yAudit Napier V2 Review

Review Resources:

- code repository

Auditors:

- Panda
- Adriro

Table of Contents

{: .no_toc }

Review Summary

Napier V2

Napier V2 is a liquidity hub for future yield trading.

The contracts of the Napier V2 Repo were reviewed over twenty days. Two auditors performed the code review between November 4th and November 29th, 2023. The repository was under active development during the review, but the review was limited to the latest commit, cea0a6b96dd372c44dea5ce5cdb8d10c48f8da02 for the Napier V2 repo.

Scope

The scope of the review consisted of the following contracts at the specific commit:

```
src
├── Constants.sol
├── Errors.sol
├── Events.sol
├── Factory.sol
├── Types.sol
├── interfaces
│   ├── EIP5095.sol
│   ├── IHook.sol
│   ├── IPoolDeployer.sol
│   ├── IRewardProxy.sol
│   └── IWETH.sol
├── modules
│   ├── AccessManager.sol
│   ├── BaseModule.sol
│   ├── FeeModule.sol
│   ├── RewardProxyModule.sol
│   ├── VerifierModule.sol
│   ├── aggregator
```

```
│   │   └── AggregationRouter.sol
│   ├── connectors
│   │   ├── DefaultConnectorFactory.sol
│   │   ├── ERC4626Connector.sol
│   │   ├── VaultConnector.sol
│   │   └── VaultConnectorRegistry.sol
│   ├── deployers
│   │   └── TwoCryptoDeployer.sol
│   └── resolvers
│       ├── CustomConversionResolver.sol
│       ├── ERC4626InfoResolver.sol
│       ├── ExternalPriceResolver.sol
│       ├── SharePriceResolver.sol
│       └── VaultInfoResolver.sol
├── tokens
│   ├── PrincipalToken.sol
│   └── YieldToken.sol
├── types
│   ├── ApproxValue.sol
│   ├── FeePcts.sol
│   ├── Token.sol
│   └── TwoCrypto.sol
├── utils
│   ├── Casting.sol
│   ├── ContractValidation.sol
│   ├── CustomRevert.sol
│   ├── FeePctsLib.sol
│   ├── HookValidation.sol
│   ├── LibApproval.sol
│   ├── LibBlueprint.sol
│   ├── LibTwoCryptoNG.sol
│   ├── ModuleAccessor.sol
│   ├── RewardMathLib.sol
│   ├── TokenNameLib.sol
│   ├── YieldMathLib.sol
│   └── ZapMathLib.sol
└── zap
    ├── TwoCryptoZap.sol
    └── ZapBase.sol
```

After the findings were presented to the Napier V2 team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, Napier V2 and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	The project implements a robust access control system through the AccessManager module.
Mathematics	Good	The mathematical implementations properly handle decimals and fee calculations. Only minor inconsistencies in fee rounding were identified.
Complexity	Low	While the core functionality is well-structured, the codebase uses a lot of assembly code, making reading and understanding the code more complex.
Libraries	Good	The project uses established libraries.
Decentralization	Good	The system demonstrates good decentralization with appropriate permission controls and limited privileged access.
Code stability	Average	The codebase is stable with well-defined interfaces and modules, but the code is still under development.
Documentation	Good	Code is well documented. Several instances of inaccurate or outdated documentation were found.
Monitoring	Average	While the system implements events for major state changes, some functions lack event emissions for important parameter updates.
Testing and verification	Average	The codebase has basic testing coverage but could benefit from more comprehensive testing, particularly around edge cases and integration scenarios.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
    - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.
- Gas savings
    - Findings that can improve the gas efficiency of the contracts.

- Informational
    - Findings including recommendations and best practices.

Critical Findings

None

High Findings

None

Medium Findings

None

Low Findings

1. Low - Missing EIP-5202 blueprint format validation

Technical Details

The `LibBlueprint` library is designed to work with EIP-5202 blueprint contracts but lacks proper validation of blueprint format. While it skips the first 3 bytes when extracting creation code (in extractCreationCode()), it doesn't verify these bytes match the EIP-5202 specification:

- 0xFE (Execution halt byte)
- 0x71 (Blueprint identifier byte)
- 0x00 (Version byte)

Any contract with more than 3 bytes of code could be used as a "blueprint", even if it doesn't follow the EIP-5202 format.

Impact

Low.

Recommendation

As part of `extractCreationCode` add the following code after the size check:

```
// Validate blueprint header bytes
bytes memory code = new bytes(3);
assembly {
    extcodecopy(_blueprint, add(code, 32), 0, 3)
}

if (code[0] != 0xFE || code[1] != 0x71 || code[2] != 0x00) {
    revert InvalidBlueprint();
}
```

Developer Response

Acknowledged.

2. Low - Contract with payable function doesn't use native ETH

The following contract can receive ETH but can not withdraw it: ETH may be sent by users by mistake, and it will be stuck in the contract.

Technical Details

```
File: src/modules/deployers/TwoCryptoDeployer.sol

15: contract TwoCryptoDeployer is IPoolDeployer {
```

TwoCryptoDeployer.sol#L15

Impact

Low.

Recommendation

Remove the payable function modifier.

Developer Response

Acknowledged

3. Low - Missing zero split check in FeeModule.sol

The split fee is checked to differ from zero on update but not on initialization.

Technical Details

The `_splitRatio` parameter is validated to be different than zero in `updateFeeSplitRatio()` but no check is done in `initialize()`, allowing the split to be initially zero.

Impact

Low.

Recommendation

Add the check to `initialize()`.

Developer Response

Fixed f8efff8

4. Low - Unchecked token amounts in `swap()` function can lead to token drainage

Technical Details

Two related issues in the swap function could lead to unintended token transfers:

1. Missing ETH Value Validation

```
File: AggregationRouter.sol

53:         (bool success,) = router.call{value: tokenIn.isNative() ? amountIn :
0}(data.payload);
```

The function doesn't verify that `msg.value` matches `amountIn` when performing native ETH swaps. This allows users to execute swaps without providing the specified ETH amount.

2. Unsafe Return of Excess Tokens

```
File: AggregationRouter.sol

61:         if (returnAmount == 0) revert Errors.AggregationRouter_ZeroReturn();
62:         if (tokenIn.isNative()) {
63:             SafeTransferLib.safeTransferAllETH(msg.sender);
64:         } else {
65:             SafeTransferLib.safeTransferAll(tokenIn.unwrap(), msg.sender);
66:         }
```

After a swap, the contract returns ALL remaining input-type tokens to the caller. This means if the contract holds any excess tokens (either from previous transactions or direct transfers), they can be drained by performing a minimal swap.

1. Contract holds 100 USDC
2. The attacker performs a swap with 0.000001 USDC input
3. After the swap completes, safeTransferAll will return all 100 USDC to the attacker

Impact

Low.

Recommendation

1. Add ETH value validation:
2. Only return the unused portion of the input amount using before and after balances on the `AgregationRouter` contract.

Developer Response

Fixed [dccf68]

5. Low - Potential overflow in blueprint code length

The `initcode` length is unsafely cast to 16 bits while preparing the blueprint deployment code.

Technical Details

The implementation of `blueprint()` unsafely casts the length of the given `initcode` when constructing the deployment code for the blueprint.

```
50:         bytes2 len = bytes2(uint16(blueprint_bytecode.length));
```

Impact

Low. If the length exceeds `type(uint16).max`, the deployment will most likely fail due to the contract size limitation.

Recommendation

Use a safe cast wrapper to convert the length of `initcode` to `uint16`.

Developer Response

Fixed db8fbea

6. Low - Inconsistent fee rounding

Some fees are calculated by rounding down, while other fees are rounded up.

Technical Details

- Performance fee rounds up: YieldMathLib.sol#L75
- Issuance and redemption fee rounds down: PrincipalToken.sol#L778 - PrincipalToken.sol#L783
- Split fee rounds down: PrincipalToken.sol#L618
- Post settlement fee rounds up: PrincipalToken.sol#L655

Impact

Low.

Recommendation

Consider normalizing the fee rounding behavior.

Developer Response

Fixed e299936

7. Low - Missing recover rewards feature for curator and protocol

While users can claim rewards for arbitrary tokens, the same isn't available for tokens assigned to the curator or the protocol.

Technical Details

Users can collect rewards for arbitrary tokens by using the `collectRewards()` which takes an array of addresses as input. As the documentation indicates, this function allows users to claim tokens for rewards not listed in `RewardProxyModule.rewardTokens()`.

However, this feature isn't available for curator or protocol rewards. These rewards also follow the exact mechanism as user rewards, but both `collectCuratorFees()` and `collectProtocolFees()` rely on the list returned by `RewardProxyModule.rewardTokens()`.

Impact

Low. Rewards assigned to the curator or protocol could be locked in the contract.

Recommendation

Add a function similar to `collectRewards()` to allow the curator or protocol to specify the list of tokens.

Developer Response

Fixed 12ae43b

8. Low - Validate call value is empty when the token is not native

When pulling tokens in TwoCryptoZap.sol, the implementation doesn't validate that the call value is zero when the input token is not native.

Technical Details

The implementation of `_pullToken()` validates that `msg.value` matches the given `amount` when the input token is ETH, but doesn't validate that `msg.value` is zero when dealing with ERC20 tokens.

Impact

Low.

Recommendation

When the token is not native, check `msg.value` is zero.

```
    function _pullToken(Token token, uint256 amount) internal {
        if (token.isNative()) {
            if (msg.value != amount) revert Errors.Zap_InconsistentETHReceived();
        } else {
+           if (msg.value != 0) revert Errors.Zap_InconsistentETHReceived();
            SafeTransferLib.safeTransferFrom(token.unwrap(), msg.sender,
address(this), amount);
        }
    }
```

Developer Response

Fixed 06f3019 and 42c9228

9. Low - Missing refund for PT leftover in `swapYtForToken()` and `swapYtForAnyToken()`

Excess of PT tokens resulting from the swap operation are not refunded to the caller.

Technical Details

Aside from the broken assumption that `get_dy(get_dx(maxPrincipal)) <= maxPrincipal`, the implementation of `swapYtForToken()` and `swapYtForAnyToken()` fail to consider any potential excess of PT tokens resulting from the swap operation in the Curve pool.

These functions use the flash-loan style of `combine()`, swapping a portion of the received shares for PT. After the hook is called, the PT contract will burn the `principal` amount of PT and YT tokens. This means that there is an implicit check that the result of the swap operation, let's call it `principalDy`, is at least `principal` to cover the cost of `combine()` `fully`. Any potential difference in `principalDy - principal` is not refunded to the caller.

Impact

Low.

Recommendation

Refund the PT leftover from the swap to the caller.

Additionally, consider moving the `sharesDx` calculation off-chain and taking this parameter along with the `principal` amount as inputs to the function. This will simplify the implementation and help save gas by avoiding the `get_dy()` and `get_dx()` calls.

Developer Response

Fixed fe79808fa45616a9ad9590c0ffcde766934a9661 and 1bda83f

10. Low - Chain reorganization attack on Principal Token deployment

Principal Token deployments are vulnerable to chain reorganization attacks, where an attacker could deploy a malicious PT contract at the same address and exploit subsequent contract interactions.

Technical Details

The Factory.sol contract deploys new Principal Tokens using CREATE2 with a salt composed of the chain id, the expiry timestamp and the address of the resolver.

```
116:         bytes32 salt = EfficientHashLib.hash(block.chainid, expiry,
uint256(uint160(resolver)));
117:         pt = LibBlueprint.computeCreate2Address(salt, suite.ptBlueprint,
"");
```

The resolver contract is deployed using CREATE, which depends on the next nonce available in the factory contract account.

```
112:         address resolver = LibBlueprint.create(suite.resolverBlueprint,
suite.resolverArgs);
```

```
78:     function _create(address _blueprint, bytes memory args) private returns
(address deployed) {
79:         bytes memory initcode = extractCreationCode(_blueprint);
80:         // Combine initcode with constructor arguments
81:         bytes memory deployCode = bytes.concat(initcode, args);
82:
83:         // Deploy the contract
84:         assembly {
85:             deployed := create(0, add(deployCode, 32), mload(deployCode))
86:         }
```

```
87:
88:         if (deployed == address(0)) revert DeploymentFailed();
89:         return deployed;
90:     }
91:
```

This means that an attacker can replace or front-run a deployment and end up with the same contract address as long as they keep the same expiry date. The vulnerability is particularly dangerous because the attacker can manipulate the resolver creation arguments and control the `scale()` function, as these parameters do not influence the resolver contract's address.

Impact

Low. While the potential impact could be critical, executing this attack would be extremely difficult. The natural margin between deployment and subsequent interactions makes this attack impractical in real-world scenarios.

Recommendation

This vulnerability can be mitigated by including the creation arguments in the contract address determination or incorporating the caller's address as part of the salt. Either solution ensures that any modification to these parameters results in a different contract address.

Developer Response

Fixed 5b15503

11. Low - Potential reentrancy in module initialization

The `initializer` modifier from the base module implementation allows reentrancy during the initialization process.

Technical Details

The implementation of the `initializer` modifier in the BaseModule.sol contract sets the initialized variable after giving control to the decorated logic, allowing for potential reentrancy.

```
41:     modifier initializer() {
42:         bool initialized;
43:         assembly {
44:             initialized := sload(_INITIALIZABLE_SLOT)
45:         }
46:         if (initialized) revert Errors.Module_AlreadyInitialized();
47:         _;
48:         assembly {
49:             sstore(_INITIALIZABLE_SLOT, 0x1) // Set `s_initialized` to true
50:         }
51:     }
```

Impact

Low.

Recommendation

Update the storage variable before executing the initialization logic.

```
    modifier initializer() {
        bool initialized;
        assembly {
            initialized := sload(_INITIALIZABLE_SLOT)
        }
        if (initialized) revert Errors.Module_AlreadyInitialized();
-       _;
        assembly {
            sstore(_INITIALIZABLE_SLOT, 0x1) // Set `s_initialized` to true
        }
+       _;
    }
```

Developer Response

Fixed [f9de9ac]

12. Low - Non-standard redemption path may allow bypassing redemption fees

Once matured, the PT can be redeemed using `unite()` or `combine()`, which do not charge redemption fees.

Technical Details

Both `unite()` and `combine()` can be used after the expiry date. As opposed to `redeem()` or `withdraw()`, these functions do not charge a redemption fee. If PT holders can get free or cheap YT, combining rather than redeeming might be more cost-effective, bypassing the protocol and curator fees.

Impact

Low. Although the dynamics of the PT and YT tokens are difficult to predict, combining PT and YT tokens to redeem the underlying might be more profitable than using the standard redemption path, which charges a fee.

Recommendation

Disallow `unite()` and `combine()` once the PT has expired, or include a redemption fee to mimic the behavior of `redeem()` and `withdraw()`.

Developer Response

Fixed 74f89dd.

Gas Saving Findings

1. Gas - Optimize FeeModule parameters storage using bit packing

The `ConstantFeeModule` currently stores fee parameters using a struct-based approach and relies on the `FeePctsLib` for packing/unpacking operations. This implementation can be optimized using bit manipulation to store all fee parameters in a single storage slot.

Technical Details

```
File: FeeModule.sol
094:         s_feePcts = FeePctsLib.pack(
095:             _splitRatio.toUint16(),
096:             FeePctsLib.getIssuanceFeePctBps(s_feePcts),
097:             FeePctsLib.getPerformanceFeePctBps(s_feePcts),
098:             FeePctsLib.getRedemptionFeePctBps(s_feePcts),
099:             FeePctsLib.getPostSettlementFeePctBps(s_feePcts)
100:         );
```

FeeModule.sol#L94-L100

Five different values must be unpacked and repacked to save the new value. It's possible to update only the most left part of the variable using masks.

Impact

Gas savings.

Recommendation

```
// Update constants to define bit positions
+ uint256 private constant FEE_MASK = 0xFFFF; // 16 bits mask
+ uint256 private constant SPLIT_RATIO_OFFSET = 0;
// Update the value stored using masks
+ s_feePcts = (s_feePcts & ~(FEE_MASK << SPLIT_RATIO_OFFSET)) | (_splitRatio &
FEE_MASK);
```

Developer Response

Fixed 8489527

2. Gas - Unnecessary variable loading in `_collectFeeRewards`

Technical Details

```
File: PrincipalToken.sol
967:             (uint256 curatorReward, uint256 protocolReward) =
968:                 (s_rewardRecords[rewardTokens[i]].curatorReward,
s_rewardRecords[rewardTokens[i]].protocolReward); // 1 SLOAD
969:
970:             rewards[i].token = rewardTokens[i];
971:             if (isCurator) {
972:                 rewards[i].amount = curatorReward;
973:                 s_rewardRecords[rewardTokens[i]].curatorReward = 0;
974:             } else {
975:                 rewards[i].amount = protocolReward;
976:                 s_rewardRecords[rewardTokens[i]].protocolReward = 0;
977:             }
```

The code loads **`curatorReward`** and **`protocolReward`** when only one of the two will be used; it also creates an intermediary memory variable used only once. The code can be changed to set rewards[i].amount using the storage variable.

PrincipalToken.sol#L928-L951

Impact

Gas savings.

Recommendation

Remove line 968 and change the code:

```
971:             if (isCurator) {
-972:                 rewards[i].amount = curatorReward;
+972:                 rewards[i].amount =
s_rewardRecords[rewardTokens[i]].curatorReward;
973:                 s_rewardRecords[rewardTokens[i]].curatorReward = 0;
974:             } else {
-975:                 rewards[i].amount = protocolReward;
+975:                 rewards[i].amount =
s_rewardRecords[rewardTokens[i]].protocolReward;
976:                 s_rewardRecords[rewardTokens[i]].protocolReward = 0;
977:             }
```

Developer Response

Fixed 4582304

3. Gas - Use minDy value when possible to reduce gas usage in case of failure

The gas usage will be reduced for transactions.

Technical Details

In this example, the `minDy` should be set to `params.minPrincipal`. The curve pool executing the swap will check the amount out before performing the transfers, which will reduce the gas usage in case of a revert.

```
File: TwoCryptoZap.sol
428:         ptOut =
429:             twoCrypto.exchange_received({i: TARGET_INDEX, j: PT_INDEX, dx:
shares, minDy: 0, receiver: address(this)});
430:
```

```
431:         if (ptOut < params.minPrincipal) revert
Errors.Zap_InsufficientPrincipalTokenOutput();
432:
```

TwoCryptoZap.sol#L428-L431

TwoCryptoZap.sol#L477-L480

Impact

Gas savings.

Recommendation

Use `params.minPrincipal` instead of zero as a minDy parameter and remove the check.

Developer Response

Acknowledged

4. Gas - TwoCrypto pool can send the output tokens directly to the receiver

In some functions of TwoCryptoZap.sol, the implementation sends the exchange output to the zapper contract, only to forward it to the end receiver.

Technical Details

`swapTokenForPt()`

```
427:         // Swap shares for PT using TwoCrypto
428:         ptOut =
429:             twoCrypto.exchange_received({i: TARGET_INDEX, j: PT_INDEX, dx:
shares, minDy: 0, receiver: address(this)});
430:
431:         if (ptOut < params.minPrincipal) revert
Errors.Zap_InsufficientPrincipalTokenOutput();
432:
433:         // Transfer PT to the receiver
434:         SafeTransferLib.safeTransfer(address(principalToken),
params.receiver, ptOut);
```

`swapAnyTokenForPt()`

```
477:         // Step 3: Swap shares for PT using TwoCrypto
478:         ptOut = twoCrypto.exchange_received({i: TARGET_INDEX, j: PT_INDEX,
dx: shares, minDy: 0});
479:
480:         if (ptOut < params.minPrincipal) revert
Errors.Zap_InsufficientPrincipalTokenOutput();
481:
482:         // Transfer PT to the receiver
483:         SafeTransferLib.safeTransfer(address(principalToken),
params.receiver, ptOut);
```

Impact

Gas savings.

Recommendation

Set the receiver of the exchange as the end receiver to save one token transfer.

Developer Response

Fixed b568f66.

5. Gas - Inefficient token transfers and yield accrual in TwoCryptoZap combine operations

In the TwoCryptoZap.combine() function, there are two inefficient aspects:

1. Two unnecessary ERC20 token transfers (PT and YT) from the user to the zap contract
2. Double accrual of yield due to these transfers

The function transfers PT and YT tokens to the TwoCryptoZap contract before combining them. This requires two separate ERC20 transfers and triggers unnecessary yield accruals - first during the YT transfer via `onYtTransfer()` and then again when calling `PrincipalToken.combine()`.
This implementation could be more gas efficient by allowing the PrincipalToken contract to directly spend and burn the tokens from the user's address, similar to how `redeem()` works.

Technical Details

1. In `TwoCryptoZap.combine()`: The YT transfer triggers two transfers that trigger `onYtTransfer()` which accrues yield.
2. In `PrincipalToken.combine()`: It is called which accrues yield again.

Impact

Gas savings.

Recommendation

Modify the `PrincipalToken.combine()` function to accept an optional `owner` parameter and add logic to spend allowance to burn PT and YT directly, similar to how `redeem()` works. This would allow `TwoCryptoZap` to combine tokens in a single operation without requiring separate transfers.

Developer Response

Acknowledged.

6. Gas - Inefficient check in AggregationRouter.sol

The zero amount check during swap can be converted into a slippage check.

Technical Details

The `swap()` function has a check to ensure the output amount is at least greater than zero. This check can be easily turned into a slippage check by adding a minimum amount argument to the function that could serve both purposes.

For example, this can be used from TwoCryptoZap.sol to execute the swap and check the slippage in the same call.

Impact

Gas savings.

Recommendation

Add a slippage check in favor of the zero-amount check.

Developer Response

Acknowledged

7. Gas - Public function in a library

The `computeCreate2Address()` function present in LibBlueprint.sol is declared as `public` which will require a contract deployment and a delegate call to access it.

Technical Details

```
130:     function computeCreate2Address(bytes32 salt, address _blueprint, bytes
memory args)
131:         public
132:         view
133:         returns (address addr)
134:     {
135:         bytes32 bytecodeHash =
keccak256(bytes.concat(extractCreationCode(_blueprint), args));
136:         return computeAddress(salt, bytecodeHash, address(this));
137:     }
```

Impact

Gas savings.

Recommendation

Change the function's visibility to `internal`.

Developer Response

Fixed b8b876d

Informational Findings

1. Informational - Missing event for state change

Parameter or configuration changes should trigger an event to allow being tracked off-chain.

Technical Details

- VerifierModule.sol#L59
- VerifierModule.sol#L63
- AggregationRouter.sol#L29

- AggregationRouter.sol#L33
- VaultConnectorRegistry.sol#L25
- PrincipalToken.sol#L987
- FeeModule.sol#L87

Impact

Informational.

Recommendation

Add events to the functions listed above.

Developer Response

Acknowledged

2. Informational - Consider splitting pause functionality into pause/unpause

Splitting the `setPauseState()` function into two variants would allow better granularity to determine the roles authorized to perform such actions.

Technical Details

PrincipalToken.sol#L993

```
993:     function setPauseState(bool pause) external restricted {
994:         if (pause) {
995:             _pause();
996:         } else {
997:             _unpause();
998:         }
999:     }
```

Impact

Informational.

Recommendation

Split `setPauseState()` into two different functions, like `pause()` and `unpause()`.

Developer Response

Acknowledged.

3. Informational - Unused import

Technical Details

The identifier(s) is imported but never used within the file

```
File: FeeModule.sol

7: import {Factory} from "../Factory.sol";

9: import {AccessManaged, AccessManager} from "./AccessManager.sol";
```

FeeModule.sol#L7, FeeModule.sol#L9

```
File: RewardProxyModule.sol

4: import {LibClone} from "solady/src/utils/LibClone.sol";
```

RewardProxyModule.sol#L4

```
File: ERC4626Connector.sol

8: import {IWETH} from "../../interfaces/IWETH.sol";
```

ERC4626Connector.sol#L8

```
File: RC4626InfoResolver.sol

4: import {LibClone} from "solady/src/utils/LibClone.sol";
```

ERC4626InfoResolver.sol#L4

```
File: PrincipalToken.sol
```

```
29: import {Snapshot, Yield, YieldIndex, YieldMathLib} from
"../utils/YieldMathLib.sol";
```

PrincipalToken.sol#L29

```
File: TwoCryptoZap.sol

10: import {VaultInfoResolver} from "../modules/resolvers/VaultInfoResolver.sol";
```

TwoCryptoZap.sol#L10

Impact

Informational

Recommendation

Remove unused imports to improve code readability.

Developer Response

Fixed 1e06c56

4. Informational - `TODO` left in the code

Technical Details

```
File: AggregationRouter.sol

56: // TODO: Refactor
```

AggregationRouter.sol#L56

Impact

Informational

Recommendation

Make sure it's addressed before deployment.

Developer Response

Fixed dccf682

5. Informational - Remove uncheck loop increments

Summary

Starting from Solidity 0.8.22, it is no longer necessary to use `unchecked` blocks for loop increments to save gas. The compiler now handles this optimization by default.

Technical Details

The codebase contains several instances where `unchecked` blocks are used for loop increments. With the release of Solidity 0.8.22, these `unchecked` blocks are redundant and can be safely removed. By default, the compiler now performs gas-saving optimizations for loop increments.

Example:

```
unchecked {
    ++i;
}
```

Affected Code Locations

Below are the locations in the code where `unchecked` loop increments are used:

- Factory.sol:154
- Factory.sol:183
- AccessManager.sol:43
- AccessManager.sol:57
- PrincipalToken.sol:411
- PrincipalToken.sol:431
- PrincipalToken.sol:668
- PrincipalToken.sol:981
- TwoCryptoZap.sol:994
- TwoCryptoZap.sol:1016

Impact

Informational

Recommendation

Replace the `unchecked` loop with standard increments within the `for` loop header or block.

Developer Response

Acknowledged

6. Informational - Principal and Yield tokens have the same name

Given an underlying vault and an expiry date, both tokens will be deployed with the same name.

Technical Details

The `principalTokenName()` and `yieldTokenName()` functions have the same implementation:

```
14:     function principalTokenName(address target, uint256 expiry) internal view
returns (string memory) {
15:         string memory underlyingName = MetadataReaderLib.readName(target);
16:         return string.concat(PY_NAME_PREFIX, underlyingName, "@",
expiryToDate(expiry));
17:     }
```

```
24:     function yieldTokenName(address target, uint256 expiry) internal view
returns (string memory) {
25:         string memory underlyingName = MetadataReaderLib.readName(target);
26:         return string.concat(PY_NAME_PREFIX, underlyingName, "@",
expiryToDate(expiry));
27:     }
```

Impact

Informational.

Recommendation

Consider changing one of the names to distinguish the principal from the yield token better.

Developer Response

Fixed 40eb73a

7. Informational - Consider using a static call to query view functions

The implementation of DefaultConnectorFactory.sol uses a normal `call` to query the asset of an ERC4626 vault, which is expected to be a non-mutable function.

Technical Details

DefaultConnectorFactory.sol#L20

```
20:         (bool success, bytes memory result) =
target.call(abi.encodeWithSelector(ERC4626.asset.selector));
```

Impact

Informational.

Recommendation

```
-   (bool success, bytes memory result) =
target.call(abi.encodeWithSelector(ERC4626.asset.selector));
+   (bool success, bytes memory result) =
target.staticcall(abi.encodeWithSelector(ERC4626.asset.selector));
```

Developer Response

Fixed bdc5da2

8. Informational - Permits can be front-run in order to grief collections

The collector approval permission can be front-run to cause the collection transaction to fail due to an expired nonce.

Technical Details

In `collectWithPermit()` and `collectRewardsWithPermit()` the caller bundles a signature to grant collection permissions to the zapper contract.

A malicious actor can use any of those signatures directly in the PrincipalToken.sol contract, causing the front-running transaction to fail since the permission will now be invalid.

Impact

Informational.

Recommendation

Consider checking if the zapper is already an approved collector for the caller before executing the permit.

Developer Response

Acknowledged.

9. Informational - Inaccurate documentation

Several comments in the contracts are not aligned with how the system works.

Technical Details

In PrincipalToken.sol, multiple incorrect comments are in the contract's NatSpec.

> *"FeeModule instance that manages the fee logic for the PrincipalToken instance. It is upgradable."*

The FeeModule is the only type of module which cannot be updated.

> *"The curator CAN grant and revoke roles for a PrincipalToken instance and its FeeModule instance."*

The FeeModule is managed by the factory AccessManager, not by the curator.

> *"The curator CAN change the fee module for a PrincipalToken instance."*

Same as before, the FeeModule cannot be updated.

> *"Napier CAN change the reward proxy for a PrincipalToken instance."*

The protocol cannot update the modules for a PT instance.

The `ModuleAccessor.get()` function says that "*The module index is zero-based and id out of range does not revert*", but the implementation reverts when the index is out of range.

In ExternalPriceResolver.sol, the associated comments in the `scale()` function mention the following:

> *"The price from the external feed is assumed to be in 18 decimal precision."*

This is incorrect. The price from the external feed should be expressed in the domain of the asset's number of decimals, which matches the PT/YT decimals. As the function returns `price * offset` (similarly to the other resolvers), the resulting conversion to principal would be `shares * price * offset / 1e18`, meaning that price should reflect the principal's decimals.

Impact

Informational.

Recommendation

Correct the wording in the mentioned comments.

Developer Response

Fixed 861d845

10. Informational - Memory-safe annotation preferred over comment variant

Technical Details

The memory-safe annotation (`assembly ("memory-safe") { ... }`), available starting in Solidity version 0.8.13 is preferred over the comment variant, which will be removed in a future breaking release. The comment variant is only meant for externalized library code that needs to work in earlier versions (e.g. `SafeTransferLib` needs to be able to be used in many different versions).

Technical Details

```
File: PrincipalToken.sol

575: /// @solidity memory-safe-assembly
```

PrincipalToken.sol#L575

```
File: LibApproval.sol

20: /// @solidity memory-safe-assembly

31: /// @solidity memory-safe-assembly
```

LibApproval.sol#L20, LibApproval.sol#L31

```
File: LibBlueprint.sol

144: /// @solidity memory-safe-assembly
```

LibBlueprint.sol#L144

Impact

Informational.

Recommendation

Use `assembly ("memory-safe") { ... }`

Developer Response

Acknowledged

11. Informational - `maxRedeem()` and `maxWithdraw()` fail to account for token maturity

The `maxRedeem()` and `maxWithdraw()` functions return non-zero amounts before expiry.

Technical Details

```
892:     function maxRedeem(address owner) external view returns (uint256) {
893:         return balanceOf(owner);
894:     }
895:
896:     function maxWithdraw(address owner) external view returns (uint256) {
897:         return convertToUnderlying(balanceOf(owner));
898:     }
```

Impact

Informational.

Recommendation

Consider returning zero if the Principal Token has not expired.

Developer Response

Fixed 13f1724.

12. Informational - Non-standard `Redeem` event

The `Redeem` event in the implementation of PrincipalToken.sol is different from the one proposed in EIP-5095.

Technical Details

The `Redeem` event proposed in the EIP-5095 standard is composed of the from and to accounts and the amount of redeemed underlying.

However, the `Redeem` event defined in Events.sol has three addresses (caller, from, to) and both amounts (principal and underlying).

```
20:     /// @dev
`keccak256(bytes("Redeem(address,address,address,uint256,uint256)"))`.
21:     uint256 constant _REDEEM_EVENT_SIGNATURE =
0xaee47cdf925cf525fdae94f9777ee5a06cac37e1c41220d0a8a89ed154f62d1c;
```

Impact

Informational.

Recommendation

Consider changing the event to conform with the standard.

Developer Response

Acknowledged. There is a comment about why we don't follow EIP5095.

13. Informational - Document router assumptions in the Zapper contract

The swap executed by the AggregationRouter.sol contract should check for the deadline. Additionally, it is assumed that the swap entirely consumes input when the operation is the first action executed by the Zapper logic.

Technical Details

All of the functions in the TwoCryptoZap.sol contract that involves an operation in the Curve pool have an explicit deadline check, while the others don't. As some of these functions that don't have an explicit deadline check also execute swaps using the AggregationRouter.sol contract, it is presumed that the underlying provider will include this check.

Similarly, some functions that use the router include a call to `_refund()`, while others that have the swap operation as the first action do not. Since token input from function arguments is directly forwarded to the router, it is assumed that the underlying swap will entirely consume the input, leaving no necessity for a refund.

Impact

Informational.

Recommendation

Consider documenting these two assumptions.

Developer Response

Fixed 6b5a408

Final remarks

The Napier V2 codebase demonstrates solid engineering practices with well-structured code and comprehensive documentation. The review identified no critical or high-severity issues, indicating a robust core design. The few medium and low-severity findings mainly

relate to edge cases and optimization opportunities rather than fundamental security concerns.