ABDK CONSULTING

SMART CONTRACT AUDIT

Notional V2

Fixes

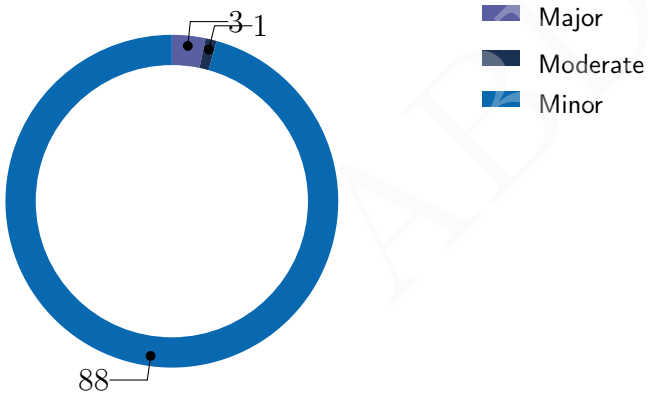
Solidity

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
5th November 2021

We've been asked to review the 19 files in a Github repo. We found 3 major, and a few less important issues.



Findings

ID	Severity	Category	Status
CVF-1	Minor	Procedural	Opened
CVF-2	Minor	Unclear behavior	Info
CVF-3	Minor	Overflow/Underflow	Info
CVF-4	Minor	Suboptimal	Info
CVF-5	Minor	Suboptimal	Opened
CVF-6	Minor	Overflow/Underflow	Opened
CVF-7	Minor	Suboptimal	Opened
CVF-8	Minor	Procedural	Opened
CVF-9	Minor	Suboptimal	Opened
CVF-10	Minor	Suboptimal	Opened
CVF-11	Minor	Overflow/Underflow	Opened
CVF-12	Minor	Procedural	Opened
CVF-13	Minor	Suboptimal	Opened
CVF-14	Minor	Documentation	Opened
CVF-15	Minor	Suboptimal	Fixed
CVF-16	Minor	Suboptimal	Fixed
CVF-17	Minor	Suboptimal	Opened
CVF-18	Minor	Documentation	Opened
CVF-19	Minor	Bad datatype	Opened
CVF-20	Minor	Bad datatype	Opened
CVF-21	Minor	Procedural	Opened
CVF-22	Minor	Suboptimal	Opened
CVF-23	Minor	Suboptimal	Opened
CVF-24	Minor	Suboptimal	Opened
CVF-25	Minor	Suboptimal	Opened
CVF-26	Minor	Documentation	Opened
CVF-27	Minor	Procedural	Opened

ID	Severity	Category	Status
CVF-28	Minor	Suboptimal	Opened
CVF-29	Minor	Suboptimal	Opened
CVF-30	Minor	Readability	Opened
CVF-31	Minor	Suboptimal	Opened
CVF-32	Minor	Procedural	Opened
CVF-33	Minor	Procedural	Opened
CVF-34	Minor	Overflow/Underflow	Opened
CVF-35	Moderate	Flaw	Fixed
CVF-36	Minor	Suboptimal	Opened
CVF-37	Minor	Suboptimal	Fixed
CVF-38	Minor	Suboptimal	Fixed
CVF-39	Minor	Suboptimal	Opened
CVF-40	Minor	Documentation	Opened
CVF-41	Minor	Suboptimal	Opened
CVF-42	Minor	Documentation	Opened
CVF-43	Minor	Documentation	Opened
CVF-44	Minor	Suboptimal	Opened
CVF-45	Minor	Suboptimal	Opened
CVF-46	Minor	Procedural	Opened
CVF-47	Major	Unclear behavior	Info
CVF-48	Minor	Suboptimal	Opened
CVF-49	Minor	Suboptimal	Opened
CVF-50	Minor	Suboptimal	Opened
CVF-51	Minor	Suboptimal	Opened
CVF-52	Minor	Overflow/Underflow	Opened
CVF-53	Minor	Unclear behavior	Opened
CVF-54	Minor	Suboptimal	Opened
CVF-55	Minor	Suboptimal	Opened
CVF-56	Minor	Suboptimal	Opened
CVF-57	Minor	Readability	Opened

ID	Severity	Category	Status
CVF-58	Minor	Bad datatype	Opened
CVF-59	Minor	Bad datatype	Opened
CVF-60	Minor	Suboptimal	Opened
CVF-61	Minor	Suboptimal	Opened
CVF-62	Minor	Documentation	Opened
CVF-63	Major	Flaw	Fixed
CVF-64	Minor	Flaw	Opened
CVF-65	Minor	Suboptimal	Opened
CVF-66	Minor	Procedural	Opened
CVF-67	Minor	Suboptimal	Opened
CVF-68	Minor	Suboptimal	Opened
CVF-69	Minor	Suboptimal	Opened
CVF-70	Minor	Suboptimal	Opened
CVF-71	Minor	Suboptimal	Opened
CVF-72	Minor	Suboptimal	Opened
CVF-73	Minor	Readability	Opened
CVF-74	Minor	Suboptimal	Opened
CVF-75	Minor	Suboptimal	Opened
CVF-76	Minor	Suboptimal	Opened
CVF-77	Minor	Suboptimal	Opened
CVF-78	Minor	Procedural	Opened
CVF-79	Minor	Unclear behavior	Opened
CVF-80	Minor	Suboptimal	Opened
CVF-81	Minor	Suboptimal	Info
CVF-82	Minor	Documentation	Opened
CVF-83	Minor	Suboptimal	Opened
CVF-84	Minor	Flaw	Opened
CVF-85	Minor	Suboptimal	Opened
CVF-86	Minor	Documentation	Opened
CVF-87	Minor	Unclear behavior	Info

ID	Severity	Category	Status
CVF-88	Minor	Bad naming	Opened
CVF-89	Major	Flaw	Info
CVF-90	Minor	Suboptimal	Opened
CVF-91	Minor	Suboptimal	Opened
CVF-92	Minor	Suboptimal	Opened

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	November 1, 2021	D. Khovratovich	Initial Draft
0.2	November 3, 2021	D. Khovratovich	Minor revision
1.0	November 3, 2021	D. Khovratovich	Release
1.1	November 5, 2021	D. Khovratovich	Add client comments
2.0	November 5, 2021	D. Khovratovich	Release

Contact

D. Khovratovich

khovratovich@gmail.com

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations. We have reviewed the next files:

- external/actions/BatchAction.sol
- external/actions/ERC1155Action.sol
- external/actions/GovernanceAction.sol
- external/actions/InitializeMarketsAction.sol
- external/actions/LiquidateCurrencyAction.sol
- external/actions/LiquidatefCashAction.sol
- external/actions/nTokenAction.sol
- external/actions/nTokenRedeemAction.sol
- internal/balances/BalanceHandler.sol
- internal/balances/Incentives.sol
- internal/balances/TokenHandler.sol
- internal/liquidation/LiquidateCurrency.sol
- internal/liquidation/LiquidatefCash.sol
- internal/liquidation/LiquidationHelpers.sol
- internal/valuation/FreeCollateral.sol
- internal/nTokenHandler.sol
- external/PauseRouter.sol
- external/actions/ActionGuards.sol
- contracts/math/FloatingPoint56.sol

2.1 About ABDK

ABDK Consulting, established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function. The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

2.2 Disclaimer

Note that the performed audit represents current best practices and smart contract standards which are relevant at the date of publication. After fixing the indicated issues the smart contracts should be re-audited.

2.3 Methodology

The methodology is not a strict formal procedure, but rather a collection of methods and tactics that combined differently and tuned for every particular project, depending on the project structure and and used technologies, as well as on what the client is expecting from the audit. In current audit we use:

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows code best practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places and that their visibility scopes and access levels are relevant. At this phase we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and is done properly. At this phase we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check that code actually does what it is supposed to do, that algorithms are optimal and correct, and that proper data types are used. We also check that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

3 Detailed Results

3.1 CVF-1

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** nTokenHandler.sol

Recommendation This comment should be removed.

Listing 1:

```
53 // TODO: how many storage reads is this?
```

3.2 CVF-2

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** nTokenHandler.sol

Description These code should be executed only when the original "lastSupplyChangeTime" value wasn't zero.
Client Comment This would improve the efficiency of the code, however, the current code cannot be manipulated because when lastSupplyChangeTime is zero it will be set to blockTime and result in multiplication by zero.

Listing 2:

```
204 require(blockTime >= lastSupplyChangeTime); // dev: invalid
    ↪ block time

209 integralTotalSupply = uint256(int256(integralTotalSupply).add(
210     int256(totalSupply).mul(int256(blockTime -
        ↪ lastSupplyChangeTime))
    ));

213 require(integralTotalSupply >= 0 && integralTotalSupply < type(
    ↪ uint128).max); // dev: integral total supply overflow
```

3.3 CVF-3

- **Severity** Minor
- **Status** Info
- **Category** Overflow/Underflow
- **Source** nTokenHandler.sol

Description Conversion to "int256" may overflow.
Recommendation Consider using safe conversions.
Client Comment Agree that int256 is not ideal here, will change in a future version of the code. However, in this function the values come directly from storage (not user input) and are capped at values that cannot overflow in int256.

Listing 3:

```
209 integralTotalSupply = uint256(int256(integralTotalSupply).add(
210     int256(totalSupply).mul(int256(blockTime -
        ↪ lastSupplyChangeTime))
```

3.4 CVF-4

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** nTokenHandler.sol

Description Using signed type in this formula doesn't make sense, as there could be no negative values.
Recommendation Consider using uint256 instead of int256.
Client Comment Similar comment to above.

Listing 4:

```
209 integralTotalSupply = uint256(int256(integralTotalSupply).add(
210     int256(totalSupply).mul(int256(blockTime -
        ↪ lastSupplyChangeTime))
));
```

3.5 CVF-5

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenHandler.sol

Recommendation These checks shpuld be made before more expensive operations.

Listing 5:

```
213 require(integralTotalSupply >= 0 && integralTotalSupply < type(
      ↪ uint128).max); // dev: integral total supply overflow
    require(blockTime < type(uint32).max); // dev: last supply
      ↪ change supply overflow
```

3.6 CVF-6

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** nTokenHandler.sol

Description Overflow is possible here.
Recommendation Consider using a safe conversion

Listing 6:

```
256     nTokenStorage.lastSupplyChangeTime = uint32(blockTime);

296 context.lastInitializedTime = uint32(lastInitializedTime);
```

3.7 CVF-7

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenHandler.sol

Recommendation Sending an array of structs would be less error-prone and would not require a length check.

Listing 7:

```
366 uint32[] calldata annualizedAnchorRates,

368  uint32[] calldata proportions
```

3.8 CVF-8

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** nTokenHandler.sol

Description The variable "i" is not initialized initialized.
Recommendation Consider explicitly initializing to zero.

Listing 8:

```
378 for (uint256 i; i < proportions.length; i++) {
```

3.9 CVF-9

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidateCurrency.sol

Description These two functions have much in common.
Recommendation Consider refactoring the code to reduce code duplication.

Listing 9:

```
437 unction _withdrawLocalLiquidityTokens(

579 unction _withdrawCollateralLiquidityTokens(
```

3.10 CVF-10

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidateCurrency.sol

Recommendation This line could be simplified using the "-=" operator.

Listing 10:

```
493 ssetAmountRemaining = assetAmountRemaining - w.netCashIncrease.
    ↪ sub(w.incentivePaid);
```

3.11 CVF-11

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** LiquidateCurrency.sol

Description Underflow is possible here

Listing 11:

```
679 int256 marketIndex = asset.assetType - 1;
```

3.12 CVF-12

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** TokenHandler.sol

Description It is not ensured that the "currencyId" value is not "Constants.ETH_CURRENCY_ID".
Recommendation Consider adding such check.

Listing 12:

```
152 tore[currencyId][underlying] = tokenStorage;
```

3.13 CVF-13

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TokenHandler.sol

Recommendation These operations can be merged to a single ‘require‘.

Listing 13:

```
206 uint256 success = CErc20Interface(assetToken.tokenAddress).
    ↪ redeem(assetAmountExternal);
equire(success == Constants.COMPOUND_RETURN_CODE_NO_ERROR, "
    ↪ Redeem");
```

3.14 CVF-14

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** TokenHandler.sol

Description The semantics of the returned value is unclear.
Recommendation Consider giving it a descriptive name and/or describing in the documentation comment.

Listing 14:

```
259  private returns (int256) {
```

3.15 CVF-15

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** TokenHandler.sol

Description This logic looks like an unnecessary complication. Dust allocation issues are usually solved by rounding calculation results toward the protocol, i.e. against the user. The value returned by this function is precise, so not need to care about rounding here.
Recommendation The proper rounding direction should be chosen in some other place, where actual rounding occurs.

Listing 15:

```
285 // If decimals is less than internal token precision, we change
    ↪ how much the the user is credited
// during this deposit so that the protocol accrues the dust (
    ↪ not the user's cash balance)
finalAmountAdjustment = 1;
```

3.16 CVF-16

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** TokenHandler.sol

Description When the value returned by the "_deposit" function is converted from external to internal precision, it is anyway roudned down. No need to reduce it by 1.

Listing 16:

```
319 / protocol will retain more balance than the user. This already
    ↪ happens in the conversion below. When
320 / depositing, we want to decrease the amount of cash balance we
    ↪ credit to the user by a dust amount
/ so that the protocol accrues the dust (rather than the user's
    ↪ balance). This is implemented in _deposit
```

3.17 CVF-17

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TokenHandler.sol

Description Allocating an array is redundant here.
Recommendation Just use memory slot referred by the free memory pointer for as temporary buffer. See warning here for details: https://docs.soliditylang.org/en/v0.7.0/internals/layout_in_memory.html

Listing 17:

```
355 int256[1] memory result;
```

3.18 CVF-18

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** CompoundToNotionalV2.sol

Recommendation Variable names usually start with small letter.

Listing 18:

```
11 otionalProxy public immutable NotionalV2;
```

3.19 CVF-19

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** CompoundToNotionalV2.sol

Recommendation The type of the "token" argument should be "CTokenInterface".

Listing 19:

```
19 unction enableToken(address token, address spender) external {
```

3.20 CVF-20

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** CompoundToNotionalV2.sol

Recommendation The type of the "cTokenBorow" argument should be "CTokenInterface".

Listing 20:

```
25 ddress cTokenBorrow,
```

3.21 CVF-21

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** CompoundToNotionalV2.sol

Description It is not checked that the lengths of these arrays are the same.
Recommendation Consider adding such check. Also, it would be more efficient to pass a single array of structs with two values instead of two parallel arrays. Such approach would make the length check unnecessary.

Listing 21:

```
27 int16[] memory notionalV2CollateralIds,
   int256[] memory notionalV2CollateralAmounts,
```

3.22 CVF-22

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** CompoundToNotionalV2.sol

Recommendation The "sender == address(this) check" is redundant, as the Notional code guarantees this.

Listing 22:

```
63 equire(msg.sender == address(NotionalV2) && sender == address(
   ↪ this), "Unauthorized callback");
```

3.23 CVF-23

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** CompoundToNotionalV2.sol

Description The "success" variable is redundant.
Recommendation Just check the returned value directly without storing it into a variable.

Listing 23:

```
74 ool success = IERC20(underlyingToken).transferFrom(account,
   ↪ address(this), cTokenRepayAmount);

84     success = CTokenInterface(assetToken.tokenAddress).
          ↪ transferFrom(account, address(this),
          ↪ notionalV2CollateralAmounts[i]);
```

3.24 CVF-24

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** CompoundToNotionalV2.sol

Description The "code" variable is redundant.
Recommendation Just check the returned value directly without storing it into a variable.

Listing 24:

```
78 int code = CErc20Interface(cTokenBorrow).repayBorrowBehalf(
     ↪ account, cTokenRepayAmount);
```

3.25 CVF-25

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** CompoundToNotionalV2.sol

Description This function is redundant.
Recommendation Just remove it to prevent the contract from receiving ether.

Listing 25:

```
94 eceive() external payable {
     // This contract cannot migrate ETH loans because there is no
        ↪ way
     // to do transferFrom on ETH
     revert("Cannot transfer ETH");
```

3.26 CVF-26

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** LiquidationHelpers.sol

Recommendation The comment is not accurate anymore.

Listing 26:

```
48 / Collateral currency must be unset or not equal to the local
     ↪ currency
```

3.27 CVF-27

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** LiquidationHelpers.sol

Description A comment in the middle of an expression is weird.
Recommendation Consider putting it before the expression.

Listing 27:

```
136 / netETHValue must be negative to be in liquidation
```

3.28 CVF-28

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidatefCash.sol

Recommendation There is a more efficient way to calcualte max(0, x) in Solidity: x & ~ x » 255.

Listing 28:

```
68 racleRate < buffer ? 0 : oracleRate.sub(buffer)

74 racleRate < buffer ? 0 : oracleRate.sub(buffer)
```

3.29 CVF-29

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidatefCash.sol

Description In case the account has a bitmap currency, but this bitmap currency is different from "currencyId", this condition will be false, and the portfolio will be scanned in the loop below.
Recommendation Consider refactoring the code like this: if (context.accountContext.isBitmapEnabled()) { return context.accountContext.bitmapCurrencyId == currencyId ? bitmapAssetsHandler.getifCashNotional(liquidateAccount, currencyId) : 0; } else { /* Scan the portfolio */ }

Listing 29:

```
89 f (context.accountContext.bitmapCurrencyId == currencyId) {
```

3.30 CVF-30

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** LiquidatefCash.sol

Description The code below looks like it is always executed, while it is actually executed only when 'context.accountContext.bitmapCurrencyId != currencyId'.
Recommendation Consider putting the rest of the function into an explicit "else" branch.

Listing 30:

```
94
```

3.31 CVF-31

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidatefCash.sol

Description The value "c.factors.localETHRate.haircut" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 31:

```
161 div(c.factors.localETHRate.haircut);
```

3.32 CVF-32

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** LiquidatefCash.sol

Recommendation Should be "else if".

Listing 32:

```
180 f (notional == 0) continue;
```

3.33 CVF-33

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** LiquidatefCash.sol

Description A comment in the middle of an expression is weird.
Recommendation Consider putting it before the expression.

Listing 33:

```
197 .fCashNotionalTransfers[i] = c.underlyingBenefitRequired
/ NOTE: Governance should be set such that these discount
    ↪ factors are unlikely to be zero. It's
/ possible that the interest rates are so low that this
    ↪ situation can occur.
200     .divInRatePrecision(liquidationDiscountFactor.sub(
        ↪ riskAdjustedDiscountFactor).abs());
```

3.34 CVF-34

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** LiquidatefCash.sol

Description Phantom overflow is possible here.
Recommendation Consider using the "muldiv" function.

Listing 34:

```
229 mul(c.localCashBalanceUnderlying)
230 div(fCashLiquidationValueUnderlying);
```

3.35 CVF-35

- **Severity** Moderate
- **Status** Fixed
- **Category** Flaw
- **Source** LiquidatefCash.sol

Description Even if the liquidation transaction doesn't incur debt, it may worsen the situation for a liquidator whose free collateral is already negative. Even if the liquidation transaction does incur debt, it may improve the situation for a liquidator whose free collateral was negative. **Recommendation** Consider changing this logic to allow transaction that improve the liquidator's situation and forbid those that worsen it and end up in a negative free collateral state.

Listing 35:

```
543 / If the liquidator takes on debt as a result of the liquidation
      ↪ and has debt in their portfolio
    / then they must have a free collateral check. It's possible for
      ↪ the liquidator to skip this if the
    / negative fCash incurred from the liquidation nets off against
      ↪ an existing fCash position.
```

3.36 CVF-36

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** FloatingPoint56.sol

Description The conversion to the "uint256" type is redundant.

Listing 36:

```
30 uint256 bitShift = uint256(uint8(value));
```

3.37 CVF-37

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Incentives.sol

Description A multiplication performed after a division could lead to precision degradation. **Recommendation** Consider doing the division at the very end of the calculation.

Listing 37:

```
36 uint256 proRataYears =
       timeSinceLastClaim.mul(uint256(Constants.
         ↪ INTERNAL_TOKEN_PRECISION)).div(Constants.YEAR);

39 eturn proRataYears.mul(emissionRatePerYear);
```

3.38 CVF-38

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** Incentives.sol

Recommendation Consider adding "|| nTokenBalance == 0" to the condition.

Listing 38:

```
64 f (lastClaimTime == 0 || lastClaimTime >= blockTime) return 0;
```

3.39 CVF-39

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Incentives.sol

Recommendation This could be calculated in a more precise way: uint256 incentivesToClaim = nTokenBalance.mul(incentiveRate).mul(timeSinceLastClaim).div(integralTotalSupply.sub(lastClaimIntegralSupply));

Listing 39:

```
99  int256 avgTotalSupply = integralTotalSupply.sub(
      ↪ lastClaimIntegralSupply).div(timeSinceLastClaim);

111 int256 incentivesToClaim = nTokenBalance.mul(incentiveRate).div(
      ↪ avgTotalSupply);
```

3.40 CVF-40

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** BalanceHandler.sol

Description This is not true anymore.

Listing 40:

```
36 // @return Returns two values:
```

3.41 CVF-41

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BalanceHandler.sol

Description This assignment should be done only if both, "token.hasTransferFee" and "forceTransfer" flags are false.

Listing 41:

```
51 nt256 assetAmountInternal = token.convertToInternal(
   ↪ assetAmountExternal);
```

3.42 CVF-42

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** BalanceHandler.sol

Description This comment is confusing. One could read it as if "like cTokens" relates to "tokens" ratehr than to "mintable", i.e. like: "Tokens like cTokens that are not mintable will be deposited as assetTokens", which would change the meaning of the comment to the opposite.
Recommendation Consider rephrasing.

Listing 42:

```
124 // Tokens that are not mintable like cTokens will be deposited
    ↪ as assetTokens
```

3.43 CVF-43

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** BalanceHandler.sol

Description This comment is not accurate anymore.
Recommendation Consider fixing it.

Listing 43:

```
261 // @dev Returns the amount transferred in underlying or asset
    ↪ terms depending on how redeem to underlying
// is specified.
```

3.44 CVF-44

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BalanceHandler.sol

Recommendation This check could be simplified as: currencyId - 1 < Constants.MAX_CURRENCIES

Listing 44:

```
553 equire(0 < currencyId && currencyId <= Constants.MAX_CURRENCIES)
    ↪ ; // dev: invalid currency id
```

3.45 CVF-45

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PauseRouter.sol

Description This should be calculated only when owner != msg.sender and msg.sender == pauseGuardian.

Listing 45:

```
31 ool isRollbackCheck = rollbackRouterImplementation != address(0)
    ↪ &&
    newImplementation == rollbackRouterImplementation;
```

3.46 CVF-46

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** PauseRouter.sol

Description This function should emit some event.

Listing 46:

```
48 unction setLiquidationEnabledState(bytes1
    ↪ liquidationEnabledState_) external {
```

3.47 CVF-47

- **Severity** Major
- **Status** Info
- **Category** Unclear behavior
- **Source** PauseRouter.sol

Description Is "pauseGuardian" really able to call the "setLiquidationEnabledState" function? If no, then is should not be considered as an authorized address.
Client Comment Yes, this is intended. The pauseGuardian would be able to enable liquidations to keep the protocol solvent if it were required. Governance would require a multi-day delay which is not desireable.

Listing 47:

```
50 equire(owner == msg.sender || msg.sender == pauseGuardian);
```

3.48 CVF-48

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PauseRouter.sol

Recommendation This function don't need to be public as it is a low-level part of the routing logic.

Listing 48:

```
58 unction getRouterImplementation(bytes4 sig) public view returns
   ↪ (address) {
```

3.49 CVF-49

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PauseRouter.sol

Recommendation These two conditions could be merged into one via logical "or" operation, and the contents of the corresponding "then" branched are the same.

Listing 49:

```
61 f (
     (sig == NotionalProxy.calculateCollateralCurrencyLiquidation.
        ↪ selector ||
       sig == NotionalProxy.liquidateCollateralCurrency.selector
          ↪ ) &&
     isEnabled(Constants.COLLATERAL_CURRENCY_ENABLED)

69 f (
70     (sig == NotionalProxy.calculateLocalCurrencyLiquidation.
        ↪ selector ||
       sig == NotionalProxy.liquidateLocalCurrency.selector) &&
     isEnabled(Constants.LOCAL_CURRENCY_ENABLED)
```

3.50 CVF-50

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PauseRouter.sol

Recommendation These two conditions could be merged into one via logical "or" operation, and the contents of the corresponding "then" branched are the same.

Listing 50:

```
77 f (
     (sig == NotionalProxy.liquidatefCashLocal.selector ||
       sig == NotionalProxy.calculatefCashLocalLiquidation.
          ↪ selector) &&
80   isEnabled(Constants.LOCAL_FCASH_ENABLED)

85 f (
     (sig == NotionalProxy.liquidatefCashCrossCurrency.selector ||
       sig == NotionalProxy.
          ↪ calculatefCashCrossCurrencyLiquidation.selector) &&
     isEnabled(Constants.CROSS_CURRENCY_FCASH_ENABLED)
```

3.51 CVF-51

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** PauseRouter.sol

Recommendation It would be cheaper to revert here in case the function selector was recognized as liquidation-related, but the corresponding bit in the "liquidationEnabledState" value is not set.

Listing 51:

```
93 / If not found then delegate to views. This will revert if there
     ↪  is no method on
   / the view contract
```

3.52 CVF-52

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** nTokenRedeemAction.sol

Description Phantom overflow is possible here.
Recommendation Consider using the "muldiv" function.

Listing 52:

```
196 nt256 assetCashShare = nToken.cashBalance.mul(tokensToRedeem).
      ↪ div(nToken.totalSupply);

259   int256 tokensToRemove = asset.notional.mul(tokensToRedeem).div
        ↪ (totalSupply);
```

3.53 CVF-53

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** nTokenAction.sol

Description This check is redundant. What problem does it prevent?

Listing 53:

```
113 equire(tokenHolder != address(0));
```

3.54 CVF-54

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenAction.sol

Recommendation We can simply do nothing if from==to.

Listing 54:

```
137 equire(from != to, "Cannot transfer to self");

161 equire(from != to, "Cannot transfer to self");
```

3.55 CVF-55

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenAction.sol

Description This allows using either specific allowance or generic (whitelist) allowance, but not both.
Recommendation Consider implementing logic to use as much specific allowance as possible and, if it is not enough, use generic allowance.

Listing 55:

```
167 f (allowance > 0) {

175  else {
```

3.56 CVF-56

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** nTokenAction.sol

Description This function always returns true.
Recommendation Consider removing the returned value.

Listing 56:

```
373  internal returns (bool) {

421    return true;
```

3.57 CVF-57

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** GovernanceAction.sol

Recommendation The conditions could be optimized like this: currencyId - 1 < maxCurrencyId

Listing 57:

```
33 equire(0 < currencyId && currencyId <= maxCurrencyId, "Invalid
   ↪ currency id");
```

3.58 CVF-58

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** GovernanceAction.sol

Recommendation The type of this argument should be "PauseRouter".

Listing 58:

```
60 ddress pauseRouter_,
```

3.59 CVF-59

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** GovernanceAction.sol

Recommendation The typeof this argument should be more specific.

Listing 59:

```
61 ddress pauseGuardian_
```

3.60 CVF-60

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** GovernanceAction.sol

Description This event is emitted even if nothing actually changed.

Listing 60:

```
66 mit PauseRouterAndGuardianUpdated(pauseRouter_, pauseGuardian_);

390 mit UpdateAuthorizedCallbackContract(operator, approved);
```

3.61 CVF-61

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** GovernanceAction.sol

Description This event is emitted even if nothing actually changed.

Listing 61:

```
140 mit UpdateMaxCollateralBalance(currencyId,
    ↪ maxCollateralBalanceInternalPrecision);
```

3.62 CVF-62

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** GovernanceAction.sol

Description The documentation comment for the "updateMaxCollateralBalance" function says that a max collateral balance is only set on asset tokens, but not on underlying tokens. However this line checks a max collateral balance for an underlying token. This is confusing.
Recommendation If there are scenarios when a max collateral balance could be set on an underlying token, consider explaining them in a comment.

Listing 62:

```
167 nderlyingToken.maxCollateralBalance == 0
```

3.63 CVF-63

- **Severity** Major
- **Status** Fixed
- **Category** Flaw
- **Source** GovernanceAction.sol

Description This allows changing the value of "underlyingDecimalPlaces" for a currency, which is weird, as it shouldn't be possible to change the underlying currency for an asset, and the number of decimals for an asset also cannot change over time.
Recommendation Consider replacing this assignment with a check to ensure that the number of underlying decimals didn't change.
Client Comment Added a require statement to ensure that the underlying token does not change.

Listing 63:

```
446 nderlyingDecimalPlaces: underlyingDecimals
```

3.64 CVF-64

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** GovernanceAction.sol

Description The validity of the "currencyId" argumetn is not checked.
Recommendation Consider adding: _checkValidCurrency(currencyId);

Listing 64:

```
463 int16 currencyId,
```

3.65 CVF-65

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidatefCashAction.sol

Recommendation Passing a single array of structs with two fields instead of two parallel arrays would be more efficient and would make the length check unnecessary.

Listing 65:

```
203 int256[] calldata fCashMaturities,
    int256[] calldata maxfCashLiquidateAmounts,

244 int256[] calldata fCashMaturities,
    int256[] calldata maxfCashLiquidateAmounts,
```

3.66 CVF-66

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** InitializeMarketsAction.sol

Recommendation The "currencyId" parameter should be indexed.

Listing 66:

```
45 event MarketsInitialized(uint16 currencyId);
   vent SweepCashIntoMarkets(uint16 currencyId, int256
     ↪ cashIntoMarkets);
```

3.67 CVF-67

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** InitializeMarketsAction.sol

Description This function seems to know too much about the scenarios where it is used. This is a bad practice, as it makes the code more fragile.
Recommendation Consider making the function more generic.

Listing 67:

```
225 / When looping for sweepCashIntoMarkets, previousMarkets is not
    ↪ defined and we only
/ want to apply withholding for idiosyncratic fCash.
```

```
252         // During initialize markets we will have access to the
            ↪ previous markets
            // and their oracle rates.
```

3.68 CVF-68

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** InitializeMarketsAction.sol

Description This logic is coded twice.
Recommendation Consider refactoring the code to remove duplication: if (previousMarkets.length != 0 || !DateTime.isValidMaturity (nToken.cashGroup.maxMarketIndex, maturity, blockTime)) { ... process the bit ... } assetsBitmap = assetsBitmap.setBit(bitNum, false); bitNum = assetsBitmap.getNextBitNum();

Listing 68:

```
236     assetsBitmap = assetsBitmap.setBit(bitNum, false);
        bitNum = assetsBitmap.getNextBitNum();
```

```
311 ssetsBitmap = assetsBitmap.setBit(bitNum, false);
    itNum = assetsBitmap.getNextBitNum();
```

3.69 CVF-69

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** InitializeMarketsAction.sol

Recommendation This code could be simplified as: int128 expValue = ABDKMath.divi ((exchangeRate.sub (rateAnchor)).mulInRatePrecision(rateScalar), Constants.RATE_PRECISION);

Listing 69:

```
437 nt128 expValue = ABDKMath64x64.fromInt(
        // (exchangeRate - rateAnchor) * rateScalar
        (exchangeRate.sub(rateAnchor)).mulInRatePrecision(rateScalar)
440 ;

443 expValue = ABDKMath64x64.div(expValue, Constants.
        ↪ RATE_PRECISION_64x64);
```

3.70 CVF-70

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BatchAction.sol

Description The expression "actions[i - 1].currencyId" was already calculated on the previous loop iteration.
Recommendation Consider reusing it from there like this: uint256 prevCurrencyId = 0; for (...) { ... require (action.currencyId > prevCurrencyId); prevCurrencyId = action.currencyId; ... }

Listing 70:

```
 53 equire(action.currencyId > actions[i - 1].currencyId, "Unsorted
        ↪ actions");

156 equire(action.currencyId > actions[i - 1].currencyId, "Unsorted
        ↪ actions");
```

3.71 CVF-71

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BatchAction.sol

Description The first argument of the call is redundant as its value always equals to the call target.
Recommendation Consider removing this argument.

Listing 71:

```
119 otionalCallback(msg.sender).notionalCallback(msg.sender, account
    ↪ , callbackData);
```

3.72 CVF-72

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BatchAction.sol

Description The expression "accountContext.isBitmapEnabled()" is calculated on every loop iteration and once again after the loop.
Recommendation Consider calculating once before the loop and reusing.

Listing 72:

```
180         if (accountContext.isBitmapEnabled()) {

233 f (!accountContext.isBitmapEnabled()) {
```

3.73 CVF-73

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** BatchAction.sol

Recommendation This line could be simplified as: accountContext.hasDebt |= Constants.HAS_ASSET_DEBT;

Listing 73:

```
197 ccountContext.hasDebt = Constants.HAS_ASSET_DEBT |
    ↪ accountContext.hasDebt;
```

3.74 CVF-74

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BatchAction.sol

Recommendation This check became redundant after using a safe conversion in the previous line.

Listing 74:

```
394 equire(withdrawAmount >= 0); // dev: withdraw action overflow
```

3.75 CVF-75

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** ERC1155Action.sol

Recommendation There is a cheaper way to calculate max (0, x) in Solidity: x & ~x >> 255

Listing 75:

```
43 eturn notional < 0 ? 0 : uint256(notional);
```

3.76 CVF-76

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** ERC1155Action.sol

Recommendation It would be more efficient to pass a single array of structs with two fields, rather than two parallel address. Such approach would also make the length check unnecessary.

Listing 76:

```
71 unction signedBalanceOfBatch(address[] calldata accounts,
   ↪ uint256[] calldata ids)

97 function balanceOfBatch(address[] calldata accounts, uint256[]
   ↪ calldata ids)

308 function decodeToAssets(uint256[] calldata ids, uint256[]
   ↪ calldata amounts)

320 unction _decodeToAssets(uint256[] calldata ids, uint256[]
   ↪ calldata amounts)
```

3.77 CVF-77

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** ERC1155Action.sol

Description The variable "i" is not initialized.
Recommendation Consider explicitly initializing to zero.

Listing 77:

```
 80 or (uint256 i; i < accounts.length; i++) {
148 for (uint256 i; i < portfolio.length; i++) {
330 for (uint256 i; i < ids.length; i++) {
```

3.78 CVF-78

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** ERC1155Action.sol

Description Not all execution branches on this function do return value.
Recommendation Consider explicitly returning zero after the loop.

Listing 78:

```
141 unction _balanceInArray(PortfolioAsset[] memory portfolio,
    ↪ uint256 id)
```

3.79 CVF-79

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** ERC1155Action.sol

Description This check makes normal transfers more expensive. Is it really necessary? What problems does it prevent?
Client Comment Since we may enable global transfer operators we want to guard against strange behavior coming from contracts external to the system with blanke authorization.

Listing 79:

```
296 equire(from != to && to != address(0) && to != address(this), "
    ↪ Invalid address");
```

3.80 CVF-80

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** ERC1155Action.sol

Description The "assets" variable is redundant.
Recommendation Just give a name to the returned value and use it instead.

Listing 80:

```
316 PortfolioAsset[] memory assets, /* */) = _decodeToAssets(ids,
    ↪ amounts);
```

3.81 CVF-81

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** ERC1155Action.sol

Recommendation This function should either explicitly require that from != to, or implement a special handling logic for the from == to case, as the current implementation cannot handle this case properly.
Client Comment These checks are made in the two external functions that call _transfer.

Listing 81:

```
382 unction _transfer(
```

3.82 CVF-82

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** ERC1155Action.sol

Description The semantics of the returned values is unclear.
Recommendation Consider explaining in the documentation comment.

Listing 82:

```
386 internal returns (AccountContext memory, AccountContext memory)
    ↪ {
```

3.83 CVF-83

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** ERC1155Action.sol

Description A silent revert is when _returnData.length == 0. In case 0 < _returnData.length < 68 it is not a silent revert, but revert returned something that we cannot parse.
Recommendation Consider wrapping the "_returnData" value into the returned string in such a case.

Listing 83:

```
489 f (_returnData.length < 68) return "Transaction reverted
    ↪ silently";
```

3.84 CVF-84

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** ERC1155Action.sol

Description This screws up the "_returnData.length".
Recommendation Look this answer for cleaner solution: https://ethereum.stackexchange.com/questions/83528/how-can-i-get-the-revert-reason-of-a-call-in-solidity-so-that-i-can-use-it-in-th/110428#110428

Listing 84:

```
493 returnData := add(_returnData, 0x04)
```

3.85 CVF-85

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** ActionGuards.sol

Description This condition is satisfied not only when the "reentrancyStatus" value is _NOT_ENTERED, but also when this value is zero, i.e. the contract is not initialized.
Recommendation Consider forbidding using the contract before initialization by changing this condition to: 'reentrancyStatus == _NOT_ENTERED'

Listing 85:

```
23 equire(reentrancyStatus != _ENTERED, "Reentrant call");
```

3.86 CVF-86

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** ActionGuards.sol

Description The comment is confusing. It is unclear what particular accounts the word "these" refers to.
Recommendation Consider rephrasing.

Listing 86:

```
35 / These accounts cannot receive deposits, transfers, fCash or
     ↪ any other
   / types of value transfers.
   unction requireValidAccount(address account) internal view {
```

3.87 CVF-87

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** ActionGuards.sol

Description This function performs a quite expensive check on every nToken transfer and on some other operations. Is it really necessary? What potential problems these checks prevent, that worth spending extra gas?
Client Comment It is, however, we are worried about unintended consequences if the nToken somehow receives fCash or executes trades outside of its purview. Agree this check is expensive and if we can remove it somehow that would be ideal but for now I think we feel safer with this check.

Listing 87:

```
37 unction requireValidAccount(address account) internal view {
```

3.88 CVF-88

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** AccountContextHandler.sol

Description The function name is misleading. This function may not only enable bitmap for an account, but also disable it, and change bitmap currency.
Recommendation Consider renaming.

Listing 88:

```
63 unction enableBitmapForAccount(
```

3.89 CVF-89

- **Severity** Major
- **Status** Info
- **Category** Flaw
- **Source** AccountContextHandler.sol

Recommendation Should be: } else if (currencyId != 0) {.
Client Comment Concern in this line is that setting the currencyId == 0 while bitmaps are not enabled would result in potentially destricutive behavior. Agree that this else if statement would be more clear in this case. If a user does attempt to execute a transaction like this, however, the contract will revert on this line: https://github.com/notional-finance/contracts-v2/blob/abdk-audit-fixes/contracts/internal/AccountContextHandler.sol#L121 since a currencyId of zero is invalid. This has been verified in a unit test.

Listing 89:

```
78  else {
```

3.90 CVF-90

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AccountContextHandler.sol

Recommendation This could be optimized as: require (currencyId - 1 < Constants.MAX_CURRENCIES);

Listing 90:

```
129 equire(currencyId != 0 && currencyId <= Constants.MAX_CURRENCIES
    ↪ ); // dev: invalid currency id

167 equire(0 < currencyId && currencyId <= Constants.MAX_CURRENCIES)
    ↪ ; // dev: invalid currency id
```

3.91 CVF-91

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AccountContextHandler.sol

Recommendation This check should be done earlier to save gas.

Listing 91:

```
289 equire(mustSettleAssets(accountContext) == false); // dev:
    ↪ cannot store matured assets
```

3.92 CVF-92

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AccountContextHandler.sol

Recommendation This assignment should be inside the "if" statement above.

Listing 92:

```
327 astCurrency = currencyId;
```