Quantstamp

Native V2 (Security Review)

Executive Summary

This security review was prepared by Quantstamp, the leader in blockchain security.

Type	DeFi
Timeline	2024-12-16 through 2025-01-06
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	README.md
Source Code	• Native-org/v2-core #c2fdeab
Auditors	• Joseph Xu Technical R&D Advisor • Valerian Callens Senior Auditing Engineer

Documentation quality	Low
Test quality	Medium
Total Findings	18 Fixed: 6 Acknowledged: 9 Mitigated: 3
High severity findings	3 Fixed: 2 Acknowledged: 1
Medium severity findings	6 Fixed: 3 Acknowledged: 3
Low severity findings	5 Fixed: 1 Acknowledged: 3 Mitigated: 1
Undetermined severity findings	3 Acknowledged: 2 Mitigated: 1
Informational findings	1 Mitigated: 1

Summary of Findings

Final Report (2025-01-06): Quantstamp has reviewed additional responses by Native team, including an updated commit hash `9430af6`. The team has Fixed another issue NATv2-1, which was previously Acknowledged to have a fix incoming. As of now, the great majority of the issues identified in the Initial Report are Fixed, Mitigated, or Acknowledged with fixes/mitigations actively planned.

Updated Report 2 (2025-01-03): Quantstamp has reviewed additional responses by Native team, including an updated commit hash `5a3b5b2`. The team has Fixed and Mitigated several issues on top of what was done in the Updated Report, including a few Auditor Suggestions. As of now, the great majority of the issues identified in the Initial Report are Fixed, Mitigated, or Acknowledged with fixes/mitigations actively planned.

Updated Report 1 (2025-01-01): Quantstamp has reviewed the responses by Native team, including an updated commit hash `8e053f14`. The team Fixed or Mitigated 5 issues, and Acknowledged most other issues with concrete plans to improve the protocol's overall security (with 1 High severity issue removed as it was identified to be False Positive). The Acknowledged issues mostly deal with off-chain signing component and management of privileged addresses, both of which would require a separate audit. To this end, Native team plans to take the following actions to strengthen the overall security of the protocol: (1) consult a security expert regarding the off-chain component, (2) conduct additional rounds of audit that includes the off-chain component, (3) ensure that no single point of failure exists between the off-chain validation and on-chain settlement procedures, and (4) consider adding a hard limit for token transfer out of the protocol to limit the damage in case of potential exploits.

Initial Report (2024-12-20): Quantstamp conducted a 3-day security review of the Native v2 smart contracts. This was not a full audit of the protocol but a time-boxed effort to perform quick security checks and review the most critical paths. Our approach focused initially on the high-level architecture, then delved into critical paths after understanding the most important flows. We primarily concentrated on the critical functions of the following contracts: `CreditVault.sol`, `LPToken.sol`, `NativeRFQPool.sol`, and `NativeRouter.sol`. The team has simplified the architecture, added new features, and re-wrote part of the existing features from Native v1. It should be noted that there are centralized off-chain components that play critical roles. For the purpose of this security review, we have assumed that the off-chain components will perform correctly in formatting the data, but we have not assumed data integrity or validity.

Despite the nature of the time-boxed security review, the auditors identified 19 issues, with 4 High severity and 6 Medium severity issues that can potentially impact protocol funds, user funds, general usability, and code correctness. In addition, two issues identified in the previous Quantstamp security review are still present, and are included in this report (NATv2-7 is escalated to Medium severity and NATv2-18 still as Low severity). It is likely that with additional time and auditor effort, more High or Medium severity issues may be identified. Nevertheless, the repository provided still appears to be in development, and the overall security will be improved, with ample opportunities to address each of the

issues identified. Quantstamp strongly recommends addressing the issues identified in this report and arranging for a full audit before deploying these smart contracts in production.

ID	DESCRIPTION	SEVERITY	STATUS
NATv2-1	Any Whitelisted Trader Can Settle Trades and Remove Collateral on Behalf of Another Trader	High	Fixed
NATv2-2	Incorrect Updates of `fundingFees`	High	Fixed
NATv2-3	Fake `LPToken.sol` contract instances can be used to drain underlying tokens	High	Acknowledged
NATv2-4	An Address Gets Added to the `poolArray` Every Time `setNativePool()` Is Called	Medium	Fixed
NATv2-5	The Exchange Rate Can Increase by More than 1% During the Epoch Update	Medium	Acknowledged
NATv2-6	Data Signed Off-Chain May Be Out of Sync with the On-Chain State or the Current Market Condition	Medium	Acknowledged
NATv2-7	Adding Collateral to Non-Trader Addresses Is Possible	Medium	Fixed
NATv2-8	Unclear Specifications of Fee Accounting and Associated Invariants	Medium	Fixed
NATv2-9	The Maximum Allowed Widget Fee Rate May Be Too High	Medium	Acknowledged
NATv2-10	`NativeRouter.sol` Does Not Always Wrap Incoming ETH in `_transferSellerToken()`	Low	Acknowledged
NATv2-11	Possible to Renounce Ownership of Contracts	Low	Acknowledged
NATv2-12	Risks of Supporting Non-Standard ERC-20 Tokens	Low	Mitigated
NATv2-13	Some Important Updates of the System Do Not Emit Events	Low	Fixed
NATv2-14	Potential Underpayment to Recipient in `externalSwap()`	Low	Acknowledged
NATv2-15	Possible Division by Zero	Informational	Mitigated
NATv2-16	Possible Empty Market Attack	Undetermined	Mitigated
NATv2-17	Roles with Strong Privileges	Undetermined	Acknowledged
NATv2-18	Unchecked Values in `RFQTQuote` Inputs	Undetermined	Acknowledged

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Disclaimer

Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

> This was a time-boxed effort and should not be considered a full audit. A full smart contract security audit would be necessary to obtain a comprehensive assessment of how all components work together. The auditors did not perform detailed examinations of the protocol's off-chain components. Due to the nature of the time-boxed security review, external integrations were not thoroughly examined, nor were configuration and deployment issues that could introduce security risks.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

Files Included

Repo: https://github.com/Native-org/v2-core/(c2fdeabc3f1232d8cdd033c9f486cd6ffcecfe89) Files: src/libraries/ConstantsLib.sol; src/libraries/ErrorsLib.sol; src/libraries/ReentrancyGuardTransient.sol; src/libraries/TStorage.sol; src/CreditVault.sol; src/LPToken.sol; src/NativeRFQPool.sol; src/NativeRouter.sol

Operational Considerations

1. The solvency of the protocol is not based on permissionless liquidations. As a result, it only depends on the availability and reactivity of the addresses allowed to perform liquidations.
2. The same underlying asset cannot be supported by more than one market in the same contract `CreditVault.sol`, due to its mapping `lpTokens[]`.

Findings

NATv2-1

Any Whitelisted Trader Can Settle Trades and Remove Collateral on Behalf of Another Trader

High Fixed

Update

`CreditVault.sol` has been updated. Only the trader or the authorized settler can perform actions such as settling trades or removing collaterals from the associated address as of commit `9430af6`.

File(s) affected: `src/CreditVault.sol`

Description: In the `CreditVault.sol` contract, the operations `settle()` and `removeCollateral()` are protected by the modifier `onlyTraderOrSettler`, which has the following code:

```
modifier onlyTraderOrSettler(address trader) {
 if (!isTraders[msg.sender] && !(isTraders[trader] && msg.sender == settlers[trader])) {
 revert ErrorsLib.OnlyTrader();
 }
 _;
}
```

We can observe that having `isTraders[msg.sender] == true` is enough to bypass this condition.

Exploit Scenario:

1. `Alice` is a whitelisted trader and liquidator, and `Bob` is a whitelisted trader. `Alice` can increase the likelihood of `Bob` getting liquidated by making his positions more risky, by successfully getting that request signed by the `signer`.
2. `Alice` and `Bob` are traders. `Alice` settles the position of `request.trader == Bob` and `request.recipient == Alice`. She gets that request signed by the `signer`. She calls `settle()` with that request and the contract `CreditVault.sol` will transfer the negative requested `amounts` to her.

Recommendation: Consider updating the code of the modifier to enforce that only a trader or an authorized settler of that trader can settle trades or remove collaterals from the associated address.

NATv2-2 Incorrect Updates of `fundingFees`

High Fixed

Update

Native team has fixed the issue, with additional modification to clarify the flow of fees to LP token holders and to the protocol itself. The portion of the fee that goes to the LP token holders is named `fundingFee` and the portion of the fee that goes to the protocol itself is named `reserveFee`.

File(s) affected: `src/CreditVault.sol`

Description: In `CreditVault.sol`, the storage variable `fundingFees` is used as an accounting variable for the funding fees accumulated when the function `epochUpdate()` is executed. However, the same variable is used to keep track of funding fees accumulated through different assets. This causes two problems (i) there is information loss on how much fee is available for each token, and (ii) there is a high risk of miscalculation due to the precision mismatch in the tokens. As an example, if `1e18 ETH` and `1e6 USDC` are accounted for in the contract as funding fees (with `fundingFees == 1e18 + 1e6`) that also owns `1e18 USDC`, it is possible for `feeWithdrawer` to withdraw `1e18 USDC` from the contract via `withdrawFundingFees()`.

In addition, the address `feeWithdrawer` can withdraw any amount of underlying asset to any arbitrary recipient address via the function `withdrawFundingFees()`. However, there is no mechanism to make sure that the underlying token is supported by the contract (which could result in incorrectly reducing the value of `fundingFees`, resulting in the loss of these fees).

Recommendation: Change the `uint256 fundingFees` to `mapping(address => uint256) fundingFeesByToken` to track funding fees available by each asset.

NATv2-3 Fake `LPToken.sol` contract instances can be used to drain underlying tokens

High Acknowledged

Update

Native team considers this risk to be minimal, as the `supportMarket()` function is only callable by the owner of the `CreditVault.sol` contract. At the same time, the team acknowledges the importance of privileged addresses in guaranteeing the security of the whole protocol. They plan to re-evaluate the overall protocol security posture to ensure that no single point of failure exists on these privileged addresses that can cause a single compromise to exploit the whole system.

File(s) affected: `src/CreditVault.sol`, `src/LPToken.sol`

Description: The `CreditVault.sol` contract uses the `supportMarket()` function to support the market corresponding to a `LPToken.sol` contract instance. Once the market is supported, the `LPToken.sol` address can withdraw its underlying ERC20 token to an arbitrary `to` address.

While `supportMarket()` function performs various checks on the `LPToken.sol` contract instance, none of the checks can correctly identify if the instance in question is indeed deployed by the Native team or its partners. Adding a fake, malicious instance of `LPToken.sol` contract may lead to all of the ERC20 tokens in `CreditVault.sol` being drained.

Recommendation: Use a factory contract to keep track of the true deployed instances of `LPToken.sol`. Additionally, implement additional checks in `transferOut()` function so that it is not possible to transfer tokens to an arbitrary `to` address.

NATv2-4
An Address Gets Added to the `poolArray` Every Time `setNativePool()` Is Called

Medium | Fixed

> **Update**
> The variable `poolArray` has been removed and the function `setNativePool()` has been updated accordingly.

File(s) affected: `src/CreditVault.sol`

Description: `CreditVault.sol` contract has a function `setNativePool()` that is used to toggle the whitelist status of an instance of `NativeRFQPool.sol` contract. However, every call to the function, whether it's activating, de-activating, or no-op, would push the pool address in the function argument to the `poolArray`, which is the storage list used to keep track of the whitelisted addresses for the `NativeRFQPool.sol` instances. This does not seem aligned with the expected behavior.

Recommendation: Re-implement the `setNativePool()` function to (i) remove the de-whitelisted pool address from the `poolArray`, and (ii) not re-add existing whitelisted pool address to the `poolArray`. While working on this fix, please keep in mind the risk of out-of-gas error.

NATv2-5
The Exchange Rate Can Increase by More than 1% During the Epoch Update

Medium | Acknowledged

> **Update**
> Native team considers this risk to be minimal, as the `epochUpdate()` function is only callable by the `epochUpdater` address. At the same time, the team acknowledges the importance of privileged addresses in guaranteeing the security of the whole protocol. They plan to re-evaluate the overall protocol security posture to ensure that no single point of failure exists on these privileged addresses that can cause a single compromise to exploit the whole system.

File(s) affected: `src/CreditVault.sol`

Description: In the function `epochUpdate()`, the address `epochUpdater` can update the funding fees for traders at the end of each epoch. When browsing the assets to update for a given trader, a check makes sure that the exchange rate of the LP token does not increase by more than `1%`. However, it is possible to bypass this check and have an increase of more than `1%` by adding multiple times the same asset in the list.

Recommendation: Consider making sure that there is no duplicated token in the data structure `accruedFees`. For instance, it can be done by enforcing the fact that assets must be sorted by address.

NATv2-6
Data Signed Off-Chain May Be Out of Sync with the On-Chain State or the Current Market Condition

Medium | Acknowledged

> **Update**
> Native team acknowledges the importance of the off-chain backend API in guaranteeing the security of the whole protocol. They plan to take the following actions to strengthen the overall security of the protocol: (1) consult a security expert regarding the off-chain component, (2) conduct additional rounds of audit that includes the off-chain component, (3) ensure that no single point of failure exists between the off-chain validation and on-chain settlement procedures, and (4) consider adding a hard limit for token transfer out of the protocol to limit the damage in case of potential exploits.

File(s) affected: `src/CreditVault.sol`, `src/NativeRFQPool.sol`, `src/NativeRouter.sol`

Description: Many of the functionalities in Native's system relies on processing transactions with calldata signed off-chain. These calldata include price quotations, request to remove collaterals, and requests for liquidations. If the calldata is processed on-chain at a sufficiently later time than the signature, it could lead to various adverse consequences due to the on-chain state or the market condition being very different from those at the time of signature generation. Some of the adverse effects include:

- Traders being able to trade more than their capacity by exploiting the timestamp differences between `addCollateral()`, `tradeRFQT()`, and `removeCollateral()`.
- The Native pools executing bad trades when market moves very quickly, particularly through the auto-sign feature.

This risk was initially brought up with the Native team during the v1 security review, and the team considered this risk to be rather unlikely due to the fact that only trusted partners will be involved. However, there have been instances since the v1 security review where protocols have been

exploited due to the off-chain signing mechanism being out of sync with the on-chain state or the market condition. Therefore, the auditors have chosen to highlight this issue again and increase the severity to Medium as this risk is concrete.

Recommendation: Add checks on important procedures such as swaps and liquidations to verify the on-chain state and current market conditions. In addition, the off-chain signatures should expire relatively quickly - an appropriate level needs to be determined based on block time, market conditions, and usability.

NATv2-7 Adding Collateral to Non-Trader Addresses Is Possible

Medium Fixed

> **Update**
> Collateral can only be added to whitelisted trader address as of commit `5a3b5b2` .

File(s) affected: `src/CreditVault.sol`

Description: In the `CreditVault.sol` contract, anyone can call the function `addCollateral()` to add collateral for another address. However, there is no check to make sure that the receiver address is a whitelisted trader in the system. If this is not the case, tokens sent as collateral may remain locked in the contract.

Recommendation: Consider making sure that the receiver address is a whitelisted `trader` .

NATv2-8
Unclear Specifications of Fee Accounting and Associated Invariants

Medium Fixed

> **Update**
> Native team has modified the code to clarify the flow of fee accumulation to both LP token holders and the protocol itself (included in the fix for issue NATv2-2 Incorrect Updates of `fundingFees`). Additional test cases have also been written to address this issue.

File(s) affected: `src/CreditVault.sol` , `src/LPToken.sol`

Description: The documentation and code comments indicate that the protocol is supposed to generate revenue by accumulating funding fees and also be able to distribute this to holders of LP tokens. The mechanisms by which this happens is not obvious from the code and in some cases the solvency of the fee mechanism is unclear.

As an example, the mechanism by which LP Token gains fees is through `LPToken.distributeYield()` external call that is made on each `CreditVault.epochUpdate()` function call. However, the `distributeYield()` function simply updates the total underlying token amount of the LP token without actually transferring token.

In addition, the auditors suspect that there may need to be some sort of invariant between `lpFee` and `fundingFee` such as `lpFee` < `fundingFee` because the LP fee actually originates from the funding fee accumulated. No such check takes place on-chain, and it is assumed that `accruedFees` input must be correct.

Recommendation: Improve the documentation and specifications on how fees are generated and distributed. Update the code to include invariant checks on-chain if needed.

NATv2-9
The Maximum Allowed Widget Fee Rate May Be Too High

Medium Acknowledged

> **Update**
> Native team has indicated that they plan to set a maximum widget fee rate around `2000` .

File(s) affected: `src/NativeRouter.sol`

Description: The market makers facilitating trade on Native can charge a fee called the `widgetFee` from the swappers. The current maximum value allowed for this parameter is `10000` , which represents 100% of the swapper's input token amount being charged as fee. This should be lowered to mitigate the risk of a market maker overcharging the swapper from fees.

Recommendation: Set a more reasonable maximum for the widget fee rate between `1000` to `3000` .

NATv2-10 `NativeRouter.sol` Does Not Always Wrap Incoming ETH in `_transferSellerToken()`

Low | Acknowledged

> **Update**
> Native team has chosen not to modify the code given that any excess ETH that remain in the `NativeRouter.sol` contract can be refunded.

File(s) affected: `src/NativeRouter.sol`

Description: The `NativeRouter.sol` contract uses an internal function `_transferSellerToken()` to facilitate the transfer of token from the swapper (i.e., seller) to the market maker's treasury or the `NativeRouter.sol` contract. The contract handles the case where `msg.value > 0` (i.e., if the swapper wants to sell native ETH) by first wrapping the swapper's native ETH as WETH and then completing the transfer. However, there is another boolean argument `multiHop` used to enter this branch, and the wrapping only takes place if `msg.value > 0` AND `multiHop == false`.

As a result, the native ETH will not be wrapped if `multiHop == true` even when the swapper somehow transfers ETH through the function call (`msg.value > 0`), resulting in excess ETH being held by the `NativeRouter.sol` contract that the contract owner must return using the `refundETH()` function.

Recommendation: The intended usage and the specification of the boolean argument `multiHop` is unclear as it likely has to do with the off-chain component, which was out of the scope of this audit. We recommend adding an `else if` statement in the `_transferSellerToken()` function to properly handle the case when `msg.value > 0` AND `multiHop == true`. If the conditions `msg.value > 0` and `multiHop == true` are not supposed to both hold at the same time, a check should be added so that the transaction reverts.

NATv2-11 Possible to Renounce Ownership of Contracts

Low | Acknowledged

> **Update**
> Native team has indicated that all of their contract owners are multi-signature addresses, similar to most DeFi protocols. There is an implicit understanding among the signers that ownership should not be renounced because doing so would make it impossible to update contract parameters or states.

File(s) affected: `src/CreditVault.sol`, `src/LPToken.sol`, `src/NativeRFQPool.sol`, `src/NativeRouter.sol`

Description: The `CreditVault.sol`, `LPToken.sol`, `NativeRFQPool.sol`, and `NativeRouter.sol` contracts allow the owner to renounce ownership. While this functionality is standard, it poses a significant risk if triggered accidentally or maliciously, as the contracts rely on the owner's role to perform critical network functions. Once ownership is renounced, the associated privileges are permanently lost.

Recommendation: Consider overriding the `renounceOwnership()` function in the child contract to disallow it entirely, wherever necessary.

NATv2-12 Risks of Supporting Non-Standard ERC-20 Tokens

Low | Mitigated

> **Update**
> Native team is aware of the consequences of supporting non-standard ERC20 tokens. The team has indicated that at this moment they only plan to support stETH or wstETH.

File(s) affected: `src/CreditVault.sol`, `src/LPToken.sol`

Description: Supporting tokens with specific features such as fees, rebasing, pausable, upgradeable, blacklistable, or hooks on transfers could negatively impact the main flows of the system if no specific mitigation measure is enforced to limit the consequences.

Exploit Scenario:
Consider analyzing thoroughly from a security perspective any token you would like to be supported.

NATv2-13 Some Important Updates of the System Do Not Emit Events

Low | Fixed

> **Update**
> Important contract state changes now emit events.

File(s) affected: `src/CreditVault.sol`, `src/NativeRouter.sol`

Description: The following updates of the contract's state do not result in emitting an event, making it harder for observers (team or users) to track important contract state changes (intended or not) and react fast if something unexpected is identified:

1. Updates to privileged addresses in `CreditVault.sol` (`liquidator` , `signer` , `epochUpdated` , and `feeWithdrawer`);
2. Execution of the functions `NativeRouter.refundERC20()` and `NativeRouter.unwrapWETH9()` ;
3. Adding routers to the whitelist in `NativeRouter.setWhitelistRouter()` ;

Recommendation: Consider emitting the suggested events.

NATv2-14 Potential Underpayment to Recipient in `externalSwap()`

• Low ⓘ Acknowledged

> **Update**
> Native team has indicated that the current implementation is the intended design. Any residual ERC20 token in the `NativeRouter.sol` contract can be refunded using the `refundERC20()` function.

File(s) affected: `src/libraries/ExternalSwapRouter.sol`

Description: In the `ExternalSwapRouterUpgradeable.externalSwap()` function, there are two variables, `recipientDiff` and `routerDiff` , that record the token balance differences before and after the external contract call. However, if both values are non-zero (positive) and `recipientDiff >= routerDiff` , the `amountOut` will ignore the `routerDiff` and consider only `recipientDiff` . The intention might be that the `routerDiff` should also be transferred to the recipient and counted as part of the `amountOut` , as the function does not intend to collect funds for the router.

In practice, we believe that only one of the two variables (`recipientDiff` and `routerDiff`) will be non-zero. The idea is likely to support external swap contracts that either swap to the `msg.sender` (which is the router contract in this case) or directly to the recipient.

Recommendation: Consider changing the condition `if (recipientDiff < routerDiff)` to `if (routerDiff > 0)` . Alternatively, add a validation that only one of `recipientDiff` and `routerDiff` can be non-zero, if that is the intention.

NATv2-15 Possible Division by Zero

• Informational ⓘ Mitigated

> **Update**
> Native team has indicated that the current implementation as of commit `5a3b5b2` maintains the invariant such that `totalShares == 0` implies `totalUnderlying == 0` and vice versa. The current implementation implicitly checks the denominator based on this invariant and other conditions (e.g., checking for `sharesToBurn > 0` AND `shares[msg.sender] > sharesToBurn` in the `redeem()` function). This means that the current implementation will not have division by zero error even if the denominators are not explicitly checked.

File(s) affected: `src/LPToken.sol`

Description: The variables `totalUnderlying` and `totalShares` are both used as denominators in critical functions. While the denominators are checked to prevent division by zero in most places, this check is missing in some places. Specifically, the functions `deposit()` and `getSharesByUnderlying()` do not check against `totalUnderlying == 0` , while the function `redeem()` does not check against `totalShares == 0` .

Recommendation: We recommend implementing checks so that denominators are never zero throughout the code to prevent problems that might arise from an unintended divide by zero operation.

NATv2-16 Possible Empty Market Attack

• Undetermined ⓘ Mitigated

> **Update**
> Native team has provided details on their market initialization process and how it mitigates the risk of empty market attacks. In short, the trading pairs must be registered with the Native backend, and therefore unsupported LP trades or withdrawals cannot be initiated.
>
> Native team acknowledges the importance of the off-chain backend API in guaranteeing the security of the whole protocol. They plan to take the following actions to strengthen the overall security of the protocol: (1) consult a security expert regarding the off-chain component, (2) conduct additional rounds of audit that includes the off-chain component, (3) ensure that no single point of failure exists between the off-chain validation and on-chain settlement procedures, and (4) consider adding a hard limit for token transfer out of the protocol to limit the damage in case of potential exploits.

File(s) affected: `src/CreditVault.sol`

Description: The `CreditVault.sol` contract enables trading in a market through the `supportMarket()` function. This function adds the market corresponding to a `LPToken.sol` contract instance. However, no checks or initialization takes place within this function to ensure that

the market is indeed ready to be traded.

The exact impact of this issue is unclear because documentation or specification for this procedure is unavailable. The conditions and invariant that must hold for the market to be tradable is also unclear and not documented. Given the lack of information we have marked it as an Undetermined, as there is still the possibility of system failure due to a transaction being called to a market that has not been properly initialized.

Recommendation: Ensure that the market is properly initialized before activating trading functionalities after the market has been supported in `CreditVault.sol`.

NATv2-17 Roles with Strong Privileges

• **Undetermined** Acknowledged

> **Update**
> Native team acknowledges the importance of privileged addresses in guaranteeing the security of the whole protocol. They plan to re-evaluate the overall protocol security posture to ensure that no single point of failure exists on these privileged addresses that can cause a single compromise to exploit the whole system.

File(s) affected: `src/CreditVault.sol`, `src/LPToken.sol`

Description: The protocol assigns special privileges to several addresses. The privileges assigned to these addresses are rather strong, and may lead to a significant amount of lost funds if compromised. In addition, there is no mechanism to challenge the actions of these strongly privileged addresses (e.g., governance).

We list the strongly privileged addresses and their privileges below:

1. The address `signer` in `CreditVault.sol` contract can sign any request to settle positions, remove collateral or liquidate positions. As a result, if compromised, it can:
    - sign malicious requests that will drain assets from users;
    - front-run as a liquidator a profitable liquidation opportunity submitted by another liquidator.
2. The address `owner` in `CreditVault.sol` contract can add a new market, enable/disable a supported market, allow a Native pool to spend tokens owned by the contract, whitelist/de-whitelist trader addresses and their associated settler addresses, whitelist/de-whitelist liquidator addresses, update the `signer` address, update the `epochUpdater` address, and update the `feeWithdrawer` address. As a result, if compromised, it can:
    - drain the protocol by whitelisting a malicious contract as a Native pool via `setNativePool()`, then allowing it to spend tokens owned by the contract via `setAllowance()`, and finally drain the vault from the malicious contract.
3. The protocol relies on the address `epochUpdater` in `CreditVault.sol` to call `CreditVault.epochUpdate()` on time to properly update funding fees and collect/distribute accumulated fees.
4. The address `owner` in `LPToken.sol` contract can:
    - transfer and renounce ownership of the token;
    - pause/unpause the contracts, which will impact deposits and redeems;
    - update the minimum amount for deposits to any `uint256` value;
    - update to any `uint256` value the minimum interval to wait without sustaining early withdrawal penalties;
    - update the fee for early withdrawals from `0` to `100%`;
    - temporarily pause only redeems by setting `minRedeemInterval` with a high value and the early withdrawal fee to `100%`;
    - temporarily pause only deposits by setting `minDeposit` with a high value.

Recommendation: Provide details on the intended level of privileges, specifications, and operational security measures taken with regards to these privileged addresses. It is recommended to distribute privileges across addresses and also distribute control (e.g., to a governance multi-sig) to ensure that there is no single point of failure for these privileged addresses. Moreover, we suggest adding some mechanism to timelock, challenge, or roll back decisions by these strongly privileged addresses.

NATv2-18 Unchecked Values in `RFQTQuote` Inputs

• **Undetermined** Acknowledged

> **Update**
> Native team acknowledges the importance of the off-chain backend API in guaranteeing the security of the whole protocol. They plan to take the following actions to strengthen the overall security of the protocol: (1) consult a security expert regarding the off-chain component, (2) conduct additional rounds of audit that includes the off-chain component, (3) ensure that no single point of failure exists between the off-chain validation and on-chain settlement procedures, and (4) consider adding a hard limit for token transfer out of the protocol to limit the damage in case of potential exploits.

File(s) affected: `src/NativeRFQPool.sol`, `src/NativeRouter.sol`, `src/interfaces/IQuote.sol`

Description: Many of the functionalities in Native's system relies on processing transactions with calldata signed off-chain. The format of the calldata is specified in `src/interfaces/IQuote.sol` contract under `struct RFQTQuote`. While we have generally assumed that the data is formatted correctly when the off-chain signatures are generated, we have not assumed data validity or data integrity. There are certain instances where additional checks on the values of the `RFQTQuote memory quote` input value may be useful. These instances are as follows:

- The nonce check is missing in `NativeRouter.tradeRFQT()`, even though it is checked in `NativeRouter.tradeAutoSign()`.
- In `NativeRouter._verifyAutoSignature()`, the values `quote.pool`, `quote.multiHop`, `quote.widgetFee`, `quote.effectiveSellerTokenAmount` are not checked.

- In `NativeRouter._verifyWidgetSignature()`, the values `quote.quoteID` and `quote.amountOutMinimum` are not checked.
- In `NativeRFQPool._verifyRFQSignature()`, the values `quote.pool`, `quote.effectiveSellerTokenAmount`, `quote.multihop`, `quote.widgetFee`, `quote.widgetFeeSignature`, `quote.externalSwapCalldata`, and `quote.amountOutMinimum` are not checked.

Recommendation: We recommend creating a detailed documentation or specification for the off-chain signatures. The documentation should indicate whether or not these parameters should be included in the signature, as well as whether or not these parameters should be verified (and the conditions to be checked). A detailed documentation can help prevent users from adding unnecessary parameters (whether intentional or not) to change the intended behavior. Furthermore, we recommend adding on-chain checks to check for valid quote parameters where necessary.

Auditor Suggestions

S1
Inconsistency in Reverting or Returning False on ECDSA Signature Mismatch

Fixed

File(s) affected: `src/NativeRFQPool.sol`

Description: The protocol has several internal functions that are used to verify ECDSA signature throughout `CreditVault.sol`, `NativeRFQPool.sol`, and `NativeRouter.sol` contracts. Most of these internal functions will revert if there is a signature mismatch. However, `NativeRFQPool._verifyRFQSignature()` does not revert but instead returns `false` and then reverts in the calling function.

Recommendation: We recommend reverting within the `_verifyRFQSignature()` to be consistent with other internal functions.

S2 No Limit on the Length of RFQ Pool Name

Acknowledged

> **Update**
> Native team confirms that they do not consider long RFQ pool names to be an issue.

File(s) affected: `src/NativeRFQPool.sol`

Description: The constructor of the `NativeRFQPool.sol` contract does not check for a maximum length of the `name` variable.

Recommendation: Include a check in the constructor to revert if the `_name` argument is excessively long.

S3 Missing Input Validation in `refundETH()`

Fixed

> **Update**
> The function `refundERC20()` will now send the minimum of the contract's ETH balance and the `amount` requested.

File(s) affected: `src/NativeRouter.sol`

Description: `NativeRouter.sol` contract has an emergency function `refundETH()` that is used to send native ETH token to a certain address. This function does not validate its input for the amount of token to be transferred so that it does not exceed the `NativeRouter.sol` contract's current ETH balance. While the lack of input validation does not lead to negative consequences, the error message returned in this case is uninformative.

Recommendation: We recommend validating the input `amount` so that it does not exceed the current contract ETH balance. This allows for reverting more informatively with `ErrorsLib.InsufficientTokenAmount();` as is done in the `refundERC20()` function.

Alternatively, the error message in `TransferHelper.safeTransferETH()` can be improved so that it is more infromative.

S4 Usage of `safeApprove()` Is Deprecated

Acknowledged

> **Update**
> Native team is aware of the deprecation of `safeApprove()`. The team indicated that the alternative `safeIncreaseAllowance()` and `safeDecreaseAllowance()` functions are not suitable for their needs, and therefore chose to modify the `safeApprove()` function within the `SafeERC20.sol` contract to match their requirement instead.

File(s) affected: `src/CreditVault.sol`

Description: The function `CreditVault.setAllowance()` uses `safeApprove()` that is deprecated: https://docs.openzeppelin.com/contracts/4.x/api/token/erc20#SafeERC20-safeApprove-contract-IERC20-address-uint256-

Recommendation: Consider not using `safeApprove()`.

S5 Unlocked Pragma

Fixed

File(s) affected: `src/LPToken.sol`, `src/libraries/ConstantsLib.sol`, `src/libraries/ErrorsLib.sol`, `src/libraries/ExternalSwap.sol`, `src/libraries/ReentrancyGuardTransient.sol`, `src/libraries/TStorage.sol`

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.*`. The caret (`^`) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked". The pragma used in the repositories are all unlocked and inconsistent, ranging from `^0.8.4` all the way to `^0.8.24`.

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend removing the caret and standardizing the Solidity version across all smart contracts in the repositories.

S6 Code Conciseness

Fixed

File(s) affected: `CreditVault.sol`, `NativeRFQPool.sol`, `NativeRouter.sol`

Description:

1. Two different libraries of `SafeCast` are used (`@openzeppelin/contracts/utils/math/SafeCast.sol` in `CreditVault.sol` and `./libraries/SafeCast.sol` in `NativeRouter.sol`).
2. In `NativeRFQPool.sol`, `Context.sol` is imported but not used.
3. The value of `address(lpToken.underlying())` is accessed twice in `CreditVault.supportMarket()`, instead of using the local variable `underlying`.
4. The keyword `payable` can be removed from the functions of `NativeRouter.sol` where it is not used.

Recommendation: Consider addressing these items.

S7 Gas Optimization

Mitigated

> **Update**
> The keyword `immutable` has been added to the storage variable `WETH9` in `NativeRFQPool.sol` and `NativeRouter.sol` contracts.

File(s) affected: `CreditVault.sol`, `NativeRFQPool.sol`, `NativeRouter.sol`

Description:

1. In `NativeRouter.sol`, the keyword `immutable` can be used for the storage variables `WETH9` and `vault` to save gas.
2. In `NativeRFQPool.sol`, the keyword `immutable` can be used for the storage variables `name`, `WETH9`, and `router` in order to save gas.
3. In `CreditVault.sol`, the value of `positionsUpdates.length` can be cached in the local variable before the `for` loop.

Recommendation: Consider addressing these items.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.

- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Files

- `cc1...1e1 ./src/NativeRFQPool.sol`
- `181...93e ./src/NativeRouter.sol`
- `40b...33b ./src/CreditVault.sol`
- `d3a...eff ./src/LPToken.sol`
- `9eb...c9b ./src/interfaces/ICreditVault.sol`
- `01d...629 ./src/interfaces/IWETH9.sol`
- `615...516 ./src/interfaces/INativeRouter.sol`
- `6cd...358 ./src/interfaces/INativeRFQPool.sol`
- `da2...859 ./src/interfaces/IQuote.sol`
- `247...1a2 ./src/libraries/Order.sol`
- `385...cda ./src/libraries/TStorage.sol`
- `b1a...4ae ./src/libraries/ConstantsLib.sol`
- `5b7...e4c ./src/libraries/Multicall.sol`
- `043...b6d ./src/libraries/ErrorsLib.sol`
- `cc0...b7c ./src/libraries/ExternalSwap.sol`
- `bba...ab2 ./src/libraries/TransferHelper.sol`
- `80f...4b9 ./src/libraries/FullMath.sol`
- `7c6...d2e ./src/libraries/SafeCast.sol`
- `691...bca ./src/libraries/ReentrancyGuardTransient.sol`
- `6b2...49c ./src/libraries/BytesLib.sol`
- `da8...32a ./src/test/WETH9.sol`
- `32d...b36 ./src/test/MockERC20.sol`

Test Suite Results

All tests are passing.

```
Ran 24 tests for test/integration/LPToken.t.sol:LPTokenTest
[PASS] test_ComplexDepositRedeemScenario() (gas: 827515)
[PASS] test_SetEarlyWithdrawFeeBips() (gas: 65259)
[PASS] test_deposit_belowMinDeposit() (gas: 46800)
[PASS] test_deposit_insufficientAllowance() (gas: 46766)
[PASS] test_deposit_insufficientBalance() (gas: 73961)
[PASS] test_deposit_multipleUsers() (gas: 278602)
[PASS] test_deposit_success() (gas: 222245)
[PASS] test_deposit_whenPaused() (gas: 44709)
[PASS] test_deposit_zeroAmount() (gas: 21550)
[PASS] test_distributeYield_multipleDistributions() (gas: 230920)
[PASS] test_distributeYield_onlyVault() (gas: 230083)
[PASS] test_distributeYield_success() (gas: 230167)
[PASS] test_distributeYield_whenPoolNotInitialized() (gas: 51112)
[PASS] test_distributeYield_zeroAmount() (gas: 184770)
[PASS] test_pauseAndUnpause() (gas: 214640)
[PASS] test_redeem_afterYieldDistribution() (gas: 509026)
[PASS] test_redeem_earlyWithdrawalFee() (gas: 228729)
```

```
[PASS] test_redeem_insufficientShares() (gas: 21843)
[PASS] test_redeem_partialAmount() (gas: 226980)
[PASS] test_redeem_success() (gas: 206231)
[PASS] test_redeem_whenPaused() (gas: 194169)
[PASS] test_redeem_zeroAmount() (gas: 19193)
[PASS] test_setMinDeposit() (gas: 53437)
[PASS] test_setMinRedeemInterval() (gas: 48501)
Suite result: ok. 24 passed; 0 failed; 0 skipped; finished in 366.74ms (101.91ms CPU time)

Ran 1 test for test/integration/NativeRFQPool.sol:NativeRFQPoolTest
[PASS] test_multipleRoutersAndPools() (gas: 18366874)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 371.54ms (78.32ms CPU time)

Ran 20 tests for test/integration/LPTokenERC20.t.sol:LPTokenERC20Test
[PASS] test_approve_and_allowance() (gas: 71014)
[PASS] test_approve_emitsEvent() (gas: 46367)
[PASS] test_approve_multipleApprovals() (gas: 87655)
[PASS] test_approve_zeroAddress() (gas: 16417)
[PASS] test_transferFrom() (gas: 318367)
[PASS] test_transferFrom_afterApprovalRevoked() (gas: 253268)
[PASS] test_transferFrom_insufficientAllowance() (gas: 248499)
[PASS] test_transferFrom_insufficientBalance() (gas: 253005)
[PASS] test_transferFrom_noApproval() (gas: 226932)
[PASS] test_transferFrom_toSelf() (gas: 251120)
[PASS] test_transferFrom_zeroAddress() (gas: 251110)
[PASS] test_transferFrom_zeroAmount() (gas: 261106)
[PASS] test_transfer_emitsEvents() (gas: 69174)
[PASS] test_transfer_insufficientBalance() (gas: 26494)
[PASS] test_transfer_pausedState() (gas: 71851)
[PASS] test_transfer_success() (gas: 108700)
[PASS] test_transfer_toSelf() (gas: 22388)
[PASS] test_transfer_withComplexExchangeRate() (gas: 3924061)
[PASS] test_transfer_zeroAddress() (gas: 22099)
[PASS] test_transfer_zeroAmount() (gas: 36734)
Suite result: ok. 20 passed; 0 failed; 0 skipped; finished in 377.52ms (71.71ms CPU time)

Ran 25 tests for test/integration/CreditVault.t.sol:CreditVaultTest
[PASS] test_addCollateral_addForOthers() (gas: 556552)
[PASS] test_addCollateral_multipleTokens() (gas: 934232)
[PASS] test_addCollateral_notLpToken() (gas: 325325)
[PASS] test_addCollateral_success() (gas: 508601)
[PASS] test_epochUpdate_cooldownPeriod() (gas: 632331)
[PASS] test_epochUpdate_exchangeRateLimit() (gas: 506883)
[PASS] test_epochUpdate_feeAccrual() (gas: 610893)
[PASS] test_epochUpdate_lpPoolNotInitialized() (gas: 50582)
[PASS] test_epochUpdate_onlyEpochUpdater() (gas: 588328)
[PASS] test_liquidate_notLiquidator() (gas: 552852)
[PASS] test_liquidate_success() (gas: 1637494)
[PASS] test_removeCollateral() (gas: 772959)
[PASS] test_repay_success() (gas: 469871)
[PASS] test_setAllowance() (gas: 3287951)
[PASS] test_setEpochUpdater() (gas: 71863)
[PASS] test_setFeeWithdrawer() (gas: 72751)
[PASS] test_setLiquidator() (gas: 85964)
[PASS] test_setNativePool() (gas: 3103791)
[PASS] test_setSigner() (gas: 70920)
[PASS] test_setTrader() (gas: 146867)
[PASS] test_settle_success() (gas: 1162043)
[PASS] test_supportMarket() (gas: 7176488)
[PASS] test_swapCallback() (gas: 3173307)
[PASS] test_transferOut() (gas: 348362)
[PASS] test_withdrawFundingFee() (gas: 649602)
Suite result: ok. 25 passed; 0 failed; 0 skipped; finished in 1.93s (124.11ms CPU time)

Ran 6 tests for test/fuzz/LPToken.fuzz.t.sol:LPTokenFuzzTest
[PASS] testFuzz_deposit(uint256) (runs: 1024, μ: 211583, ~: 211640)
[PASS] testFuzz_depositAndEarlyRedeem(uint256,uint256,uint256) (runs: 1024, μ: 258991, ~: 259354)
[PASS] testFuzz_depositAndNormalRedeem(uint256,uint256,uint256) (runs: 1024, μ: 234678, ~: 235000)
[PASS] testFuzz_depositReverts(uint256) (runs: 1024, μ: 20339, ~: 20339)
[PASS] testFuzz_redeem(uint256,uint256) (runs: 1024, μ: 256843, ~: 256898)
[PASS] testFuzz_redeemReverts(uint256) (runs: 1024, μ: 18797, ~: 18797)
Suite result: ok. 6 passed; 0 failed; 0 skipped; finished in 2.96s (6.38s CPU time)
```

```
Ran 4 tests for test/integration/NativeRouter.t.sol:NativeRouterTest
[PASS] test_tradeAutoSign() (gas: 18106466)
[PASS] test_tradeAutoSign_revertScenarios() (gas: 17993734)
[PASS] test_tradeRFQWithCredit() (gas: 18124965)
[PASS] test_tradeRFQWithPMMInventory() (gas: 18105974)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 19.24s (285.36ms CPU time)

Ran 5 tests for test/invariant/LPToken.invariants.t.sol:LPTokenInvariants
[PASS] invariant_depositWithdrawalAccounting() (runs: 512, calls: 32768, reverts: 16359)
[PASS] invariant_sumOfSharesEqualsTotalShares() (runs: 512, calls: 32768, reverts: 16322)
[PASS] invariant_totalSupplyMatchesUnderlying() (runs: 512, calls: 32768, reverts: 16466)
[PASS] invariant_underlyingBalance() (runs: 512, calls: 32768, reverts: 16311)
[PASS] invariant_underlyingMatchesNetDeposits() (runs: 512, calls: 32768, reverts: 16221)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 22.09s (97.36s CPU time)

Ran 7 test suites in 22.14s (47.33s CPU time): 85 tests passed, 0 failed, 0 skipped (85 total tests)
```

Code Coverage

Tests achieve 75-95% coverage for the `CreditVault.sol` and `LPToken.sol` contracts. Test coverages for `NativeRFQPool.sol` and `NativeRouter.sol` contracts are lower at ~40-70%.

```
╭-----------------------------------------+-----------------+-------------------+----------------+--
-------------╮
| File                                    | % Lines         | % Statements      | % Branches     | %
Funcs         |
+=============================================================================================================
==============+
| script/Deploy_ERC20.s.sol               | 0.00% (0/10)    | 0.00% (0/11)      | 100.00% (0/0)  |
0.00% (0/1)     |
|-----------------------------------------+-----------------+-------------------+----------------+--
-------------|
| script/Deploy_PMM_Pool.s.sol            | 0.00% (0/15)    | 0.00% (0/21)      | 100.00% (0/0)  |
0.00% (0/1)     |
|-----------------------------------------+-----------------+-------------------+----------------+--
-------------|
| script/Deploy_USDC.s.sol                | 0.00% (0/7)     | 0.00% (0/8)       | 100.00% (0/0)  |
0.00% (0/1)     |
|-----------------------------------------+-----------------+-------------------+----------------+--
-------------|
| script/Deploy_WETH9.s.sol               | 0.00% (0/7)     | 0.00% (0/8)       | 100.00% (0/0)  |
0.00% (0/1)     |
|-----------------------------------------+-----------------+-------------------+----------------+--
-------------|
| script/Deploy_core.s.sol                | 0.00% (0/27)    | 0.00% (0/31)      | 100.00% (0/0)  |
0.00% (0/1)     |
|-----------------------------------------+-----------------+-------------------+----------------+--
-------------|
| script/Deploy_lp.s.sol                  | 0.00% (0/14)    | 0.00% (0/20)      | 100.00% (0/0)  |
0.00% (0/1)     |
|-----------------------------------------+-----------------+-------------------+----------------+--
-------------|
| script/Utils.sol                        | 0.00% (0/10)    | 0.00% (0/13)      | 100.00% (0/0)  |
0.00% (0/3)     |
|-----------------------------------------+-----------------+-------------------+----------------+--
-------------|
| src/CreditVault.sol                     | 95.24% (160/168) | 95.45% (189/198) | 76.47% (26/34) |
100.00% (24/24) |
|-----------------------------------------+-----------------+-------------------+----------------+--
-------------|
| src/LPToken.sol                         | 95.51% (85/89)  | 94.74% (90/95)    | 88.89% (16/18) |
95.00% (19/20)  |
|-----------------------------------------+-----------------+-------------------+----------------+--
-------------|
| src/NativeRFQPool.sol                   | 69.09% (38/55)  | 75.93% (41/54)    | 10.00% (1/10)  |
81.82% (9/11)   |
|-----------------------------------------+-----------------+-------------------+----------------+--
```

```
---------------|
| src/NativeRouter.sol                        | 48.96% (47/96)   | 49.04% (51/104)   | 18.52% (5/27)   |
64.29% (9/14)   |
|---------------------------------------------+------------------+-------------------+-----------------+--
---------------|
| src/libraries/BytesLib.sol                  | 0.00% (0/35)     | 0.00% (0/35)      | 0.00% (0/10)    |
0.00% (0/4)     |
|---------------------------------------------+------------------+-------------------+-----------------+--
---------------|
| src/libraries/ExternalSwap.sol              | 0.00% (0/35)     | 0.00% (0/44)      | 0.00% (0/10)    |
0.00% (0/2)     |
|---------------------------------------------+------------------+-------------------+-----------------+--
---------------|
| src/libraries/FullMath.sol                  | 0.00% (0/34)     | 0.00% (0/35)      | 0.00% (0/8)     |
0.00% (0/2)     |
|---------------------------------------------+------------------+-------------------+-----------------+--
---------------|
| src/libraries/Multicall.sol                 | 0.00% (0/13)     | 0.00% (0/15)      | 0.00% (0/4)     |
0.00% (0/2)     |
|---------------------------------------------+------------------+-------------------+-----------------+--
---------------|
| src/libraries/Order.sol                     | 0.00% (0/23)     | 0.00% (0/21)      | 0.00% (0/4)     |
0.00% (0/5)     |
|---------------------------------------------+------------------+-------------------+-----------------+--
---------------|
| src/libraries/ReentrancyGuardTransient.sol | 72.73% (8/11)    | 62.50% (5/8)      | 0.00% (0/1)     |
100.00% (4/4)   |
|---------------------------------------------+------------------+-------------------+-----------------+--
---------------|
| src/libraries/SafeCast.sol                  | 0.00% (0/7)      | 0.00% (0/4)       | 0.00% (0/6)     |
0.00% (0/3)     |
|---------------------------------------------+------------------+-------------------+-----------------+--
---------------|
| src/libraries/TStorage.sol                  | 50.00% (2/4)     | 0.00% (0/2)       | 100.00% (0/0)   |
100.00% (2/2)   |
|---------------------------------------------+------------------+-------------------+-----------------+--
---------------|
| src/libraries/TransferHelper.sol            | 18.18% (2/11)    | 14.29% (1/7)      | 0.00% (0/2)     |
20.00% (1/5)    |
|---------------------------------------------+------------------+-------------------+-----------------+--
---------------|
| src/test/MockERC20.sol                      | 72.73% (8/11)    | 71.43% (5/7)      | 0.00% (0/4)     |
75.00% (3/4)    |
|---------------------------------------------+------------------+-------------------+-----------------+--
---------------|
| src/test/WETH9.sol                          | 46.43% (13/28)   | 52.00% (13/25)    | 11.11% (1/9)    |
28.57% (2/7)    |
|---------------------------------------------+------------------+-------------------+-----------------+--
---------------|
| test/BaseTest.t.sol                         | 68.45% (115/168) | 65.62% (126/192)  | 0.00% (0/8)     |
55.56% (5/9)    |
|---------------------------------------------+------------------+-------------------+-----------------+--
---------------|
| test/helpers/SignatureUtils.sol             | 100.00% (32/32)  | 100.00% (38/38)   | 100.00% (0/0)   |
100.00% (8/8)   |
|---------------------------------------------+------------------+-------------------+-----------------+--
---------------|
| test/invariant/handlers/LPTokenHandler.sol | 86.15% (56/65)   | 87.88% (58/66)    | 0.00% (0/2)     |
80.00% (4/5)    |
|---------------------------------------------+------------------+-------------------+-----------------+--
---------------|
| test/invariant/helpers/AddressSet.sol       | 54.55% (6/11)    | 57.14% (4/7)      | 33.33% (1/3)    |
50.00% (2/4)    |
|---------------------------------------------+------------------+-------------------+-----------------+--
---------------|
| Total                                       | 58.01% (572/986) | 58.09% (621/1069) | 31.25% (50/160) |
63.89% (92/144) |
╰---------------------------------------------+------------------+-------------------+-----------------+--
---------------╯
```

Changelog

- 2024-12-20 - Initial Report
- 2025-01-01 - Updated Report 1
- 2025-01-03 - Updated Report 2
- 2025-01-06 - Final Report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

monitoring any any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.