Quantstamp

Hashflow Protocol 3

Executive Summary

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

Type	MEV-Resistant DEX
Timeline	2023-05-22 through 2023-06-16
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Source Code	• hashflownetwork/protocol #4b27a46 • hashflownetwork/protocol #a8ff219
Auditors	• Rabib Islam Auditing Engineer • Ibrahim Abouzied Auditing Engineer • Shih-Hung Wang Auditing Engineer • Guillermo Escobero Auditing Engineer

Documentation quality	Medium
Test quality	Low
Total Findings	23 Fixed: 11 Acknowledged: 12
High severity findings	1 Fixed: 1
Medium severity findings	7 Fixed: 3 Acknowledged: 4
Low severity findings	8 Fixed: 4 Acknowledged: 4
Undetermined severity findings	1 Acknowledged: 1
Informational findings	6 Fixed: 3 Acknowledged: 3

Summary of Findings

Update: The client has addressed all the issues found during the audit, obtaining a status of "Fixed" or "Acknowledged" for each one. However, some of the fixes introduced new best practices issues, and the update to HASH-6 should be reviewed in order to fortify the current implementation.

On the other hand, test coverage has decreased, which is very concerning. We highly recommend improving the test suite, especially for core contracts like `HashflowRouter` and `HashflowPool` that implement critical functionality.

Initial audit: Quantstamp performed an audit on Hashflow's protocol contracts based on the code present in the linked repository.

The contracts within this repository form the basis for the on-chain behaviour of a decentralized exchange. We have audited previous iterations of this project, but the feature set of the version under consideration in this audit contains some differences:

- public Pools are gone;
- cross-chain message passing functionality has been split into different contracts pertaining to each protocol;
- non-EVM is supported with bytes32 payloads;
- limit orders;
- token approvals for pools + ERC1271 — to work with 1inch Limit Order protocol.

Key points of concern during this audit were as follows:

- Cross-chain functionality;
- Proper use of signatures;
- The prevalence of permissioned roles.

The most critical finding, HASH-1, demonstrates how it may be possible to completely drain pools of their funds using cross-chain trades. We also mention a few other issues involving cross-chain functionality that may cause problems down the line.

While code comments were quite good, additional, higher-level documentation would have been suitable given the nature of the project. Moreover, the test suite is lacking in certain respects. The Hashflow team would ideally shore up both of these aspects of the project.

We recommend that all the issues in this report are addressed.

ID	DESCRIPTION	SEVERITY	STATUS
HASH-1	Market Makers Funds Can Be Drained	High	Fixed
HASH-2	Solidity Version Is Too Recent	Medium	Fixed
HASH-3	Privileged Roles and Ownership	Medium	Acknowledged
HASH-4	Failure to Send Message to Destination Chain Leads to Loss of User Funds on Source Chain	Medium	Acknowledged
HASH-5	Gas Griefing Attack on Relayers	Medium	Acknowledged
HASH-6	DoS Attack on Cross-Chain Trades through LayerZero	Medium	Fixed
HASH-7	Relayer Logic in Case of Underpaid Fees Is Undefined	Medium	Acknowledged
HASH-8	Unable to Provide Fees When Publishing Messages through Wormhole	Medium	Fixed
HASH-9	Missing Input Validation	Low	Fixed
HASH-10	Extensive Reliance on External Applications	Low	Acknowledged
HASH-11	Critical Role Transfer Not Following Two-Step Pattern	Low	Fixed
HASH-12	Unable to Update Pool Implementation For the Factory on zkSync	Low	Fixed
HASH-13	Unable to Set Approval for Specific ERC20 Tokens	Low	Fixed
HASH-14	Front-Running DoS on Trades Using Permit	Low	Acknowledged
HASH-15	Remaining Cross-Chain Fees Are Locked in the Messenger	Low	Acknowledged
HASH-16	Relayer Fees Remain in Messenger Contract	Low	Acknowledged
HASH-17	No Way to Unauthorize a Pool in `HashflowRouter`	Informational	Fixed
HASH-18	LayerZero Best Practices	Informational	Acknowledged
HASH-19	Tokens in `HashflowRouter` Can Be Drained	Informational	Fixed
HASH-20	Application Monitoring Can Be Improved by Emitting More Events	Informational	Fixed
HASH-21	Considerations Regarding Deployment on zkSync	Informational	Acknowledged

ID	DESCRIPTION	SEVERITY	STATUS
HASH-22	Clone-and-Own	Informational	Acknowledged
HASH-23	Messenger Configurations Changes May Fall Out of Sync	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.

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/hashflownetwork/protocol(0184832f7b61b6e86504499fbdd4c2f3143a23ae) Files: contracts/*

Files Excluded

Repo: https://github.com/hashflownetwork/protocol(0184832f7b61b6e86504499fbdd4c2f3143a23ae) Files: contracts/testing/*

Findings

HASH-1 Market Makers Funds Can Be Drained

High Fixed

Update

Marked as "Fixed" by the client. Addressed in: `ae39d3f29fccc2d6befe125938255fbd6f1af610`. The client provided the following explanation:

> This is needed in order to create x-Apps that talk to each other and do things upon transfer. The motivator for this was the implementation of AAVE Portals, where the x-Apps will:
> - redeem a user's aToken for Token
> - execute a x-chain swap of Token for TargetToken
> - deposit TargetToken for aTokens
>
> As a fix, we require callees to register the callers. This way, the protocol contracts that the Router has privilege to (the Pools, the Messengers) would never allow for an x-Call.

File(s) affected: `HashflowRouter.sol`, `HashflowWormholeMessenger.sol`, `HashflowPool.sol`

Description: When performing a cross-chain trade, the trader can pass arbitrary data to be executed at an arbitrary address when the message is received in the destination chain. This call will be executed from the `HashflowRouter` deployed in that chain.

This functionality is blocked if the user (trader) uses LayerZero as transport layer. If the message is sent using Wormhole, the functionality is not blocked, allowing the user to execute arbitrary calldata from the `HashflowRouter` and gaining access to privileged functions (e.g. executing malicious quotes in the Hashflow pools).

Exploit Scenario:

1. Execute the trade calling any of these functions, with `dstContract` set to HashflowPool and `dstCalldata = fillXChain(arbitrary, malicious, parameters)`
   - `tradeXChainRFQM()`
   - `tradeXChainRFQMWithPermit()`
   - `tradeXChainRFQT()`
   - `tradeXChainRFQTWithPermit()`
2. If the message is sent in LayerZero, it will revert as it validates that `dstContract` is zero (in `HashflowLayerZeroMessenger.tradeXChain()`)
3. If sent through Wormhole, the message will be sent successfully.
4. The message is received in `wormholeReceive()` and passed to `router.fillXChain()`
5. Router calls `pool.fillXChain()` and transfers the expected funds to the attacker.
6. Router will execute `dstContract.call(dstCalldata)` - this values were set by the attacker calling `anypool.fillXChain()` (or any authorized function, as `msg.sender` is now router), draining the funds. Can also kill switch the pool, or use old quotes (replay attack).

Recommendation: As the destination chain execution is not permitted when using LayerZero, double check if it is needed in Wormhole case. If not, consider removing this external call.

HASH-2 Solidity Version Is Too Recent

Medium Fixed

Update

Marked as "Fixed" by the client. Addressed in: `224e5c937abd37fb5fef71ac43be8d3ae8be0641`.

Description: All contracts are using `0.8.20` version of the Solidity compiler. As Hashflow interacts and has contracts deployed in Arbitrum, `0.8.20` is not recommended for these contracts.

Arbitrum is not fully compatible with `0.8.20`:

> OPCODE push0 is not yet supported, but will soon be available. This means that solidity version 0.8.20 or higher can only be used with an evm-version lower than the default shanghai (see instructions here to change that parameter in solc, or here to set the solidity or evmVersion configuration parameters in hardhat). Versions up to 0.8.19 (included) are fully compatible.

Reference

Recommendation: To avoid unexpected behavior or bugs, use the recommended `pragma solidity 0.8.18`. Although it may be feasible to use `0.8.19`, we suggest using `0.8.18` as using the older of the two may help avoid unknown bugs introduced by feature additions.

HASH-3 Privileged Roles and Ownership

Medium | Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> Acknowledged. Furthermore, two features have been removed in future commits:
> - multi-hop trades
> - the ability to change the implementation once it's been set These changes limit the risks of the owner of HashflowRouter and HashflowFactory being compromised.

Description: The Hashflow protocol makes extensive use of privileged roles. A list of the actions accorded to each role follows:

- `HashflowRouter`
  - `owner`
    - `initialize()`
    - `killswitchPool()` -- disables pool actions
    - `withdrawFunds()` -- withdraws native tokens (e.g. Ether) and ERC20 tokens from the router
    - `updateLimitOrderGuardian()` -- sets the guardian role
  - `limitOrderGuardian`
    - This role, operated by Hashflow, signs limit order quotes in order to match market makers to traders, preventing MEV. However, users must rely on this role being operated fairly.
- `HashflowPool`
  - `router`
    - `tradeRFQT()`
    - `tradeRFQM()`
    - `tradeXChainRFQT()`
    - `fillXChain()`
    - `tradeXChainRFQM()`
    - `killSwitchOperations()`
  - `operations`
    - `updateXChainPoolAuthorization()`
    - `updateXChainMessengerAuthorization()`
    - `approveToken()`
    - `increaseTokenAllowance()`
    - `decreaseTokenAllowance()`
    - `removeLiquidity()`
    - `updateWithdrawalAccount()`
    - `updateSigner()`
- `HashflowFactory`
  - `owner`
    - `initialize()`
    - `updatePoolCreatorAuthorization()`
    - `updatePoolImpl()`
  - `allowedPoolCreators`
    - `createPool()`
- `HashflowXChainMessengerBase`
  - `owner`
    - `updateXChainRemoteAddress()`
    - `withdrawFunds()`
- `HashflowWormholeMessenger`
  - `owner`
    - `updateWormhole()`
    - `updateWormholeConsistencyLevel()`
    - `updateWormholeConsistencyLevelFast()`
    - `updatePermissionedRelayer`
    - `updateWormholeChainIdForHashflowChainId`
- `HashflowLayerZeroMessenger`
  - `owner`
    - `updateLzEndpoint()`
    - `updateLzGasEstimate()`
    - `updateLzChainIdForHashflowChainId()`
    - `setSendVersion()`
    - `setReceiveVersion()`
    - `setConfig()`
    - `forceResumeReceive()`

Exploit Scenario: A malicious or compromised owner of `HashflowFactory` can:

1. Update the pool implementation to a malicious one and affect the future deployed pools.
2. Set themselves as an allowed pool creator and deploy malicious pools after updating the implementation.

A malicious or compromised owner of `HashflowRouter` can:

1. Stop a specific pool, which is the simplest way to DoS the protocol.
2. Withdraw funds from the router and receive all funds left in the router after a user calls the `tradeRFQTMultiHop()` function.
3. Update the limit order guardian to themselves.

A malicious or compromised `limitOrderGuardian` of `HashflowRouter` can unfairly prioritize the fulfilling of certain limit orders over others.

A malicious or compromised `operations` of `HashflowPool` can:

1. Approve themselves to transfer out every ERC20 token in the pool.
2. Remove the entire liquidity from the pool to themselves.
3. Update the withdrawal account so that they can remove the pool liquidity to any address.
4. Update the signer to themselves to accept arbitrary trades generated from them and trades signed by the original signer will be rejected.
5. Update the cross-chain pool authorization information to allow trading with a potentially malicious pool on the remote chain.
6. Update the cross-chain messenger to disrupt or DoS the cross-chain messaging process and steal the cross-chain fees (if any).

A malicious or compromised `signerConfiguration.signer` of a pool can:

1. Create arbitrary trades regardless of the market price and execute them using another account, effectively stealing all funds from the pool.

A malicious or compromised owner of `HashflowXChainMessengerBase` can:

1. Withdraw all native tokens from the contract to themselves.
2. Update the chain remote address to potentially allow interaction with a malicious contract on the remote chain.

A malicious or compromised owner of `HashflowLayerZeroMessenger` can:

1. Update the LayerZero endpoint to a malicious contract and send messages to the messenger from the malicious contract.
2. Update the chain ID mapping from a LayerZero ID to Hashflow ID and vice versa.
3. Update the LayerZero gas estimation.
4. Update the LayerZero send and receive version and configuration.

A malicious or compromised owner of `HashflowWormholeMessenger` can:

1. Update the Wormhole endpoint to a malicious contract and send messages to the messenger from the malicious contract.
2. Update the chain ID mapping from a Wormhole ID to Hashflow ID and vice versa.
3. Update the wormhole consistency level for the current chain to affect the cross-chain messaging process.
4. Update the permissioned relayer to allow an address to become a relayer of some chain.

Recommendation: This centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the owner.

HASH-4

Failure to Send Message to Destination Chain Leads to Loss of User Funds on Source Chain

Medium | Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> Acknowledged. Additionally, all Market Makers on Hashflow undergo a KYC process. This means that Law Enforcement can be engaged in the rare eventuality that funds are not made available. Furthermore, the Hashflow API implementation checks that enough tokens have been deposited at the time of RFQ.

Description: When a cross-chain swap is executed, first, the trader's funds are sent from the trader to the market maker on the destination chain. Then, a message is sent to the destination chain via LayerZero or Wormhole so that the corresponding funds can be sent to the trader from the market maker. However, if the message fails to execute on the destination chain, then the trader will effectively be short funds on the source chain.

In the event of a message failing to be passed via LayerZero, the owner of the messenger contract must call `forceResumeReceive()` in order to resume activity; this will cause LayerZero to skip over the failing trade.

Recommendation: Alert users to the risk of this issue via end-user-facing documentation. Additionally, provide a means of returning funds to traders on the source chain in the event of a cross-chain trade failing.

HASH-5 Gas Griefing Attack on Relayers

Medium | Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
> > Acknowledged. A few measures are going to be taken:
> > - Relayers will run gas estimation prior to issuing a relay
> > - The RFQ API will check for known smart contract wallets (e.g. Argent) in order to reduce interactions with arbitrary contracts
> > - For API trading, keys are issued based on request forms and they can be revoked if malicious behavior is observed

File(s) affected: `HashflowRouter.sol`

Description: When a relayer executes an RFQ-M trade, the `_validateRFQMSignature()` function allows the trader to validate the taker's signature. If the trader is a smart contract, the router calls the `isValidSignature()` function on the contract, which then decides whether the signature is valid by returning a specific magic value.

However, a malicious contract can exploit this validation process by rejecting the signature and causing the transaction to fail. The attacker can even consume a large amount of gas before rejecting the signature. Moreover, the attacker can send many requests for relaying RFQ-M trades with the malicious contract as the trader and make the relayer pay gas fees whenever they try to relay the transactions.

Recommendation: A possible mitigation is to simulate the RFQ-M trades and ensure the transaction succeeds before broadcasting. However, it should be noted it could be possible for the malicious contract to detect whether the transaction is under a local simulation and return different results accordingly.

HASH-6
DoS Attack on Cross-Chain Trades through LayerZero

Medium | Fixed

> **Update**
> The issue has been fixed; however, in the `retryPayload()` function, the failed message is cleared to zero after `_lzReceive()`. It would be better if it is cleared before `_lzReceive()` to prevent potential reentrancy issues. A different solution would be to add the `nonReentrant` modifier, which would be useful for all functions that invoke untrusted external calls.
>
> Marked as "Fixed" by the client. Addressed in: `0c8f5d79ba5036320d9f02277cd43e1170ad7a14`.

File(s) affected: `HashflowLayerZeroMessenger.sol`

Description: When receiving a cross-chain message from LayerZero, the LayerZero Endpoint calls the `lzReceive()` function of `HashflowLayerZeroMessenger`. Suppose the `lzReceive()` function call fails. In that case, subsequent messages from the same source chain and address are blocked until the messenger on the destination chain calls `forceResumeReceive()`, a permissioned function only the owner can execute.

The `lzReceive()` function calls `fillXChain()` on the router, and the router then calls `fillXChain()` on the pool specified in the quote. During the pool's `fillXChain()` call, funds are transferred either from the pool or an external account based on whether the `externalAccount` field is set.

However, a malicious receiver can intentionally revert the transaction when the pool tries to send funds to them. For example,

1. If the `quoteToken` is the native token of the destination chain, the receiver, as a contract, can have a reverting `receive()` function to revert the entire transaction.
2. If the `quoteToken` is a token with transfer hooks (e.g., ERC777), the receiver can register a reverting receive hook to revert the entire transaction as well.

As a result, the malicious receiver blocks the message delivery from a specific source pool to the destination pool. The pool owner has to call `forceResumeReceive()` whenever the attacker launches such an attack.

Recommendation: Consider implementing a non-reverting `lzReceive()` function by catching the exceptions raised during the function call. Store the failed messages in the contract and allow the owner to retry them afterward.

HASH-7
Relayer Logic in Case of Underpaid Fees Is Undefined

Medium | Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
> > Acknowledged. Hashflow relays are permissionless and the app will allow for self relay.

File(s) affected: `HashflowXChainMessengerBase.sol`, `HashflowWormholeMessenger.sol`, `HashflowLayerZeroMessenger.sol`

Description: Relayers that facilitate cross-chain transactions typically require being paid fees for their work. These fees are sent via `IHashflowXChainMessenger.tradeXChain()`. However, there are no restrictions enforced on-chain concerning amount to be paid in terms of these fees, save for that in the case of LayerZero, they must also cover the cost of paying the amount returned by `ILayerZeroEndpoint(lzEndpoint).estimateFees()`. Therefore, it is unclear what is to be done in the event that a relayer is underpaid for a given quote.

It is also unclear whether relayer fees will always be able to cover the cost of gas on the destination chain, as gas tokens on different chains are often priced differently.

Recommendation: Clarify in end-user-facing documentation what each party (trader, relayer, market maker) is to do in the case that a relayer is underpaid for the facilitation of a cross-chain trade.

HASH-8
Unable to Provide Fees When Publishing Messages through Wormhole

Medium | Fixed

> **Update**
> Marked as "Fixed" by the client. Addressed in: `3797fb6d6f89241d66e54d756513d3117bc1d22f`.

Description: According to the Wormhole Development Book,

> Currently there are no fees to publish a message (with the exception of publishing on Solana) but this is not guaranteed to always be the case in the future.

However, in the current implementation, the Wormhole messenger is unable to provide native tokens when calling the `publishMessage()` function. If, in the future, Wormhole charges the publish fee, the messenger will fail to deliver cross-chain messages.

Recommendation: Consider fetching the required publish fee from the Wormhole endpoint with the `messageFee()` function and forwarding the exact number of native tokens when calling `publishMessage()`.

HASH-9 Missing Input Validation

Low | Fixed

> **Update**
> Marked as "Fixed" by the client. Addressed in: `dde4782513a03db9f0cc20520d35540d97ee11cc`.

File(s) affected: `HashflowRouter.sol`, `HashflowPool.sol`, `HashflowFactoryZkSync.sol`

Related Issue(s): SWC-123

Description: It is important to validate inputs, even if they only come from trusted addresses, to avoid human error. Specifically, in the following functions arguments of type `address` may be initialized with value `0x0`:

1. `HashflowRouter.initialize()`
   - validate that `_factory` is not the zero address.
2. `HashflowPool.constructor()`
   - validate that `weth` is not the zero address.
3. `HashflowPool.initialize()`
   - validate that `_name` is not an empty string.
4. `HashflowFactoryZkSync.constructor()`

- validate that `_WETH` is not the zero address.

5. `HashflowFactory.updatePoolCreatorAuthorization()`
   - validate that `poolCreator` is not the zero address.
6. `HashflowFactory._createPoolInternal()`
   - validate that `name` is not an empty string.

Recommendation: We recommend adding the relevant checks.

HASH-10 Extensive Reliance on External Applications

Low Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
> > The risks are documented / going to be documented clearly. Currently, the protocol makes pools opt into protocol. This decision is critical to security, as we want to ensure that one interop protocol facing a breach does not drain every Market Maker pool.

Description: The Hashflow protocol integrates LayerZero and Wormhole, cross-chain message-passing applications, in order to facilitate cross-chain trades. However, these applications come with their own set of risks. Perhaps the most prescient risk is downtime, which may impact the ability of the protocol to fill quotes.

Recommendation: Make sure users are well-apprised of the risks associated with using LayerZero and Wormhole. Moreover, consider allowing users to use a greater variety of cross-chain message-passing applications in the future for the sake of user freedom and diversification as a means of ensuring uptime.

HASH-11
Critical Role Transfer Not Following Two-Step Pattern

Low Fixed

> **Update**
> Marked as "Fixed" by the client. Addressed in: `c9f3071609184fd19edfbd73bedb4100c8050a2c`.

File(s) affected: `HashflowRouter.sol`, `HashflowFactory.sol`, `HashflowXChainMessengerBase.sol`

Description: The `owner` of the protocol contracts can call `transferOwnership()` to transfer the ownership to a new address. Suppose the new owner's address is accidentally provided as an uncontrollable address. In that case, the contract will lose the owner, and no one will be able to execute any function with the `onlyOwner` modifier anymore.

Recommendation: Consider using OpenZeppelin's `Ownable2Step` contract to adopt a two-step ownership pattern to prevent this issue.

HASH-12
Unable to Update Pool Implementation For the Factory on zkSync

Low Fixed

> **Update**
> Marked as "Fixed" by the client. Addressed in: `dde4782513a03db9f0cc20520d35540d97ee11cc`. The client provided the following explanation:
>
> > The protocol utilizes Clones in order to reduce gas costs when creating pools. There is no need to change the Pool implementation once it's been set. Further changes can be implemented via re-deployment of the protocol.

Description: The `HashflowFactoryZkSync` contract inherits from the `HashflowFactory` contract and therefore inherits the `updatePoolImpl()` function. However, since the factory deployed on zkSync uses `new HashflowPool()` to create a pool contract, the bytecode is determined at compile time and cannot change in the future.

As a result, this function does not effectively updates the pool implementation as expected. Protocol owners using the `HashflowFactoryZkSync` factory contract should be aware of this limitation and redeploy the factory contract whether the `HashflowPool` contract needs to be updated.

Recommendation: Consider clarifying the limitation of factory contracts on zkSync in technical documentation or code comments and overriding the `updatePoolImpl()` to avoid unintentional manual errors.

HASH-13 Unable to Set Approval for Specific ERC20 Tokens

Low Fixed

> **Update**
> Marked as "Fixed" by the client. Addressed in: `c3c978f2c1d8c342f1c9876fa9c88a74047d6dae`.

File(s) affected: `HashflowPool.sol`

Description: Some ERC20 tokens, e.g., `USDT` on Ethereum, do not return a boolean from the `approve()` function call. Therefore, calling `IERC20(token).approve()` on these tokens causes the transaction to revert when decoding the return data.

In addition, some ERC20 tokens have a built-in approval race protection that disallows approval of a non-zero amount when there is already a non-zero allowance. For example, `USDT` requires the allowance to be set to 0 before setting it to a non-zero value.

Recommendation:

1. Consider using the `safeIncreaseAllowance()` and `safeDecreaseAllowance()` functions defined in `SafeERC20` for the `increaseTokenAllowance()` and `decreaseTokenAllowance()` functions.
2. For the `approveToken()` function, consider using the `forceApprove()` function. However, such usage is discouraged since it is vulnerable to the ERC20 double-approve issue.

HASH-14 Front-Running DoS on Trades Using Permit

Low Acknowledged

> **Update**
> Marked as "Acknowledged" by the client.

Description: An attacker can front-run the trades using `permit()`, e.g., `tradeRFQTWithPermit()`, to cause DoS. Before the trade executes, the attacker calls the `permit()` function of the base token with the same signature to increase the trader's permit nonce by one. As a result, when the trade is executed, the transaction reverts since the permit signature is not valid anymore.

This attack is feasible on chains with a public mempool, e.g., Ethereum, BSC. However, the impact of such an attack may be low as users may retry executing the trade using the functions without `permit()`, e.g., `tradeRFQT()`.

Recommendation: Consider clarifying the risk of the front-running DoS issue in public-facing documentation.

HASH-15 Remaining Cross-Chain Fees Are Locked in the Messenger

Low Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
> > In the case of Wormhole, it is hard to tell whether fees are in excess, as the protocol does not leverage oracles. In the case of LayerZero it is possible, however we elect to capture the surplus in the contract and have it serve as insurance for cases where relayers are underpaid.

File(s) affected: `HashflowLayerZeroMessenger.sol`

Description: The `HashflowLayerZeroMessenger` contract estimates the fees for messaging through LayerZero by the `estimateFees()` function and ensures the provided `msg.value` is greater or equal to the estimated fees.

Such fees for Wormhole are not being calculated at the moment, so the entirety of `msg.value` is held by the contract.

In the case of LayerZero, if the provided `msg.value` exceeds the platform's fees, only the exact estimated fee is forwarded to the LayerZero endpoint. The remaining native tokens are left in the messenger and can only be withdrawn by the owner.

As for Wormhole, the entirety of `msg.value` is retained in the messenger smart contract.

A path is not currently implemented for the case where the remaining fees after paying LayerZero/Wormhole is in excess of the expected amount.

Recommendation: Consider transferring the excess quantity of native tokens to the caller at the end of the `_lzSend()` and `_wormholeSend()` functions.

HASH-16 Relayer Fees Remain in Messenger Contract

Low Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
> > Acknowledged. The accounting for fee collection by the relayer is going to happen off-chain.

File(s) affected: `HashflowRouter.sol`, `HashflowXChainMessengerBase.sol`, `HashflowLayerZeroMessenger.sol`, `HashflowWormholeMessenger.sol`

Description: Relayers that facilitate cross-chain transactions typically require being paid fees for their work. These fees are sent via `IHashflowXChainMessenger.tradeXChain()`. However, upon inspecting both `HashflowLayerZeroMessenger` and `HashflowWormholeMessenger`, it appears that fees are sent to the messenger contracts, and in the case of `HashflowLayerZeroMessenger`, an amount is subtracted to pay LayerZero. Only a messenger contract's owner can retrieve the fees paid. It is as of yet unclear whether the messenger contract owners operate the sole relayers available.

Recommendation: Consider the following two courses of action:

1. Create a way for the relayer to receive their fees directly;
2. Clearly document the available relayers and how they will be reimbursed for their work.

HASH-17 No Way to Unauthorize a Pool in `HashflowRouter`

Informational Fixed

> **Update**
> Marked as "Fixed" by the client. Addressed in: `2cef0bb8e9af790d2d9a1e16efe15a8064a3c6ed`.

File(s) affected: `HashflowRouter.sol`

Description: The `HashflowRouter` contract tracks which pools are authorized for trading using the storage mapping `mapping(address => bool) public authorizedPools`. Pools within the mapping can be authorized by calling `HashflowRouter.updatePoolAuthorization()` that is only callable from `HashflowFactory.createPool()` when a pool is created. This function will set the new pool as authorized in the `HashflowRouter`.

However, there is no way to unauthorize a pool within the `HashflowRouter` contract. In other words, if `authorizedPools[addressX]` is set to `true`, there is no way to change it so that `authorizedPools[addressX]` becomes false. This may be a desirable feature, particularly if a pool is compromised. Short of this feature being available, the `HashflowRouter.killswitchPool()` function must be called.

Recommendation: Add a function for the owner of the `HashflowRouter` contract to unauthorize a pool if it is compromised.

HASH-18 LayerZero Best Practices

Informational Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
> > The current version of the protocol does not use ZRO for fees. In the future, we may want to change that.
> > Regarding adapter params, they are currently passed in order to signal required gas. They are not hardcoded to 0.

File(s) affected: `HashflowLayerZeroMessenger.sol`

Description: LayerZero provides a integration checklist with common pitfalls and recommendations for projects here. Points from this checklist should be taken into consideration for `HashflowLayerZeroMessenger`.

Recommendation:

1. Do not hardcode address zero (`address(0)`) as `zroPaymentAddress` when estimating fees and sending messages. Pass it as a parameter instead.
2. Do not hardcode `useZro` to `false` when estimating fees and sending messages. Pass it as a parameter instead.

HASH-19 Tokens in `HashflowRouter` Can Be Drained

Informational Fixed

Update

Marked as "Fixed" by the client. Addressed in: `c9f3071609184fd19edfbd73bedb4100c8050a2c` . The client provided the following explanation:

> The tradeMultihop method has been removed. This can be implemented in a separate contract.

Description: The `HashflowRouter.tradeMultiHop()` function can drain all tokens from the `HashflowRouter` contract. The function allows users to swap from one token to another through multiple paths. It takes a `baseToken` and a `quoteToken` parameter representing the token to swap from and the token to swap to, respectively. However, it is also possible to instruct the `HashflowRouter.tradeMultiHop()` function to swap from a token that is not the `baseToken` to the `quoteToken` . Any token which is not the `baseToken` is taken from the `HashflowRouter` contract itself, as seen below:

```
if (quotes[i].quoteToken == quoteToken) {
    require(
        quotes[i].trader == msg.sender,
        'HashflowRouter: Trader address for must be sender.'
    );
} else {
    require(
        quotes[i].trader == address(this),
        'HashflowRouter: Trader address for must be router.'
    );
}
```

An attacker can provide an `RFQTQuote` to the `tradeMultiHop()` contract, which swaps from a token held by the `HashflowRouter` contract to the `quoteToken` . Then the tokens outputted by the swap will be sent to the attacker.

Exploit Scenario:

1. The `HashflowRouter` contract currently holds 1 WETH token
2. The attacker calls the `HashflowRouter.tradeMultiHop()` function with the following parameters:
    - `quotes` contains one `RFQTQuote` whereby:
        - `RFQTQuote.baseToken` is the address of WETH
        - `RFQTQuote.quoteToken` is the address of DAI
    - `baseToken` is the address of USDC
    - `quoteToken` is the address of DAI
3. The attacker's account receives 1 WETH worth of DAI.

Recommendation: We recognize that the `HashflowRouter` contract is not meant to hold tokens, and that funds accumulated as a result of rounding issues and accidental transfers to the contract are considered forfeit. We recommend warning users about this through end-user-facing documentation.

HASH-20 Application Monitoring Can Be Improved by Emitting More Events

Informational Fixed

Update

Marked as "Fixed" by the client. Addressed in: `ef3ce9f38ee8087cb503d81ee8bb6af212d3669a` .

File(s) affected: `HashflowLayerZeroMessenger.sol` , `HashflowWormholeMessenger.sol`

Description: In order to validate the proper deployment and initialization of the contracts, it is a good practice to emit events. Also, any important state transitions can be logged, which is beneficial for monitoring the contract, and also tracking eventual bugs or hacks. The cross chain messengers do not always emit events for which quotes are sent and received. This could make it difficult to monitor the protocol.

Recommendation: Add events emitting the `quote.txid` in `HashflowLayerZeroMessenger._lzSend()` , `HashflowLayerZeroMessenger.lzReceive()` , and `HashflowWormholeMessenger.wormholdReceive()` .

HASH-21
Considerations Regarding Deployment on zkSync

Informational | Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> Due to security, we elect to not make contracts upgradeable for now. Instead, the zkSync implementation will undergo extensive testnet tests.

Description: According to the zkSync's official technical documentation, it is recommended using a proxy pattern at the early stage of development as there could be compiler bugs and issues with EVM compatibility discovered in the future.

Recommendation:

1. Consider using a proxy pattern to allow contracts deployed on zkSync to upgrade when necessary. Note that this course of action carries its own risks regarding centralization, and these should be carefully weighed against the risks of deploying the contracts as-is.
2. As the zkSync ecosystem is in an early development stage, consider deploying the contracts on a testnet and thoroughly testing the functionalities before deploying on the zkSync mainnet.

HASH-22 Clone-and-Own

Informational | Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> The LayerZero packages mentioned were not available. In general, the LZ code organization has been volatile, which is why we elected to go with Clone-and-Own. Once source code stabilizes, we will migrate to an imported package.

Description: The clone-and-own approach involves copying and adjusting open source code at one's own discretion. From a development perspective, it is initially beneficial as it reduces the amount of effort. However, from the security perspective, it involves some risks as the code may not follow the best practices, may contain a security vulnerability, or may include intentionally or unintentionally modified upstream libraries.

In the current case, LayerZero smart contract code is being copied to the current repository.

Recommendation: Rather than the clone-and-own approach, a good industry practice is to use a package manager (e.g., npm) for handling library dependencies. This eliminates the clone-and-own risks yet allows for following best practices, such as, using libraries. If the file is cloned anyway, a comment including the repository, commit hash of the version cloned, and the summary of modifications (if any) should be added. This helps to improve traceability of the file.

In the current case, one should import the `@layerzerolabs/contracts` package using `yarn` or `npm`.

HASH-23
Messenger Configurations Changes May Fall Out of Sync

Undetermined | Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> Acknowledged. A process will be created to ensure safe migration in such exceptional circumstances.

File(s) affected: `HashflowLayerZeroMessenger.sol`, `HashflowWormholeMessenger.sol`

Description: The cross-chain messengers manage mappings from the `hChainId` to the `wormholeChainId` / `layerZeroChainId`. It is important that configuration changes are synced between chains to avoid failing to process messages that were en route.

Recommendation: Confirm that the message queues are empty before making configuration changes. Consider pausing the contracts while the configuration updates complete on all relevant chains.

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).

Code Documentation

1. Each repository's `README.md` file should contain basic instructions on how to run the test suite and any prerequisites for running tests. For each of these prerequisites also specify the versions supported/tested.
2. Multiple comments in `IHashflowRouter.sol` erroneously refer to RFQ-T instead of RFQ-M
    1. #139
    2. #143
    3. #159
    4. #167
    5. #185
    6. #195
3. Include a comment describing the structure of `authorizedXChainPools` at `HashflowRouter.sol#38`.

Adherence to Best Practices

1. In `HashflowWormholeMessenger`, when sending a message using `publishMessage()`, pass `nonce` as a parameter instead of hardcoding it as zero.
2. In some places `msg.sender` is used, whereas in other places, `_msgSender()` is used. Consider reconciling these instances for consistency.
3. `updateXChainCallerAuthorization()` contains a check for whether the caller is a contract. Consider adding the same check in `updateXChainMessengerCallerAuthorization()`.
4. The `IERC20AllowanceExtension` interface defined in `HashflowPool` can be removed.

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.

Contracts

- `9fb...614 ./contracts/HashflowFactoryZkSync.sol`
- `7c8...065 ./contracts/HashflowRouter.sol`
- `6af...9f3 ./contracts/HashflowFactory.sol`
- `022...81a ./contracts/interfaces/IHashflowRouter.sol`

- `b48...f72 ./contracts/interfaces/IQuote.sol`
- `c1a...81e ./contracts/interfaces/IHashflowPool.sol`
- `31c...a42 ./contracts/interfaces/IHashflowFactory.sol`
- `2a1...8f5 ./contracts/interfaces/external/ILayerZeroOracle.sol`
- `1ea...248 ./contracts/interfaces/external/IWormhole.sol`
- `878...e48 ./contracts/interfaces/external/ILayerZeroEndpoint.sol`
- `f7f...c4e ./contracts/interfaces/external/IWETH.sol`
- `369...598 ./contracts/interfaces/external/ILayerZeroNonceContract.sol`
- `e26...c05 ./contracts/interfaces/external/ILayerZeroRelayer.sol`
- `d00...a36 ./contracts/interfaces/external/ILayerZeroMessagingLibrary.sol`
- `8d8...225 ./contracts/interfaces/external/ILayerZeroReceiver.sol`
- `641...517 ./contracts/interfaces/external/ILayerZeroUserApplicationConfig.sol`
- `219...fd2 ./contracts/interfaces/external/WormholeStructs.sol`
- `d35...add ./contracts/interfaces/xchain/IHashflowWormholeMessenger.sol`
- `671...a93 ./contracts/interfaces/xchain/IHashflowXChainMessenger.sol`
- `12e...6ab ./contracts/interfaces/xchain/IHashflowLayerZeroMessenger.sol`
- `440...86f ./contracts/xchain/HashflowWormholeMessenger.sol`
- `cc5...9df ./contracts/xchain/HashflowXChainMessengerBase.sol`
- `1c8...287 ./contracts/xchain/HashflowLayerZeroMessenger.sol`
- `b0e...6ab ./contracts/testing/WETH9.sol`
- `fd6...b06 ./contracts/testing/WormoleVAAReader.sol`
- `88e...83b ./contracts/testing/USDC.sol`
- `59a...637 ./contracts/testing/USDT.sol`
- `9d1...54b ./contracts/testing/TestToken1.sol`
- `251...90a ./contracts/testing/TestToken2.sol`
- `4ca...874 ./contracts/testing/LZEndpointMock.sol`
- `72c...eec ./contracts/testing/LzLib.sol`
- `899...01a ./contracts/testing/DummyXChainApp.sol`
- `9e1...b40 ./contracts/testing/DummyMKR.sol`
- `e9c...8b7 ./contracts/pools/HashflowPool.sol`

Tests

- `8c0...1ce ./test/privatePoolZkSync.spec.ts`
- `d19...adf ./test/rfqm.spec.ts`
- `fdd...782 ./test/privPoolSingleHop.spec.ts`
- `626...719 ./test/contracts.ts`
- `14d...e75 ./test/utils.ts`
- `6c5...78b ./test/privPoolSingleHopEOA.spec.ts`
- `e53...875 ./test/xchain.spec.ts`
- `1e7...97f ./test/contractsZkSync.ts`

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.9.2

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Automated Analysis

Slither

Slither found issues concerning missing input validation.

Test Suite Results

Update: The test suite remains lacking.

Initial audit: The test suite is not complete in its current state. Consider, for example, the lack of tests for functions like `HashflowRouter.tradeXChainRFQTWithPermit()` and `HashflowRouter.fillXChain()`.

```
privPoolSingleHop
  Ownership
    ✔ should not renounce ownership (47ms)
  Pool creation
    ✔ should create a private pool
    ✔ should have correct WETH value
  Add liquidity
    ✔ should deposit Eth
    ✔ should deposit test token 1
  Trade
    ✔ should trade ETH for ERC-20 token (110ms)
    ✔ should fail if user has insufficient allowance or balance (41ms)
    ✔ should trade ERC-20 token for ETH (40ms)
    ✔ should fail if the quote expires
    ✔ should trade ERC-20 for ERC-20 token (42ms)
    ✔ should fail if effective base token exceeds max
    ✔ should trade if effective base token less than max
  Remove liquidity
    ✔ should remove eth
    ✔ should remove tt1

privPoolSingleHopExternalAccount
  Pool creation
    ✔ should create a private pool
  Trade
    ✔ should trade ETH for ERC-20 token (69ms)
    ✔ should fail if user has insufficient allowance or balance
    ✔ should fail if externalAccount has insufficient weth balance
    ✔ should trade ERC-20 token for ETH (40ms)
    ✔ should fail if the quote expires
    ✔ should fail if externalAccount has insufficient token balance
    ✔ should trade ERC-20 for ERC-20 token
    ✔ should fail if effective base token exceeds max
    ✔ should trade if effective base token less than max

privatePoolZkSync
  Ownership
    ✔ should not renounce ownership
  Pool Creation
    ✔ should create a Pool
    ✔ should create a Pool with the same metadata
  Add liquidity
    ✔ should deposit ETH
    ✔ should deposit test token 1
  Trade
    ✔ should trade ETH for ERC-20 token (67ms)

RFQ-m
  RFQM
    ✔ should trade (63ms)
    ✔ should trade with permit (81ms)
```

```
X-Chain
  Trade
    ✔ should trade x-chain (137ms)
    ✔ should trade x-chain RFQ-m (159ms)
    ✔ should trade x-chain with external accounts (86ms)
    ✔ should withdraw funds


36 passing (3s)
```

Code Coverage

Update: Coverage has actually decreased after the fixes were applied. We highly recommend improving coverage to 100% for all the contracts, and especially for those implementing core functionality like `HashflowRouter`.

Initial audit: Code coverage must be improved for `HashflowFactory` and `HashflowRouter` contracts.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	73.38	45.78	71.74	77.39	
HashflowFactory.sol	100	50	100	100	
HashflowFactoryZkSync.sol	100	50	100	100	
HashflowRouter.sol	68.94	44.85	65.79	73.99	... 725,864,939
contracts/interfaces/	100	100	100	100	
IHashflowFactory.sol	100	100	100	100	
IHashflowPool.sol	100	100	100	100	
IHashflowRouter.sol	100	100	100	100	
IQuote.sol	100	100	100	100	
contracts/interfaces/external/	100	100	100	100	
ILayerZeroEndpoint.sol	100	100	100	100	
ILayerZeroMessagingLibrary.sol	100	100	100	100	
ILayerZeroNonceContract.sol	100	100	100	100	
ILayerZeroOracle.sol	100	100	100	100	
ILayerZeroReceiver.sol	100	100	100	100	
ILayerZeroRelayer.sol	100	100	100	100	
ILayerZeroUserApplicati	100	100	100	100	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
onConfig.sol					
IWETH.sol	100	100	100	100	
IWormhole.sol	100	100	100	100	
WormholeStructs.sol	100	100	100	100	
contracts/interfaces/xchain/	100	100	100	100	
IHashflowLayerZeroMessenger.sol	100	100	100	100	
IHashflowWormholeMessenger.sol	100	100	100	100	
IHashflowXChainMessenger.sol	100	100	100	100	
contracts/pools/	85.39	51.79	75.86	81.98	
HashflowPool.sol	85.39	51.79	75.86	81.98	... 444,446,448
contracts/testing/	42.54	33.72	29.41	42.02	
DummyMKR.sol	0	100	0	0	... 24,28,32,40
DummyXChainApp.sol	0	100	0	0	... 25,35,46,58
LZEndpointMock.sol	43.04	34	28.13	41.88	... 537,556,560
LzLib.sol	38.46	25	28.57	33.33	... 127,128,136
TestToken1.sol	100	100	100	100	
TestToken2.sol	100	100	100	100	
USDC.sol	0	100	0	0	16,17,21
USDT.sol	0	100	0	0	16,17,21
WETH9.sol	92.31	75	83.33	94.74	50
WormoleVAAReader.sol	0	0	0	0	... 54,62,64,67
contracts/xchain/	38.52	20.55	51.72	45.26	
HashflowLayerZeroMessenger.sol	59.74	38.24	73.33	67.01	... 413,423,437

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
HashflowWormholeMessenger.sol	0	0	0	0	... 339,341,346
HashflowXChainMessengerBase.sol	100	50	100	100	
All files	58.2	37.84	52.33	59.59	

Changelog

- 2023-06-16 - Initial report
- 2023-07-10 - 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

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Quantstamp.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites&aspo; owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that your access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring 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.

Quantstamp