Napier V2 Audit Report

2025-01-09

About

https://napier.finance/

https://xuwinnie.review/

Scope

https://github.com/napierfi/napier-v2/tree/winnie/src

*lens/** is not included*

Conclusion

During this audit, 3 high risk findings and 3 low risk findings were identified, all of which have been addressed accordingly.

Findings

High Risk

H-1 Yield calculation after maturity is invalid

```
function calcYield(uint256 prevMaxscale, uint256 maxscale, uint256 ytBalance)
internal pure returns (uint256) {
    if (prevMaxscale >= maxscale) return 0;
    if (prevMaxscale == 0) return 0;
    return ((ytBalance * (maxscale - prevMaxscale)) * 1e18) / (prevMaxscale *
maxscale);
}
```

Some underlying could have been redeemed after maturity and PT balance could be less than YT balance. Current formula will lead to insolvency, PT balance should be used instead.

Reaction: Fixed in https://github.com/napierfi/napier-v2/pull/232

H-2 `balanceBefore` should be determined by `tokenOut.isNative()`, not `tokenIn.isNative()`

In AggregationRouter.sol, `balanceBefore` refers to the balance of the output token before the swap.

Reaction: Fixed in https://github.com/napierfi/napier-v2/pull/222

H-3 Slippage control for `swapYtForAnyToken` is ineffective

```
    function _swapYtForUnderlying(
        TwoCrypto twoCrypto,
        PrincipalToken principalToken,
        address yt,
        uint256 principal,
        ApproxValue sharesDx // Off-chain estimation `get_dx` result
    ) internal returns (uint256 shares) {
        principal = twoCrypto.get_dy(TARGET_INDEX, PT_INDEX, sharesDx.unwrap());
        SafeTransferLib.safeTransferFrom(yt, msg.sender, address(this),
principal);
```

Here `principal` could be larger than the user specified `params.principal`. In this case, the price of the twoCrypto swap is actually in favor of the user. (So attacker cannot take profit for `swapYtForToken`)

However, in `swapYtForAnyToken`, there is another aggregator swap, attacker can sandwich the aggregator swap, and make user get exactly `params.amountOutMin`. Overall, the user will pay more than `params.principal` but only get `params.amountOutMin`, so attacker can take profit this time.

Reaction: Fixed in https://github.com/napierfi/napier-v2/pull/226

Low Risk

L-1 Attacker can inflate reward index to dos the first depositor

Attacker can directly transfer underlying token to the PT contract right after deployment. Then the attacker frontruns the first depositor's tx and issue 1 wei of PT and YT, at this moment, some reward would already have been generated by the previously transferred token, so reward index will be extremely large. if the index is very close to (but smaller than) `max(uint128)` (10^18 × 10^18 ≈ 2^120 so it's possible), then after some time the index will overflow. As a result, the first depositor can no longer claim yield, unite or combine, they can only wait till maturity and redeem, and all yield will be lost.

Consider using uint256 for reward index.

Reaction: Acknowledged. Usually initial deposit is done atomically as soon as a pool is deployed through FE by `TwoCryptoZap.createAndAddLiquidity` function

L-2 Read-only reentrancy

Consider adding reentrancy guard to preview functions to prevent read-only reentrancy.

Reaction: Fixed in https://github.com/napierfi/napier-v2/pull/237

L-3 Missing `_refund` in some functions

Some functions in Zap use aggregator but miss `_refund`. The unspent dust will be lost.

Reaction: Fixed in https://github.com/napierfi/napier-v2/pull/231

Informational

I-1 Reward proxy can transfer underlying balance

Although reward proxy is considered trusted, consider checking the underlying balance before and after the delegatecall to increase the credibility of the core contract.

Reaction: Fixed in https://github.com/napierfi/napier-v2/pull/227