Veridise Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Aqua

Veridise Inc.

March 7, 2024

► **Prepared For:**

Native-Org
https://native.org

► **Prepared By:**

Ajinkya Rajput
Andreea Bușterchi

► **Contact Us:** contact@veridise.com

► **Version History:**

Mar. 7, 2024 V1 Feb 28, 2024 Initial Draft

Contents

Contents iii

1 Executive Summary 1

2 Project Dashboard 3

3 Audit Goals and Scope 5
- 3.1 Audit Goals . . . 5
- 3.2 Audit Methodology & Scope . . . 5
- 3.3 Classification of Vulnerabilities . . . 6

4 Vulnerability Report 7
- 4.1 Detailed Description of Issues . . . 8
  - 4.1.1 V-AQU-VUL-001: Unsafe type-cast . . . 8
  - 4.1.2 V-AQU-VUL-002: getAccountSnapshot() returns stale values . . . 11
  - 4.1.3 V-AQU-VUL-003: Possible loss of admin access to the contract . . . 13
  - 4.1.4 V-AQU-VUL-004: Duplicate variable declaration . . . 14
  - 4.1.5 V-AQU-VUL-005: Centralisation Risk . . . 15
  - 4.1.6 V-AQU-VUL-006: Missing non-zero checks . . . 16
  - 4.1.7 V-AQU-VUL-007: Possible spurious reverts due to missing non-zero check 18
  - 4.1.8 V-AQU-VUL-008: Unused imports / errors . . . 19
  - 4.1.9 V-AQU-VUL-009: Missing nonce in widget signatures . . . 20
  - 4.1.10 V-AQU-VUL-010: Unnecessary visibility modifier . . . 22
  - 4.1.11 V-AQU-VUL-011: Remove unused function . . . 23

Glossary 25

1 Executive Summary

From Feb. 20, 2024 to Feb. 28, 2024, Native-Org engaged Veridise to review the security of their Aqua smart contracts The review covered the contracts for an additional way to provide liquidity for RFQ providers (Traders). Traders can borrow swap tokens from Aqua against collateral and later settle the borrow positions. Veridise conducted the assessment over 18 person-days, with 2 engineers reviewing code over 9 days on commit `0x4333d53`. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual auditing.

Code assessment. The Aqua developers provided the source code of the Aqua contracts for review. The source code for Aqua extends code for Compound V2 with minor modifications by the Aqua developers. It contains some documentation in the form of READMEs and documentation comments on functions and storage variables. The developers also provided us with the document which describes the endpoints of off-chain API.

The source code contained a test suite, which the Veridise auditors noted tests for settlement functions in AquaVault contract.

Summary of issues detected. The audit uncovered 11 issues, including 4 warnings and 2 informational finding were reported. The Aqua developers responded to all the issues.

Of the 11 issues, Native-Org has fixed 9 issues. The high severity issue V-AQU-VUL-001 reports an unsafe type cast that may be exploited to steal funds. The medium issue V-AQU-VUL-003 points to a possible accidental loss of admin rights and is easily fixable. Another medium issue refers to state not updated immediately before a view function is called, see V-AQU-VUL-002 for more details. It was determined to be intended behaviour as state update was performed at other places in the call stack to the view function.

Recommendations. After auditing the protocol, the auditors had a few suggestions to improve Aqua. The auditors recommend expanding the test suite to further test other workflows in the protocol. The auditors recommend adding data validation during initialization of variables. Auditors recommend implementing consistent naming of variables across different components.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Aqua	`0x4333d53`	Solidity	Ethereum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Feb. 20 - Feb. 28, 2024	Manual & Tools	2	18 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Fixed	Acknowledged
Critical-Severity Issues	0	0	0
High-Severity Issues	1	1	1
Medium-Severity Issues	2	1	1
Low-Severity Issues	2	2	2
Warning-Severity Issues	4	3	3
Informational-Severity Issues	2	2	2
TOTAL	11	9	9

Table 2.4: Category Breakdown.

Name	Number
Maintainability	4
Data Validation	3
Logic Error	2
Access Control	2

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of Aqua's smart contracts. In our audit, we sought to answer questions such as:

- Is it possible for an attacker to bypass signature checks?
- Is it possible for an attacker to borrow funds without depositing collateral?
- Is it possible for an attacker to repay less funds than required?
- Is it possible for an attacker assuming the role of trader to steal from other pools?
- Is the interest calculation correct?
- Is the pricing for LP tokens tokens correct?
- Are the upgradable contracts initialized correctly?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a combination of human experts and automated program analysis. In particular, we conducted our audit with the aid of the following technique:

- *Static analysis.* To identify potential common vulnerabilities, we leveraged our custom smart contract analysis tool Vanguard. These tools are designed to find instances of common smart contract vulnerabilities, such as reentrancy and uninitialized variables.

Scope. The scope of this audit is limited to the contracts/ folder of the source code provided by the Aqua developers, which contains the smart contract implementation of the Aqua. This audit covered the following files

- contracts/libraries/*
- contracts/Aqua/*
- contracts/Compound/*

Aqua integrates in and interacts with with the rest of the Native-Org code base. Therefore, parts of following contracts that interact with Aqua are also included in the scope.

- contracts/NativeRouter.sol
- contracts/NativePool.sol
- contracts/NativeRfqPool.sol
- contracts/NativePoolFactory.sol

The protocol also has an off-chain component that calculates the parameters and a signature for those parameters to be provided to the external functions in contract. The off-chain component is out of scope of the audit and is considered as a black box implementation. We assume its implementation is correct unless explicitly stated.

Methodology. Veridise auditors reviewed the reports of previous audits for Aqua, inspected the provided tests, and read the Aqua documentation. They then began a manual audit of the code assisted by static analyzers. During the audit, the Veridise auditors stayed in contact with the Aqua developers to ask questions about the code.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

Vulnerability Report 4

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-AQU-VUL-001	Unsafe type-cast	High	Fixed
V-AQU-VUL-002	getAccountSnapshot() returns stale values	Medium	Intended Behavior
V-AQU-VUL-003	Possible loss of admin access to the contract	Medium	Fixed
V-AQU-VUL-004	Duplicate variable declaration	Low	Fixed
V-AQU-VUL-005	Centralisation Risk	Low	Fixed
V-AQU-VUL-006	Missing non-zero checks	Warning	Fixed
V-AQU-VUL-007	Possible spurious reverts due to missing non-ze...	Warning	Fixed
V-AQU-VUL-008	Unused imports / errors	Warning	Fixed
V-AQU-VUL-009	Missing nonce in widget signatures	Warning	Intended Behavior
V-AQU-VUL-010	Unnecessary visibility modifier	Info	Fixed
V-AQU-VUL-011	Remove unused function	Info	Fixed

4.1 Detailed Description of Issues

4.1.1 V-AQU-VUL-001: Unsafe type-cast

Severity	High	Commit	4333d53
Type	Logic Error	Status	Fixed
File(s)	NativeRfqPool.sol		
Location(s)	tradeRFQT()		
Confirmed Fix At	N/A		

`NativeRfqPool` implements RFQ trades. An user calls `NativeRouter.tradeRFQT()` with signed `quote` received from Native off chain API. The router performs necessary validations, token transfers and calls `NativeRfqPool.tradeRFQT()`. This function verifies the signature of quote and calculates the tokens to be provided to the user and trader.

It then transfers the required tokens from treasury to the user and performs a callback to the `Treasury` by calling `INativeTreasuryV2(treasury).nativeTreasuryCallback()`

When calling the callback, the protocol casts `quote.effectiveSellertokenAmount` and `buyerTokenAmount` from `uint256` to `int`.

This typecast is unsafe when

- `quote.effectiveSellerTokenAmount > int.max()` or,
- `buyerTokenAmount > int.max()`

Such typecasts will cause the values greater than `int.max()` to be wrapped around.

Impact In case of Aqua, this callback is used to update the borrow positions of the trader for `buyerToken` and `sellerToken`.

Carefully chosen values of these variables will cause inconsistencies in the borrow positions of the traders. The positions might change in opposite directions which might lead to traders borrow positions to be reduced without repaying for the borrowed positions. An attacker may utilise this to host a RFQ pool with Aqua liquidity, and request a swap. This will lead to tokens being transferred to attacker without increasing his borrow position.

Recommendation Though there might be mitigation off chain, they may be worked around. Implementing the following checks on chain reduces the attack surface of the protocol

Add the following checks

- `quote.effectiveSellerTokenAmount < int.max()` or,
- `buyerTokenAmount < int.max()`

Developer Response Developers fixed this issue

```
function tradeRFQT(RFQTQuote memory quote) external override onlyRouter {
    /// Trust assumption: the Router has transferred sellerToken.
    if (paused) {
        revert TradePaused();
    }

    address originalBuyerToken = quote.buyerToken;

    quote.buyerToken = quote.buyerToken == address(0) ? weth : quote.buyerToken;
    quote.sellerToken = quote.sellerToken == address(0) ? weth : quote.sellerToken;

    if (!verifySignature(quote)) {
        revert InvalidSignature();
    }

    _updateNonce(quote.nonce);

    uint256 buyerTokenAmount = quote.buyerTokenAmount;
    if (quote.effectiveSellerTokenAmount < quote.sellerTokenAmount) {
        buyerTokenAmount = (quote.effectiveSellerTokenAmount * quote.buyerTokenAmount
    ) / quote.sellerTokenAmount;
    }

    emit RfqTrade(
        quote.recipient,
        quote.sellerToken,
        quote.buyerToken,
        quote.effectiveSellerTokenAmount,
        buyerTokenAmount,
        quote.quoteId,
        quote.signer
    );

    _transferFromTreasury(originalBuyerToken, quote.recipient, buyerTokenAmount);

    if (enableTreasuryCallback) {
        INativeTreasuryV2(treasury).nativeTreasuryCallback(
            quote.signer,
            quote.sellerToken,
            int(quote.effectiveSellerTokenAmount),
            quote.buyerToken,
            int(buyerTokenAmount)
        );
    }
}
```

Snippet 4.1: Snippet from `tradeRFQT()` in NativeRfqPool.sol:132

```
function swapCallback(
      address trader,
      address sellerToken,
      int256 amountIn,
      address buyerToken,
      int256 amountOut,
      mapping(address => AquaLpToken) storage lpTokens,
      mapping(address => mapping(address => int256)) storage positions
  ) external {
      lpTokens[sellerToken].updateNetBorrow(-amountIn);
      lpTokens[buyerToken].updateNetBorrow(amountOut);
      positions[trader][sellerToken] += amountIn;
      positions[trader][buyerToken] -= amountOut;
  }
```

Snippet 4.2: Snippet from `swapCallback()` in AquaVaultLogic.sol:43

4.1.2 V-AQU-VUL-002: getAccountSnapshot() returns stale values

Severity	Medium	**Commit**	4333d53
Type	Logic Error	**Status**	Intended Behavior
File(s)	contracts/Compound/CToken.sol		
Location(s)	getAccountSnapshot()		
Confirmed Fix At	N/A		

Current balance, borrowBalance and exchange rate for any account can be retrieved using `getAccountSnapshot()` shown below.

```
function getAccountSnapshot(address account) external view override returns (uint,
    uint, uint, uint) {
    return (NO_ERROR, accountTokens[account], borrowBalanceStoredInternal(account),
    exchangeRateStoredInternal());
}
```

Snippet 4.3: Snippet from `getAccountSnapshot()` in CToken.sol:144

The returned borrow balance is calculated by calling `borrowBalanceStoredInternal()`.

```
function borrowBalanceStoredInternal(address account) internal view returns (uint) {
    BorrowSnapshot storage borrowSnapshot = accountBorrows[account];
    if (borrowSnapshot.principal == 0) {
        return 0;
    }
    uint principalTimesIndex = borrowSnapshot.principal * borrowIndex;
    return principalTimesIndex / borrowSnapshot.interestIndex;
}
```

Snippet 4.4: Snippet from `borrowBalanceStoredInternal()` in CToken.sol:203

`borrowBalanceStored()` calculates the users `principal+interest` according to

$$\frac{(principal \times borrowIndex)}{users\_borrow\_index}$$

The `borrowIndex` is updated only when interest is accrued.

`getAccountSnapshot()` calls `borrowBalanceStoredInternal()` without calling `accrueInterest()`, which leads to lesser than actual interest calculated for the user.

Impact `getAccountSnapshot()` is used in Comptroller to calculate the health of the an account while calculating liquidity and while exiting the market. Due to above mentioned issue, inflated liquidity of the user will be reported leading to loss of funds for the protocol.

The impact of this issue is proportional to the number of blocks passed since last accrual.

Recommendation Call `accrueInterest()` in `getAccountSnapshot()` before calling `borrowBalanceStored()`.

Developer Response

We have some discussion and review. All the functions that would actually make state changes and that utilize this function `getAccountSnapshot` , calls `accrueInterest()` first. e.g. `mint`, `borrow`, `redeem`, `liquidateBorrow`. We agree that for the view functions, `mintAllowed`, `borrowAllowed` , etc, they might return stale data. But they are not meant to be directly called by users, but by the cToken contracts, where the `accrueInterest` has be called already. I guess the Consideration from Compound would be separating these functions as `view` and make the state changes in the actual operating on

One exception is `exitMarket`, it calls `getAccountSnapshot` and use the borrow amount, but it requires that `amountOwed` needs to be 0, and `accrueInterest` cannot make non-zero borrow amount become 0.

4.1.3 V-AQU-VUL-003: Possible loss of admin access to the contract

Severity	Medium	**Commit**	4333d53
Type	Data Validation	**Status**	Fixed
File(s)	contracts/Aqua/AquaVault.sol		
Location(s)	setAdmin()		
Confirmed Fix At	N/A		

The admin address is an privileged address in the `AquaVault` contract that controls many important parameters of the protocol like

- setting the address of the `NativePool` contract
- setting the `aquaVaultSignatureCheck`

The `admin` address is initialised in the initialiser and later can be updated by calling the `setAdmin()` function from the current admin address.

Note: `setAdmin()` is protected by the onlyAdmin modifier and can be only called by current admin.

```
1 function setAdmin(address newAdmin) external onlyAdmin {
2     admin = newAdmin;
3 }
```

Snippet 4.5: Snippet from `setAdmin()` in AquaVault.sol:105

Impact If the admin is set to zero address by mistake in configuration error, the developers will lose the admin rights on the `AquaVault` without having a way to recover it.

Recommendation Provide a non-zero check in the `setAdmin()`

Developer Response Developers informed us that the want the possibility of admin to be set to zero because they might eventually go for full decentralisation and relinquish admin rights.

Veridise's Response In that case we would recommend a two step process to update the admin address where one call is made to update the new admin and a second call finally makes the update. This way, it is still possible to relinquish admin rights but reduce the risk of it happening accidentally.

4.1.4 V-AQU-VUL-004: Duplicate variable declaration

Severity	Low	**Commit**	4333d53
Type	Maintainability	**Status**	Fixed
File(s)	contracts/Aqua/AquaVault.sol		
Location(s)	AquaVault		
Confirmed Fix At	21b6db2		

The constant variable `EPOCH_UPDATE_COOL_DOWN_SECONDS` is declared in both the `AquaVault` smart contract and in `AquaVaultLogic`, a library imported by `AquaVault`.

```
1 uint256 public constant EPOCH_UPDATE_COOL_DOWN_SECONDS = 8 hours;
```

Snippet 4.6: Snippet from `AquaVault` in AquaVault.sol:21

```
1 uint256 public constant EPOCH_UPDATE_COOL_DOWN_SECONDS = 8 hours;
```

Snippet 4.7: Snippet from `AquaVaultLogic` in AquaVault.sol:22

This duplication may lead to maintenance issues, as outlined below.

Impact Maintaining both declarations of the `EPOCH_UPDATE_COOL_DOWN_SECONDS` variable might cause issues when updating its value. If its value is not changed in both places, inconsistencies may arise.

Recommendation Our recommendation is to declare `EPOCH_UPDATE_COOL_DOWN_SECONDS` in only on place, preferably within `AquaVaultLogic` alongside other constants.

Developer Response Developers fixed this issue

4.1.5 V-AQU-VUL-005: Centralisation Risk

Severity	Low	**Commit**	4333d53
Type	Access Control	**Status**	Fixed
File(s)	Multiple files		
Location(s)	See issue description		
Confirmed Fix At	N/A		

Similar to many projects, Native's `NativePoolFactory` and `NativeRouter` declare an administrator role that is given special permissions. In particular, these administrators are given the following abilities:

- The ability to transfer the tokens out of treasuries of all the pools
- The ability to upgrade the implementation of all the pools
- The ability to change the pricer registry
- The ability to pause the protocol
- The ability to update the widget signers

Impact If a private key were stolen, a hacker would have access to sensitive functionality that could compromise the project. For example, upgrading the implementation of the NativePools is a privileged operation and an attacker can upgrade the implementation to withdraw all tokens from the treasuries of all the pools.

Recommendation As these are all particularly sensitive operations, we would encourage the developers to utilize a decentralized governance or multi-sig contract as opposed to a single account, which introduces a single point of failure.

Developer Response The developers informed that they are already using multi-sig mechanism

4.1.6 V-AQU-VUL-006: Missing non-zero checks

Severity	Warning	**Commit**	4333d53
Type	Data Validation	**Status**	Fixed
File(s)	contracts/Aqua/AquaVault.sol		
Location(s)	See issue description		
Confirmed Fix At	N/A		

Several parameters in protocol in initialised without non-zero checks. These locations are mentioned here

```
aquaVaultSignatureCheck = AquaVaultSignatureCheck(aquaVaultSignatureCheck_);
```

Snippet 4.8: Snippet from `initialize()` in AquaVault.sol:40

```
function setNativePool(address newNativePool) external onlyAdmin {
    nativePool = newNativePool;
  }
```

Snippet 4.9: Snippet from `setNativePool()` in AquaVault.sol:111

```
function setSignatureCheck(address newSignatureCheck) external onlyAdmin {
  aquaVaultSignatureCheck = AquaVaultSignatureCheck(newSignatureCheck);
}
```

Snippet 4.10: Snippet from `setSignatureCheck()` in AquaVault.sol:117

```
function setAllowance(TokenAmountUint[] calldata tokens) external onlyAdmin {
  AquaVaultLogic.setAllowance(tokens, nativePool);
}
```

Snippet 4.11: Snippet from `setAllowance()` in AquaVault.sol:123

```
function setSigner(address _signer) external onlyAdmin {
  signer = _signer;
}
```

Snippet 4.12: Snippet from `setSigner()` in AquaVault.sol:146

```
function setEpochUpdater(address _epochUpdater) external onlyAdmin {
  epochUpdater = _epochUpdater;
}
```

Snippet 4.13: Snippet from `setEpochUpdater()` in AquaVault.sol:152

Impact Setting of zero values will not revert and the protocol will not be reported. But when these values are used, this will lead to spurious reverts.

Recommendation Add non-zero checks while setting these values.

```
function initialize(
    string memory _name,
    address _owner,
    address _signer,
    address _router,
    address _weth,
    address _treasury
) public initializer {
  if (_owner == address(0) || _router == address(0) || bytes(_name).length == 0 ||
    _weth == address(0)) {
    revert ZeroOrEmptyInput();
  }

  __EIP712_init("native pool", "1");

  name = _name;
  owner = _owner;
  router = _router;
  weth = _weth;
  treasury = _treasury;
  poolFactory = msg.sender;
  isSigner[_signer] = true;
}
```

Snippet 4.14: Snippet from `initialize()` in NativeRfqPool.sol:52. Missing non-zero check for `treasury` and `signer`

Developer Response Developers acknowledged the issue

4.1.7 V-AQU-VUL-007: Possible spurious reverts due to missing non-zero check

Severity	Warning	**Commit**	4333d53
Type	Data Validation	**Status**	Fixed
File(s)	contracts/Compound/BaseJumpRateModelV2.sol		
Location(s)	constructor		
Confirmed Fix At	`https://github.com/Native-org/native-contracts/pull/70`		

In the original implementation of the Compound-v2 linked here, `blocksPerYear` is a `constant` contract variable set to `2102400`. In the adapted version of Compound v2 within the Native protocol, this variable is marked as `immutable` and its value is set within the `constructor`. Given that the value to be set is passed as an argument to the `constructor`, `blocksPerYear` could be accidentally set to `0`, as there is no validation for its value. This lack of validation for `blocksPerYear` 's value, could lead to a `division-by-0` vulnerability in the `updateJumpRateModelInternal`. The issue lies in the arithmetic expressions on lines 132-134. Same concern applies to `kink`.

Impact The contract creation will be reverted, as the call to `updateJumpRateModelInternal` will fail.

Recommendation Add data validation checks for `blocksPerYear` and `kink` within the `constructor` . For instance:

```
...
require(blocksPerYear_ != 0 & kink_ != 0, "ERROR");
...
```

Developer Response Developers acknowledged the issue.

4.1.8 V-AQU-VUL-008: Unused imports / errors

Severity	Warning	**Commit**	4333d53
Type	Maintainability	**Status**	Fixed
File(s)	Multiple locations		
Location(s)	See issue description		
Confirmed Fix At	69490e9		

In multiple smart contracts we have identified imports/ errors that are not used, as follows:

- In `contracts/Aqua/AquaVault.sol` → `import {IERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.sol";`
- In `contracts/storage/AquaVaultStorage.sol` → `import {PriceOracle} from "../Compound/PriceOracle.sol";`
- In `contracts/libraries/AquaVaultLogic.sol` → `error InvalidCreditChangeAfterSettlement();, error TokenNoOracle();`

Impact Undesired increased bytecode size.

Recommendation Remove unused imports/ errors.

Developer Response Developers fixed this issue

4.1.9 V-AQU-VUL-009: Missing nonce in widget signatures

Severity	Warning	**Commit**	4333d53
Type	Maintainability	**Status**	Intended Behavior
File(s)	NativeRouter.sol		
Location(s)	verifyRfqWidgetFeeSignature()		
Confirmed Fix At	N/A		

For RFQ trades, a user calls `NativeRouter.tradeRFQT()` with parameters and two signatures procured from Native off-chain component

- A signature for `Quote`
- A signature for `Quote.widgetFee`

```
function verifyRfqWidgetFeeSignature(NativeRfqPool.RFQTQuote memory quote) private
    view returns (bool) {
    bytes32 quoteHash = keccak256(
        abi.encode(
            quote.pool,
            quote.signer,
            quote.recipient,
            quote.sellerToken,
            quote.buyerToken,
            quote.sellerTokenAmount,
            quote.buyerTokenAmount,
            quote.deadlineTimestamp,
            quote.nonce,
            quote.multiHop,
            quote.signature
        )
    );

    bytes32 digest = _hashTypedDataV4(
        keccak256(
            abi.encode(
                RFQ_QUOTE_WIDGET_SIGNATURE_HASH,
                quoteHash,
                quote.widgetFee.signer,
                quote.widgetFee.feeRecipient,
                quote.widgetFee.feeRate
            )
        )
    );

    address recoveredSigner = ECDSAUpgradeable.recover(digest, quote.
    widgetFeeSignature);
    return widgetFeeSigner == recoveredSigner;
}
```

Snippet 4.15: Snippet from `verifyRfqWidgetFeeSignature()` in NativeRounter.sol:509

The signature verification for the `widgetFee` is performed in `verifyRfqWidgetFeeSignature()`. This parameters used in signature for `widgetFee` does not have a nonce. Therefore, a signature replay attack is possible for `widgetFee` .

It is to be noted here that, a replay attack is mitigated here as the

- `widgetFee` is tightly coupled with quote because quote hash is a parameter in widget hash
- the nonce is used in quote hash
- The replay attack will fail when the `Quote` signature is verified.

Impact The protocol is protected against replay attacks due to tight coupling between `quote` signature and `widgetFee` signature. The protocol is upgradable and after upgrades the tight coupling may not exist. In such event, it is error prone if the nonce is not added to `widgetFee` signature and enable attackers to performs replay attacks on widgetFee signatures.

According discussions with Native developers, the `widgetFee` signature is critical as this signature is generated after the backed checks that traders using Aqua have enough collateral to borrow tokens to perform this trade.

Recommendation [If applicable]

Developer Response [If applicable]

4.1.10 V-AQU-VUL-010: Unnecessary visibility modifier

Severity	Info	**Commit**	4333d53
Type	Access Control	**Status**	Fixed
File(s)	contracts/Compound/BaseJumpRateModelV2.sol		
Location(s)	constructor()		
Confirmed Fix At	1769b0b		

Since the `BaseJumpRateModelV2` is an abstract contract, the visibility of the `constructor` (currently marked as `internal`) will be ignored. More details can be found on the Solidity's breaking changes page linked here.

Impact NaN

Recommendation Remove the `internal` visibility modifier from the `constructor`'s definition on `L54`.

Developer Response Developers fixed the issue

4.1.11 V-AQU-VUL-011: Remove unused function

Severity	Info	**Commit**	4333d53
Type	Maintainability	**Status**	Fixed
File(s)	contracts/Compound/CToken.sol		
Location(s)	See issue description		
Confirmed Fix At	c9cd93a		

In the adapted version of Compound v2 within the Native protocol, the implementation of `borrowRatePerBlock` in the `CToken` contract has been removed. Given that there is no use of `borrowRatePerBlock` in `CToken`, there is no need to override it in `CToken` too.

Impact Undesirable increased bytecode size.

Recommendation Remove the `borrowRatePerBlock` function from `CToken` as it is already overridden in `AquaLpToken`.

Developer Response Developers fixed the issue

Glossary

smart contract A self-executing contract with the terms directly written into code. Hosted on a blockchain, it automatically enforces and executes the terms of an agreement between buyer and seller. Smart contracts are transparent, tamper-proof, and eliminate the need for intermediaries, making transactions more efficient and secure.. 1