Pendle V2

May 24, 2022

by Ackee Blockchain

Contents

1. Document Revisions

0.1	Draft report	May 24, 2022

2. Overview

This document presents our findings in reviewed contracts.

2.1. Ackee Blockchain

Ackee Blockchain is an auditing company based in Prague, Czech Republic, specialized in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge – we run a free certification course Summer School of Solidity and teach at the Czech Technical University in Prague. Ackee Blockchain is backed by the largest VC fund focused on blockchain and DeFi in Europe, Rockaway Blockchain Fund.

2.2. Audit Methodology

1. **Technical specification/documentation** - a brief overview of the system is requested from the client and the scope of the audit is defined.
2. **Tool-based analysis** - deep check with automated Solidity analysis tools and Slither is performed.
3. **Manual code review** - the code is checked line by line for common vulnerabilities, code duplication, best practices and the code architecture is reviewed.
4. **Local deployment + hacking** - the contracts are deployed locally and we try to attack the system and break it.
5. **Unit and fuzzy testing** - run unit tests to ensure that the system works as expected, potentially write missing unit or fuzzy tests.

2.3. Review team

Member’s Name	Position
Jan Kalivoda	Lead Auditor
Jan Smolik	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.4. Disclaimer

We’ve put our best effort to find all vulnerabilities in the system, however our findings shouldn’t be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

3. Executive Summary

Pendle V2 is a DeFi protocol based on Ethereum and Avalanche. It allows users to tokenize and trade the yield of yield generating mechanisms.

Pendle engaged Ackee Blockchain to conduct a security review of Pendle V2 with a total time donation of 4 engineering weeks. The review took place between April 25 and May 20, 2022.

The scope included the following repository with a given commit:

- pendle-core-internal-v2 - `9d93fc1`

All contracts under `contracts` folder was in-scope, except for the following:

- `core/PendleSCYImpl/AaveV3/WadRayMath.sol`
- `core/RouterStatic.sol`
- `libraries/ExpiryUtilsLib.sol`
- `libraries/JoeLibrary.sol`

We began our review by using static analysis tools and then took a deep dive into the logic of the contracts. During the review, we paid special attention to:

- checking if nobody can breach the protocol,
- checking the correctness of the upgradeability implementation,
- checking the arithmetics of Math libraries,
- ensuring access controls are not too relaxed,
- and looking for common issues such as data validation.

The code quality is very good in general. Tests are well written and with comprehensive coverage. We have received excellent documentation, including well-processed whitepapers on important components. The team always responded quickly.

Our review resulted in 11 findings, ranging from Informational to Medium severity.

Ackee Blockchain recommends Pendle to:

- address all reported issues.

4. System Overview

This section contains an outline of the audited contracts. Note that this is meant for understandability purposes and does not replace project documentation.

4.1. Contracts

Contracts we find important for better understanding are described in the following section.

SCYBase

SCYBase is a Pendle-provided basic logic for a SCY implementation, to be extended by actual implementations of different SCY tokens. It contains the basic logic to mint and redeem SCY tokens.

RewardManager

RewardManager keeps track of various reward tokens and is responsible for distributing these tokens to users. By inheriting from RewardManager, any contract can be easily extended to have this functionality.

SCYBaseWithRewards

SCYBaseWithRewards is a SCYBase extended by RewardManager. Users can claim their reward tokens by calling the `redeemReward` function.

When creating an actual SCY implementation of a yield generating asset, SCYBaseWithRewards ought to be used as a base contract if there are any reward tokens (on top of the interest). If there are no reward tokens, SCYBase is sufficient.

PendleYieldContractFactory

PendleYieldContractFactory is the factory contract to create new PendleYieldToken and PendlePrincipalToken contracts. It also keeps track of all yield contracts created and collects fees. `createYieldContract` is the core function that creates a new yield contract pair and needs SCY and expiry as an input.

PendleYieldToken

The contract for the yield token. It contains the logic to mint and redeem YT & PT from SCY and vice versa and all the logic to distribute the yield to the users (interest and reward tokens).

PendlePrincipalToken

The contract for the principal token. All the logic to `mint` and `burn` is in the PendleYieldToken.

PendleMarketFactory

The factory contract to create new PendleMarkets.

PendleMarket

PendleMarket is the market contract that allows the exchange of the PT and SCY. The users can swap one for the other and add or remove liquidity. The contract extensively uses Pendle's internally developed math libraries.

PendleRouter

PendleRouter is the contract users will interact with. It is responsible for swaps (mint and burn) between the base tokens, SCY, YT, and PT. It delegates logic to the Action contracts in the `core/actions` folder.

4.2. Actors

This part describes actors of the system, their roles, and permissions.

Governance

Besides publicly-accessible entrypoints, the governance can:

- authorize upgrade of the router,
- set expiry divisor, interest fee rate, and treasury address in PendleYieldContractFactory,
- set `lnFeeRateRoot` (responsible for a fee calculation), oracle time window, reserve fee percentage, and treasury address in PendleMarketFactory.

4.3. Trust model

Apart from Governance privileged access, there are no other essential roles from a trust perspective. Users have to trust Governance that they will not change the parameters of the existing market inconveniently (see Dynamic config issue).

The following figure marks dependencies between individual components. [1]

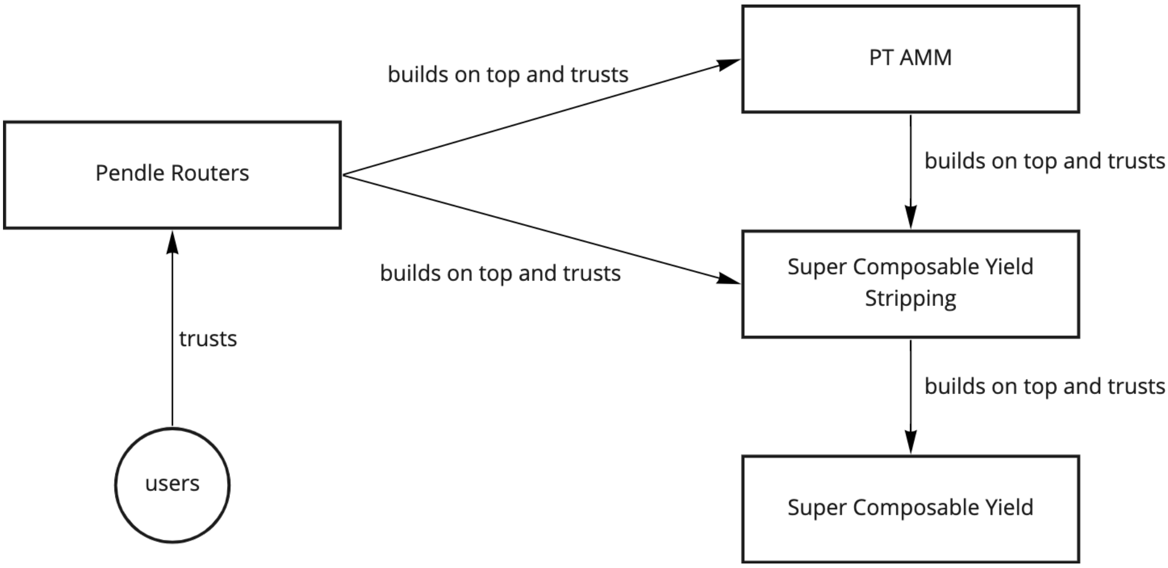

Figure 1. Trust model of components

[1] Source of the figure is a PendleV2 - Notes for auditors documentation

5. Vulnerabilities risk methodology

Each finding contains an *Impact* and *Likelihood* ratings.

If we have found a scenario in which the issue is exploitable, it will be assigned an impact of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't found a way, or the issue is only exploitable given a change in configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Informational*.

Low to *High* impact issues also have a *Likelihood* which measures the probability of exploitability during runtime.

5.1. Finding classification

The full definitions are as follows:

Impact

High

Code that activates the issue will lead to undefined or catastrophic consequences for the system.

Medium

Code that activates the issue will result in consequences of serious substance.

Low

Code that activates the issue will have outcomes on the system that are either recoverable or don't jeopardize its regular functioning.

Warning

The issue cannot be exploited given the current code and/or configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.), but could be a security vulnerability if these were to change slightly. If we haven't found a way to exploit the issue given the time constraints, it might be marked as "Warning" or higher, based on our best estimate of whether it is currently exploitable.

Informational

The issue is on the border-line between code quality and security. Examples include insufficient logging for critical operations. Another example is that the issue would be security-related if code or configuration (see above) was to change.

Likelihood

High

The issue is exploitable by virtually anyone under virtually any circumstance.

Medium

Exploiting the issue currently requires non-trivial preconditions.

Low

Exploiting the issue requires strict preconditions.

6. Findings

This section contains the list of discovered findings. Unless overriden for purposes of readability, each finding contains:

- a *Description*,
- an *Exploit scenario*, and
- a *Recommendation*

Many times, there might be multiple ways to solve or alleviate the issue, with varying requirements in terms of the necessary changes to the codebase. In that case, we will try to enumerate them all, making clear which solve the underlying issue better (albeit possibly only with architectural changes) than others.

Summary of Findings

	Type	Impact	Likelihood
M1: Insufficient data validation in PendleAaveV3SCY	Data validation	High	Low
M2: Integer overflow in Math library	Integer overflow	High	Low
M3: Usage of `solc` optimizer	Compiler configuration	High	Low
W1: Potential front-running of several withdraw and mint functions	Front-running	Warning	N/A
W2: Exotic tokens	Logic, Reentrancy	Warning	N/A

	Type	Impact	Likelihood
W3: Dangerous callbacks	Reentrancy, External calls	Warning	N/A
W4: Unintended change of the reentrancy lock state	Reentrancy	Warning	N/A
W5: Dynamic config potential inconsistency	Front-running	Warning	N/A
I1: Redundant cycle in RewardManager	Gas optimization	Informational	N/A
I2: Same function names across the project	Code quality	Informational	N/A
I3: Unused code	Dead code	Informational	N/A

Table 1. Table of Findings

M1: Insufficient data validation in PendleAaveV3SCY

Impact:	High	Likelihood:	Low
Target:	PendleAaveV3SCY	Type:	Data validation

Description

PendleAaveV3SCY does not perform any data validation of passed addresses in its constructor.

Exploit scenario

An incorrect or malicious `_rewardsController` is passed it. Instead of reverting, the call succeeds.

Recommendation

Add more stringent data validation for `_rewardsController`, `aToken` and `_aavePool`). At least, this would include a zero-address check.

Go back to Findings Summary

M2: Integer overflow in Math library

Impact:	High	Likelihood:	Low
Target:	Math	Type:	Integer overflow

Description

The project uses its own `Math` library. We have found two functions that may return an unexpected result: `subNoNeg` and `Int128`.

```
function subNoNeg(int256 a, int256 b) internal pure returns (int256) {
        require(a >= b, "NEGATIVE");
        unchecked {
            return a - b;
        }
    }
```

In `subNoNeg`, when `a` is a huge number and `b` is a negative number, the `require` statement will not revert, and `a - b` can overflow.

For example, `subNoNeg(int256_max, -1)` returns `int256_min`.

```
function Int128(int256 x) internal pure returns (int128) {
        require(x < (1 << 127)); // signed, lim = bit-1
        return int128(x);
    }
```

In `Int128`, there is a check that `x < (1 << 127)` (i.e. `uint256 x` is smaller than maximal value for `int128`). But what if `x` is smaller than the minimal value for `int128`? The `require` statement will not revert, and the result will be incorrect.

For example, `Int128(int128_min - 1)` returns `int128_max`.

Recommendation

Short term, the `subNoNeg` function needs more stringent validation, such as verification that `b >= 0`.

```
require(a >= b && b >= 0, "NEGATIVE");
```

This guarantees that both `a` and `b` are not negative, and the result is always correct. For the `Int128` function, add verification that the input variable `x` is within the allowed interval for `int128`.

Long term, try to avoid custom libraries and use known public ones which are battle-tested during their existence.

Go back to Findings Summary

M3: Usage of `solc` optimizer

Impact:	High	Likelihood:	Low
Target:	*	Type:	Compiler configuration

Description

The project uses `solc` optimizer. Enabling `solc` optimizer may lead to unexpected bugs.

The Solidity compiler was audited in November 2018, and the audit concluded that the optimizer may not be safe.

Vulnerability scenario

A few months after deployment, a vulnerability is discovered in the optimizer. As a result, it is possible to attack the protocol.

Recommendation

Until the `solc` optimizer undergoes more stringent security analysis, opt-out using it. This will ensure the protocol is resilient to any existing bugs in the optimizer.

Go back to Findings Summary

W1: Potential front-running of several withdraw and mint functions

Impact:	Warning	Likelihood:	N/A
Target:	*	Type:	Front-running

Description

Non-atomic interactions with components can lead to loss of funds. For example, if users bypass routers and interact with components directly.

Exploit scenario

Alice sends funds to PendleYieldToken contract and wants to call `mintPY` to gain PT and YT tokens from deposited SCY. Bob notices that the contract holds Alice's SCY and front-run her `mintPY` transaction, thus stealing her SCY.

Recommendation

Ensure that every honest user interacts with the router.

Go back to Findings Summary

W2: Exotic tokens

Impact:	Warning	Likelihood:	N/A
Target:	*	Type:	Logic, Reentrancy

Description

The protocol works with various external contracts (base tokens, assets, reward tokens, etc.).

- There are some situations in the codebase when token transfers are done in the middle of a state-changing function. If the tokens transferred have callbacks (e.g. all ERC223 and ERC777 tokens), this might create reentrancy possibilities.
- If the asset used in SCYBase is not a static token (i.e., its balance increases on its own - asset could be some kind of a rebase token), SCYBase can mistakenly mint more SCY for users depositing. When the pool earns more assets, `lastBalanceOf[asset]` is not updated, which means that in the next `mint` or `mintNoPull` call, `_afterReceiveToken` returns higher `amountBaseIn`.

Exploit scenario

An asset is a rebase token, and the pool has 1000 assets. Therefore, `lastBalanceOf[asset]` is 1000.

Bob wants to deposit 100 assets and get 100 SCY tokens (for simplicity, the ratio is 1:1).

Suddenly, a rebase happens, and now the pool has 1500 assets, but `lastBalanceOf[asset]` is not updated.

After that, Bob calls the `mint` function through the router and transfers 100 assets to the pool.

In `_mintFresh`, `_afterReceiveToken` says that Bob deposited 600 tokens. Hence Bob obtains 600 SCY tokens.

Recommendation

Ensure that no tokens with callbacks and no tokens that increase balances on their own are added.

Go back to Findings Summary

W3: Dangerous callbacks

Impact:	Warning	Likelihood:	N/A
Target:	PendleMarket	Type:	Reentrancy, External calls

Description

The following functions contain dangerous callback:

- addLiquidity
- removeLiquidity
- swapExactPtForScy
- swapScyForExactPt

These callbacks are dangerous because they are triggered in `msg.sender` context. The function's caller can be an arbitrary contract and thus an arbitrary external call or series of them. Moreover, this external call is performed **before** the state is written.

Listing 1. Dangerous callback

```
if (data.length > 0) {
    IPMarketAddRemoveCallback(msg.sender).addLiquidityCallback(
        lpToAccount,
        scyUsed,
        ptUsed,
        data
    );
}
```

Recommendation

We didn't find any specific exploit scenario because of the `nonReentrant` modifier usage. However, it could potentially lead to unknown consequences when new dependencies and functions are added in future development.

Go back to Findings Summary

W4: Unintended change of the reentrancy lock state

Impact:	Warning	Likelihood:	N/A
Target:	PendleMarket	Type:	Reentrancy

Description

The `MarketStorage` structure contains several variables including the reentrancy lock variable (`_reentrancyStatus`).

Listing 2. `MarketStorage` *contains reentrancy lock variable*

```
struct MarketStorage {
    int128 totalPt;
    int128 totalScy;
    // 1 SLOT = 256 bits
    uint96 lastLnImpliedRate;
    uint96 oracleRate;
    uint32 lastTradeTime;
    uint8 _reentrancyStatus;
    // 1 SLOT = 232 bits
}
```

With the following architecture of writing the state:

Listing 3. The function which resets the reentrancy lock

```
function _writeState(MarketState memory market) internal {
    MarketStorage memory tempStore;

    tempStore.totalPt = market.totalPt.Int128();
    tempStore.totalScy = market.totalScy.Int128();
    tempStore.lastLnImpliedRate = market.lastLnImpliedRate.Uint96();
    tempStore.oracleRate = market.oracleRate.Uint96();
    tempStore.lastTradeTime = market.lastTradeTime.Uint32();

    _storage = tempStore;

    emit UpdateImpliedRate(block.timestamp, market.lastLnImpliedRate);
}
```

Each time the state is written, the reentrancy lock is set to zero, and in the current context, the function could be entered again.

Although this is not a problem when `_writeState` is at the end of a function, it presents potential risks.

Recommendation

Change `_writeState` to preserve the current reentrancy lock state and not reset it.

Go back to Findings Summary

W5: Dynamic config potential inconsistency

Impact:	Warning	Likelihood:	N/A
Target:	PendleMarket	Type:	Front-running

Description

PendleMarket is using a dynamic config in its `readState` method.

```
function readState(bool updateRateOracle) public view returns (MarketState
memory market) {
    MarketStorage memory local = _storage;

    market.totalPt = local.totalPt;
    market.totalScy = local.totalScy;
    market.totalLp = totalSupply().Int();
    market.oracleRate = local.oracleRate;

    (
        market.treasury,
        market.lnFeeRateRoot,
        market.rateOracleTimeWindow,
        market.reserveFeePercent
    ) = IPMarketFactory(factory).marketConfig(); ①

    ...
```

① dynamic config

There is a possibility that a change of `marketConfig` can have an undesired impact on pending transactions where inputs were chosen according to the old state.

Recommendation

We haven't identified any critical scenarios. Only ones that affect the Trust Model include immediate change of fees (`lnFeeRateRoot`) or treasury address and so we decided to include it in the report.

Go back to Findings Summary

I1: Redundant cycle in RewardManager

Impact:	Informational	Likelihood:	N/A
Target:	RewardManager	Type:	Gas optimization

Description

In RewardManager there is a function `_initGlobalReward` to initialize the indexes of all reward tokens. The function loops through all reward tokens and sets their index to the initial value if the current index value equals zero.

```
function _initGlobalReward(address[] memory rewardTokens) internal virtual
{
        for (uint256 i = 0; i < rewardTokens.length; ++i) {
            if (globalReward[rewardTokens[i]].index == 0) {
                globalReward[rewardTokens[i]].index = INITIAL_REWARD_INDEX;
            }
        }
    }
```

When set, the index value will never go down to zero. Therefore, it makes sense to call this function only once.

However, this function is being called each time in `_updateGlobalReward` (lines 82-83):

```
address[] memory rewardTokens = getRewardTokens();
_initGlobalReward(rewardTokens);
```

`_updateGlobalReward` is called before every SCYBaseWithRewards token transfer (in `_beforeTokenTransfer` transfer hook), in every `redeemReward` and many times elsewhere.

Recommendation

If there is no specific reason to call `_initGlobalReward` each time, adjust the contract to calling it only once in the constructor or having a variable `bool _initialized` so that the function is called only once.

Go back to Findings Summary

I2: Same function names across the project

Impact:	Informational	Likelihood:	N/A
Target:	*	Type:	Code quality

Description

Several functions have identical name but different content or at least very similiar name, e.g. `swapScyForExactPt` exists in context of PendleMarket:

Listing 4. PendleMarket

```
function swapScyForExactPt(
    address receiver,
    uint256 exactPtOut,
    uint256 maxScyIn,
    bytes calldata data
) external nonReentrant returns (uint256 netScyIn, uint256 netScyToReserve)
{
    require(block.timestamp < expiry, "MARKET_EXPIRED");

    MarketState memory market = readState(true);

    (netScyIn, netScyToReserve) = market.swapScyForExactPt(
        SCYIndexLib.newIndex(SCY),
        exactPtOut,
        block.timestamp,
        true
    );
    require(netScyIn <= maxScyIn, "scy in exceed limit");

    IERC20(PT).safeTransfer(receiver, exactPtOut);
    IERC20(SCY).safeTransfer(market.treasury, netScyToReserve);

    if (data.length > 0) {
        IPMarketSwapCallback(msg.sender).swapCallback(exactPtOut.Int(),
netScyIn.neg(), data);
    }

    // have received enough SCY
    require(market.totalScy.Uint() <= IERC20(SCY).balanceOf(address(
this)));
    _writeState(market);

    emit Swap(receiver, exactPtOut.Int(), netScyIn.neg(), netScyToReserve);
}
```

and also in MarketMathAux.

Listing 5. MarketMathAux

```
function swapScyForExactPt(
    MarketState memory market,
    SCYIndex index,
    uint256 exactPtToAccount,
    uint256 blockTime,
    bool updateState
) internal pure returns (uint256 netScyToMarket, uint256 netScyToReserve) {
    (int256 _netScyToAccount, int256 _netScyToReserve) = MarketMathCore
.executeTradeCore(
        market,
        index,
        exactPtToAccount.Int(),
        blockTime,
        updateState
    );

    netScyToMarket = _netScyToAccount.neg().Uint();
    netScyToReserve = _netScyToReserve.Uint();
}
```

This approach can cause unknown bugs (e.g., by mistake) in future development.

Recommendation

Adjust the architecture of the project to prevent duplicities and unnecessary complexity.

[Go back to Findings Summary](#)

I3: Unused code

Impact:	Informational	Likelihood:	N/A
Target:	PendleMarket	Type:	Dead code

Description

Line 22 is unused.

```
using Math for uint128;
```

Recommendation

Remove unused or unnecessary code from the project.

Go back to Findings Summary

Endnotes

7. Appendix A

7.1. How to cite

Please cite this document as:

Ackee Blockchain, "Pendle V2", May 24, 2022.

If an individual issue is referenced, please use the following identifier:

`ABCH-{project_identifer}-{finding_id}`,

where `{project_identifier}` for this project is `PENDLE-V2` and `{finding_id}` is the id which can be found in Summary of Findings. For example, to cite H1 issue, we would use `ABCH-PENDLE-V2-H1`.

ackee blockchain

Thank You

Ackee Blockchain a.s.

Prague, Czech Republic

hello@ackeeblockchain.com

https://discord.gg/wpM77gR7en