematical malfunction.

This audit report shall not be printed, saved, disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Omniscia.

All the information/opinions/suggestions provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report.

Information in this report is provided 'as is'. Omniscia is under no covenant to the completeness, accuracy or solidity of the contracts reviewed. Omniscia's goal is to help reduce the attack vectors/surface and the high level of variance associated with utilizing new and consistently changing technologies.

Omniscia in no way claims any guarantee, warranty or assurance of security or functionality of the technology that was in scope for this security review.

In no event will Omniscia, its partners, employees, agents or any parties related to the design/creation of this security review be ever liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this security review.

Cryptocurrencies and all other technologies directly or indirectly related to cryptocurrencies are not standardized, highly prone to malfunction and extremely speculative by nature. No due diligence and/or safeguards may be insufficient and users should exercise maximum caution when participating and/or investing in this nascent industry.

The preparation of this security review has made all reasonable attempts to provide clear and actionable recommendations to the Project team (the “client”) with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts in scope for this engagement.

All services, the security reports, discussions, work product, attack vectors description or any other materials, products or results of this security review engagement is provided "as is" and "as available" and with all faults, uncertainty and defects without warranty or guarantee of any kind.

Omniscia will assume no liability or responsibility for delays, errors, mistakes, or any inaccuracies of content, suggestions, materials or for any loss, delay, damage of any kind which arose as a result of this engagement/security review.

Omniscia will assume no liability or responsibility for any personal injury, property damage, of any kind whatsoever that resulted in this engagement and the customer having access to or use of the products, engineers, services, security report, or any other other materials.

For avoidance of doubt, this report, its content, access, and/or usage thereof, including any associated services or materials, shall not be considered or relied upon as any form of financial, investment, tax, legal, regulatory, or any other type of advice.