Hashflow Hashverse

Executive Summary

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

Timeline	2023-02-27 through 2023-03-03
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Source Code	• hashflownetwork/hashverse-contracts #b682e9d
Auditors	• Rabib Islam Auditing Engineer • Ibrahim Abouzied Auditing Engineer • Jeffrey Kam Auditing Engineer • Cameron Biniamow Auditing Engineer

Documentation quality	Medium
Test quality	Medium
Total Findings	20 Fixed: 16 Acknowledged: 3 Mitigated: 1
High severity findings	1 Fixed: 1
Medium severity findings	1 Fixed: 1
Low severity findings	11 Fixed: 10 Mitigated: 1
Undetermined severity findings	1 Acknowledged: 1
Informational findings	6 Fixed: 4 Acknowledged: 2

Summary of Findings

Update: The client addressed all the issues discussed in the report.

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

The contracts within this repository form the basis for the on-chain behaviour of an NFT gaming system. Although it is to be integrated with a DAO, this aspect was not part of the audit. Instead, the audit focused on the NFT contracts, an administrative module termed `RenovaCommandDeck`, and a quest contract where users may use their NFTs and may sometimes engage in trading with the Hashflow trading system.

Of particular note is the use of Wormhole, a cross-chain generalized message passing protocol that is being used in the current instance to enable two types of functionality:

1. cross-chain minting of player avatar NFTs; and
2. cross-chain transfers of item NFTs.

While avatar NFTs can be freely minted, item NFTs can only be minted by authorized accounts and in situations determined by a Merkle tree system; for example, when successful in a quest, the admin may update a Merkle tree indicating that a user is eligible to mint an item, and then provide them a Merkle proof that allows them to mint that item.

It is also important that quests are facilitated through the `RenovaQuest` contract, which is deployed for every quest that players may take part in, and which may custody users' funds when they interact with it for the purposes of trading.

Therefore, the audit was mainly focused around the following key areas of concern:

- Proper interaction with the Wormhole contracts;
- Correct accounting of custodied funds in `RenovaQuest`;
- Appropriate implementation of minting and transfer functionalities for the NFTs.

The most severe issue found is in relation to accounting of custodied funds: when dealing with deflationary tokens, some users may be unable to retrieve all the tokens to which they are entitled.

With regards to Wormhole, we identified an issue concerning the lack of an appropriate recipient contract on the destination chain: if there is

none, then the transferred item is effectively burned. We also discuss issues about fees and relayers that are not problems now, but may become problems in the future.

Although the code is commented very well, the project overall is severely lacking in technical documentation that details the intended functioning of the smart contracts. We did our best to remedy this issue by engaging in extensive communication with the client during the course of the audit. The test suite, though supplemented with Wormhole integration tests, is also lacking, and should be improved before deploying the project to production.

We recommend that the client address and/or consider all the findings included in this report.

ID	DESCRIPTION	SEVERITY	STATUS
QS-1	**Participating in Quests with Tokens that Do Not Preserve Amount on Transfer May Lead to Improper Accounting**	High	Fixed
QS-2	**Upgradeable Contracts Should Include Constructors with `_disableInitializers()`**	Low	Fixed
QS-3	**Items Can Be Bridged to Inactive Chain**	Medium	Fixed
QS-4	**Denial of Service for Quests when `_maxPlayers` Is Small**	Low	Fixed
QS-5	**Unable to Pay Wormhole Fee During `publishMessage()`**	Low	Fixed
QS-6	**Solidity Version `0.8.13` Is Bugged**	Low	Fixed
QS-7	**Unchecked Return Value During ERC20 Approval**	Low	Fixed
QS-8	**`XChainBridgeOut` Event Emitted with Incorrect `hashverseItemId`**	Low	Fixed
QS-9	**Trapped Items when the Item Address Is Updated During Active Quests**	Low	Fixed
QS-10	**Privileged Roles and Ownership**	Low	Mitigated
QS-11	**Missing Input Validation**	Low	Fixed
QS-12	**Potential Locked Funds Due to Positive Slippage**	Low	Fixed
QS-13	**Greedy Contract**	Low	Fixed
QS-14	**Block Timestamp Manipulation**	Informational	Acknowledged
QS-15	**Code Reuse May Lead to Wormhole Replay Attack Due to VAA Hash Not Being Marked as Processed**	Informational	Fixed
QS-16	**Cannot Create Quests without Items**	Informational	Fixed
QS-17	**Incorrect Event Emission for `updateHashflowRouter()`**	Informational	Fixed
QS-18	**Incompatibility with Generic Wormhole Relayer**	Informational	Acknowledged
QS-19	**A Singular Merkle Tree Cannot Reward a User Multiple Times**	Informational	Fixed
QS-20	**High Degree of Reliance on Off-Chain Components**	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**
>
> If the final commit hash provided by the client contains features that are not within the scope of the audit or an associated fix review, those features are excluded from consideration in this report.
>
> The following files are considered out of the scope of this audit:
>
> - `contracts/StakingVault.sol`
> - `contracts/interfaces/IStakingVault.sol`
>
> It is important to note that this project relies heavily on off-chain components, and that these are also out of the scope of this audit. See QSP-20 for details.

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.

Findings

QS-1

Participating in Quests with Tokens that Do Not Preserve Amount on Transfer May Lead to Improper Accounting

High · Fixed

> **Update**
>
> Marked as "Fixed" by the client. Addressed in: `4e0eaccacc44a9a7391d95207a9ba8868bfe16dd` . The client provided the following explanation:
>
> > We create a token whitelist and also delegate quest ownership to a new address (the quest owner) -- this is due to the fact that it's likely for this key to be hotter than the key used to deploy contracts.

File(s) affected: `RenovaQuest.sol`

Description: When a user deposits tokens into a quest, they do so using a struct, `TokenDeposit` , that includes the address of the token and the amount that will go into `safeTransferFrom()` . Moreover, the amount of tokens that a user is allowed to trade with in a quest depends on the `portfolioTokenBalances` , which is increased by the `TokenDeposit` amount when depositing tokens via `_depositTokens()` . As a result, if the `transferFrom` function does not act strictly via the input amount when changing balances (for example, in the case of deflationary tokens), then users may be able to trade with more tokens than actually reached the contract when depositing them, as other users

may have deposited funds ahead of them. Consequently, the users who trade later may be unable to trade with the full amount of what they deposited.

`potfolioTokenBalances` is also used when determining how much users are able to withdraw. Hence, if the amount of a token in the contract is lower than the cumulative amount recorded in `portfolioTokenBalances`, some users will be unable to withdraw in accordance with their rightful proportion of the total funds in the contract.

Recommendation: Either use a whitelist to prevent users from using deflationary tokens, or implement an accounting system that appropriately records the amounts that actually get sent to the contract.

QS-2
Upgradeable Contracts Should Include Constructors with `_disableInitializers()`

Low · Fixed

> **Update**
> Marked as "Fixed" by the client. The relevant constructors now contain the code. Addressed in: `74496b665998e7cb3b9cb7101756e78cc1569d1a`.

File(s) affected: `All upgradeable contracts`

Description: The implementation contract behind a proxy can be initialized by any address. This is not a security problem in the sense that it impacts the system directly, as the attacker will not be able to cause any contract to self-destruct or modify any values in the proxy contracts. However, taking ownership of implementation contracts can open other attack vectors, like social engineering or phishing attacks.

Recommendation: Call `_disableInitializers()` in constructors to avoid the initialization of the implementation contracts.

QS-3 Items Can Be Bridged to Inactive Chain

Medium · Fixed

> **Update**
> Marked as "Fixed" by the client. The check was added. Addressed in: `74496b665998e7cb3b9cb7101756e78cc1569d1a`.

File(s) affected: `RenovaItemBase.sol`, `WormholeBaseUpgradeable.sol`

Description: Users can bridge their items between chains by calling `wormholeBridgeOut()` in `RenovaItem` and `RenovaItemSatellite`. The user chooses the destination Wormhole chain ID with `dstWormholeChainId`. If the user passes the chain ID for a chain that does not have a `RenovaItemSatellite` contract deployed, the item is essentially lost as the item is burnt on the source chain during the bridging. There are no relevant checks to ensure that the destination chain is configured for bridging Renova items.

Recommendation: Implement the check `_wormholeRemotes[dstWormholeChainId] != bytes(0)` to ensure a valid destination chain is selected.

QS-4 Denial of Service for Quests when `_maxPlayers` Is Small

Low · Fixed

> **Update**
> Marked as "Fixed" by the client. Addressed in: `e83e71282909ae73758225b1c3a4672cfbde4812`. The client provided the following explanation: We require a minimum amount of vote-escrowed HFT in order to mint an Avatar. This should make it economically infeasible for a bad actor to DoS Avatar creation / Quest entering.

File(s) affected: `RenovaQuest.sol`

Description: Users can enter a quest as long as it is before the start time and the maximum number of players has not been reached. Once a user has entered a quest, `numRegisteredPlayers` is incremented. If `_maxPlayers` is small, a malicious actor could quickly register effectively inactive players until the maximum number of players is met. This would cause the malicious actor to be the only active player in the quest and deny other users the ability to play.

Additionally, when entering a quest, `numRegisteredPlayersPerFaction` is incremented for the faction the user selected. If a quest is designed to have opposing factions compete against each other, a malicious actor could quickly register inactive players for the opposing faction to increase the malicious actor's chances of winning.

Recommendation:

1. As `_maxPlayers` increases, so does the gas cost of a potential malicious attack aimed at filling all available spots; hence a larger number may mitigate such attacks.

2. Consider refactoring the contracts to enforce a minimum deposit amount of ERC20 tokens for every player that is registering. Withdrawals of the ERC20s before the start of the quest would need to be disabled as a malicious actor could continuously deposit and withdraw the same tokens to register a large number of players.

QS-5 Unable to Pay Wormhole Fee During `publishMessage()`

Low Fixed

> **Update**
> Marked as "Fixed" by the client. Addressed in: `f1b72c81d9d026b834118e7994cff7172ef92958` . The client provided the following explanation: Instead of querying the fee on-chain, we query it off-chain and pass it over to the contract. This can theoretically revert, but it does save gas, as it avoids running a cross-contract call + SLOAD.

File(s) affected: `WormholeBaseUpgradeable.sol`

Description: In order to use Wormhole, the user must first cause the emission of a Verifiable Action Approval (VAA) by callling the `publishMessage()` function of the Wormhole core layer implementation contract . In order for this call to succeed, the call's `msg.value` field must be equal to `messageFee()` , a variable of the implementation contract.

In `WormholeBaseUpgradeable` , there is a call to `publishMessage()` , but it is always with `msg.value` set to zero. Currently, `messageFee()` returns zero. However, in the future, it is possible for Wormhole to increase this fee, causing all attempts to call `publishMessage()` through `WormholeBaseUpgradeable` to revert, and hence preventing any attempts at cross-chain minting of avatars and cross-chain transferring of items.

Recommendation: We recommend implementing a means to fulfill the fee checking component of `publishMessage()` for the possible case where `messageFee()` is increased.

QS-6 Solidity Version `0.8.13` Is Bugged

Low Fixed

> **Update**
> Marked as "Fixed" by the client. The Solidity version was updated to `0.8.19` . Addressed in: `74496b665998e7cb3b9cb7101756e78cc1569d1a` .

Description: Solidity versions `0.8.13` and `0.8.14` contain important bugs, that if exploited, could result in serious issues. These bugs were fixed in `0.8.15` .

Recommendation: We recommend upgrading the compiler version to `0.8.15` or above.

QS-7 Unchecked Return Value During ERC20 Approval

Low Fixed

> **Update**
> Marked as "Fixed" by the client. The return value of `approve()` is now being checked. Addressed in: `74496b665998e7cb3b9cb7101756e78cc1569d1a` .

File(s) affected: `RenovaQuest.sol`

Description: In `trade()` , the function makes an ERC20 approval but does not check the return value. As a result, if the `approve()` function returns false, this may go entirely unnoticed depending on the implementation behind `IHashflowRouter` .

Recommendation: Consider using the wrapper functions in `SafeERC20` . However, note that not all tokens (e.g. USDT) react correctly to `safeIncreaseAllowance()` and `safeDecreaseAllowance()` .

QS-8 `XChainBridgeOut` Event Emitted with Incorrect `hashverseItemId`

Low Fixed

> **Update**
> Marked as "Fixed" by the client. Addressed in: `74496b665998e7cb3b9cb7101756e78cc1569d1a` .

File(s) affected: `RenovaItemBase.sol`

Description: When bridging a Renova item to another chain, the user calls `wormholeBridgeOut()` on the source chain. After the item is burnt on the source chain, the line `delete _hashverseItemIds[tokenId];` deletes the Hashverse item ID associated with the token ID. However, after deleting the Hashverse item ID from the mapping, the event `XChainBridgeOut` calls `_hashverseItemIds[tokenId]` again to get the associated Hashverse ID. Therefore, the emitted Hashverse item ID will always be `0` .

Recommendation: Store the value of `_hashverseItemIds[tokenId]` in a memory variable before deletion. Use the memory variable within the `XChainBridgeOut` event.

QS-9
Trapped Items when the Item Address Is Updated During Active Quests

Low Fixed

> **Update**
> Marked as "Fixed" by the client. Addressed in: `eacc01d66462e85967df2acaef53161d2af0d532` . The client provided the following explanation: We make the Item and Avatar immutable. They should only be mutated in the case of a catastrophic event, during which it is fair to say that the Command Deck should be re-deployed.

File(s) affected: `RenovaCommandDeckBase.sol` , `RenovaQuest.sol`

Description: The Hashflow team retains the ability to update `renovaItem` by calling `RenovaCommandDeckBase.updateRenovaItem()` . If the Renova item contract is updated while a user has loaded their items into a quest, the user will not be able to unload their items once the quest is complete. This is due to `RenovaQuest.unloadAllItems()` calling `RenovaCommandDeckBase.renovaItem()` to get the Renova item address, which is then used to transfer the items back to the user.

Recommendation: Updating the Renova item contract should only ever be done when **all quests are inactive** and **all users have unloaded their items from the quests**. Keep in mind that a user can load items into a quest, play the quest, and then forget to unload their items once the quest is complete.

Confirm if this is the intended functionality. If not, consider removing `updateRenovaItem()` from `RenovaCommandDeckBase` .

QS-10 Privileged Roles and Ownership

Low Mitigated

> **Update**
> Marked as "Mitigated" by the client. Addressed in: `eacc01d66462e85967df2acaef53161d2af0d532` , `4e0eaccacc44a9a7391d95207a9ba8868bfe16dd` . The client provided the following explanation:
> 1. Once a Quest is created, the Router, Avatar, and Item stay immutable. This means that users can participate in a Quest without having to worry about their funds, considering that the Hashflow Router / Avatar / Item contracts that the Quest was initialized with can be trusted.
> 2. To improve OpSec, we add the concept of a Quest Owner -- this will be an occasionally hot address that can create quests and upload roots.

File(s) affected: `RenovaCommandDeck.sol` , `RenovaCommandDeckBase.sol` , `RenovaAvatarBase.sol` , `RenovaItem.sol` , `RenovaItemBase.sol` , `WormholeBaseUpgradeable.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract.

In the case where a malicious actor obtains control of the owner's wallet, privileged functionality can be exploited in a way that harms future users. Specifically, a malicious actor could call `updateHashflowRouter()` in `RenovaCommandDeck` or `RenovaCommandDeckSatellite` to change the router address to a malicious contract. After the router is updated, users that call `RenovaQuest.trade()` are subject to having their funds stolen by the malicious router contract.

The following are all other cases of owner privileges:

- `RenovaCommandDeck.uploadItemMerkleRoot()`
- `RenovaCommandDeck.mintItemAdmin()`
- `RenovaCommandDeckBase.createQuest()`
- `RenovaCommandDeckBase.updateRenovaAvatar()`
- `RenovaCommandDeckBase.updateRenovaItem()`
- `RenovaAvatarBase.refreshMetadata()`
- `RenovaAvatarBase.refreshAllMetadata()`
- `RenovaItem.updateMinterAuthorization()`
- `RenovaItem.mint()`
- `RenovaItemBase.setDefaultRoyalty()`
- `RenovaItemBase.setCustomBaseURI()`
- `RenovaItemBase.refreshMetadata()`
- `RenovaItemBase.refreshAllMetadata()`
- `WormholeBaseUpgradeable.updateWormhole()`
- `WormholeBaseUpgradeable.updateWormholeConsistencyLevel()`
- `WormholeBaseUpgradeable.updateWormholeRemote()`

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.

QS-11 Missing Input Validation

Low Fixed

> **Update**
> Marked as "Fixed" by the client. Addressed in: `eacc01d66462e85967df2acaef53161d2af0d532` and `a9a75efe6a657cb36c96d614287073b5dc83db34`.

File(s) affected: `RenovaCommandDeckBase.sol`, `RenovaAvatarBase.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`:

- `RenovaCommandDeckBase.updateRenovaItem()`: `_renovaItem`
- `RenovaCommandDeckBase.updateRenovaAvatar()`: `_renovaAvatar`
- `RenovaCommandDeckBase.updateHashflowRouter()`: `_hashflowRouter`
- `RenovaAvatarBase.__RenovaAvatarBase_init()`: `_renovaCommandDeck`

Recommendation: We recommend adding the relevant checks.

QS-12 Potential Locked Funds Due to Positive Slippage

Low Fixed

> **Update**
> Marked as "Fixed" by the client. Strict equality is now enforced. Addressed in: `74496b665998e7cb3b9cb7101756e78cc1569d1a`.

File(s) affected: `RenovaQuest.sol`

Description: The `require` statement in `RenovaQuest.trade()#L335` checks that `balanceBefore + quoteTokenAmount <= balanceAfter`. If the trade has positive slippage, say the user receives more tokens than quoted, then the extra funds will be locked in the account. In Hashflow's documentation, it is claimed that the trading platform provides zero-slippage trade, so this should not happen. If this is indeed the case, we should enforce strict equality instead, namely `balanceBefore + quoteTokenAmount == balanceAfter`.

Recommendation: Confirm whether it should be possible to have positive slippage. If not, consider replacing the requirement with equality. Otherwise, consider implementing a mechanism that would appropriately handle new funds that are in excess of `quoteTokenAmount`.

QS-13 Greedy Contract

Low Fixed

> **Update**
> Marked as "Fixed" by the client. A function to withdraw ETH has been implemented. Addressed in: `f1b72c81d9d026b834118e7994cff7172ef92958`.

File(s) affected: `RenovaItemBase.sol`

Description: A greedy contract is a contract that can receive ether which can never be redeemed.

`RenovaItemBase.wormholeBridgeOut()` is a payable function which means that users can send ETH to the contract using this function; however, at no point does this contract use any ETH in its functions, nor is there any means to withdraw ETH from the contract.

Notably, ETH would ordinarily be used for interacting with Wormhole via `publishMessage()`.

Recommendation: The developer should implement one or both of the following two solutions:

1. Implement a function that allows an admin to recover ETH from the contract;
2. Use ETH sent to the contract to interact with Wormhole.

QS-14 Block Timestamp Manipulation

Informational Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation: Block timestamps can be off by a few seconds, which we acknowledge. However, they are merely used for quest start / end time, which should be large intervals (e.g. days) apart. A few seconds are not going to pose any major issues during quests, whether it's PnL accounting or DoS.

File(s) affected: `RenovaQuest.sol`

Related Issue(s): SWC-116

Description: Projects may rely on block timestamps for various purposes. Given the cross-chain nature of this application, it is important to note that different networks may impose different requirements for their validators when it comes to setting the timestamp of a block, and attackers may be able to manipulate timestamps for their own purposes. If a smart contract relies on a timestamp, it must take this into account.

Recommendation: Consider the degree to which validators may influence `block.timestamp` on the target networks.

QS-15
Code Reuse May Lead to Wormhole Replay Attack Due to VAA Hash Not Being Marked as Processed

• Informational ⓘ Fixed

> **Update**
> Marked as "Fixed" by the client. Processed message hashes are now being recorded. Addressed in: `74496b665998e7cb3b9cb7101756e78cc1569d1a` .

File(s) affected: `WormholeBaseUpgradeable.sol`

Description: One attack vector that must be accounted for when implementing cross-chain functionality is that of replay attacks. It is possible for someone to design a system implementing Wormhole that allows a VAA coming from some chain to be transmitted more than once on the destination chain, as the Wormhole core layer contract does not prevent this. Therefore, it is the responsibility of the system designer to determine whether this is a desirable trait and, if not, to implement a countermeasure. In the case of Hashverse, there are two possible cross-chain messages to consider with respect to this issue.

The first message is cross-chain minting of avatars. Any address is able to mint an avatar NFT; this NFT is soulbound (the transfer function is disabled) and the first minting must be on Ethereum Mainnet. It also cannot be burned. Subsequently, users can use the `RenovaAvatar.wormholeMintSend()` function to mint this avatar NFT on another chain and to the same address as it belonged to on Ethereum Mainnet. Attempting to replay the emitted VAA would accomplish nothing as the previously cross-chain minted NFT would still be at the same address, causing the attempt at replaying the VAA to revert.

The second message is cross-chain transferring of items. Items are NFTs that can only be initially minted on Ethereum Mainnet. These mints are controlled -- they either require the minting user to be directly authorized by the NFT contract owner, or they must be minted through the command deck, either directly by the admin of the command deck, or a process involving Merkle trees controlled by the admin of the command deck. Once minted, an item can be sent to another chain by use of the `RenovaItem.wormholeBridgeOut()` function -- this burns the item on the source chain and emits a VAA that when received on the destination chain will mint the item with the same `tokenId` . To prevent replaying of the VAA, each VAA is sent with a nonce associated with the item and that increases every time the item is sent to another chain. Hence, attempting to replay the VAA would only result in execution being reverted.

Therefore, there are overall no concerns about cross-chain replay attacks in the current project. However, the `WormholeBaseUpgradeable` contract is an abstract contract that developers may want to reuse in other projects. It is important to note that, in general, this contract does not prevent replaying VAAs in the `_wormholeReceive()` function.

Recommendation: Be careful when considering whether cross-chain replays of VAAs should be allowed when reusing `WormholeBaseUpgradeable` as it does not prevent replays by default. The Wormhole Development Book offers some best practices regarding replaying VAAs: If a message should not be replayed,

1. Immediately mark its hash as processed upon receipt;
2. Strongly consider enforcing an intended recipient, as otherwise, others may be able to use VAAs without the developer's/user's knowledge or approval.

QS-16 Cannot Create Quests without Items

• Informational ⓘ Fixed

> **Update**
> Marked as "Fixed" by the client. Addressed in: `eacc01d66462e85967df2acaef53161d2af0d532` . The client provided the following explanation: We no longer allowed uncapped Quests. In reality, it will be more frequent that we create Quests with no items instead of Quests with unlimited items. For the latter, we can use an arbitrarily large number as the limit.

File(s) affected: `RenovaQuest.sol`

Description: When creating quests, the maximum number of items is assigned to `_maxItemsPerPlayer` . In `_loadItems()` , if `_maxItemsPerPlayer` is set to zero, then the maximum number of items is `type(uint256).max` . Therefore, the smallest maximum number of items a quest can have is one.

Recommendation: Confirm whether this is the intended functionality.

QS-17 Incorrect Event Emission for `updateHashflowRouter()`

Informational | Fixed

> **Update**
> Marked as "Fixed" by the client. The order has been corrected. Addressed in: `597a3bcd7b53c0037a7aaf4d12965e5763d7a886`.

File(s) affected: `RenovaCommandDeckBase.sol`

Description: The ordering of the parameters to the event `UpdateHashflowRouter` in `updateHashflowRouter()` is incorrect, as it currently suggests that we are updating the Hashflow router from the new router to the old one, based on the documentation of the event in the interface.

Recommendation: Replace `UpdateHashflowRouter(hashflowRouter, _hashflowRouter)` with `UpdateHashflowRouter(_hashflowRouter, hashflowRouter)`.

QS-18 Incompatibility with Generic Wormhole Relayer

Informational | Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation: We have looked into the Generic Wormhole Relayer. Given that it is not live, and the general best practices are to send the Request-For-Relay in the same transaction, we plan on adding the functionality as an upgrade when the time comes.

File(s) affected: `RenovaAvatarSatellite.sol`, `RenovaItemBase.sol`

Description: In discussion with the Hashflow team, we learned that they plan to operate specialized Wormhole relayers and have the client relay the VAA as a fallback measure. It should be noted that the current implementation does not work with the generic relayer, which is currently only available on testnet.

The generic relayer system requires that there is a function in the target contract on the destination chain called `receiveWormholeMessages()` as an entry point for execution. However, this function is not implemented in contracts that receive VAAs from Wormhole, e.g. `RenovaAvatarSatellite` and `RenovaItemBase`.

For a reference implementation, see the last section of the Wormhole documentation on Best Practices here.

This would additionally require proper use of nonces, as they are used to batch VAAs before they are relayed. However, they are currently hardcoded to zero.

Recommendation: Determine whether interaction with the generic relaying system once it reaches mainnet is desirable, and if so, implement the `receiveWormholeMessages()` function on potential target contracts. Moreover, ensure that the nonce is configured for each VAA.

QS-19 A Singular Merkle Tree Cannot Reward a User Multiple Times

Informational | Fixed

> **Update**
> Marked as "Fixed" by the client. A new `mintIdx` variable allows trees to mint multiple times to the same user. Addressed in: `4fc10a1213111483f101edd3e6f9802cdf60cbd1`.

File(s) affected: `RenovaCommandDeck.sol`

Description: In `mintItem()`, minted items are tracked using the `rootId` and `tokenOwner` address. If a `rootId` happens to reward a user multiple times, every call to `mintItem()` after the first one will fail.

Recommendation: Ensure that all roots will reward at most one item to a player, or track processed item mints in a way that allows for multiple item mints for a given `rootId`.

QS-20 High Degree of Reliance on Off-Chain Components

Undetermined | Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation: Documentation will be added to the website to explain the mechanics in detail.

Description: In discussion with the client, it was determined that the vast majority of user interactions with the dApp will occur in off-chain components, meaning the proper functioning of the system as a whole largely depends on off-chain code that is not within the scope of this audit. This makes it difficult for us to evaluate the risks associated with interacting with the application, especially when there is a lack of documentation for these off-chain computations (e.g. quest reward mechanism, Merkle tree computations).

Furthermore, the cross-chain component of the application will be facilitated by a relaying system that is not within the scope of this audit.

Recommendation: Consider writing documentation for the off-chain components, as it aids in increasing accountability for and understanding of the system.

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. The code is well-commented.
2. The project is lacking a detailed specification that outlines the functional requirements of the system. This is especially concerning given the large degree of involvement of off-chain computation. Unresolved

Adherence to Best Practices

1. The check at `RenovaAvatarBase.sol#147` is redundant as it occurs within `_mint()` which is called further down. Fixed
2. Given that `RenovaAvatar` NFTs are soulbound, approval-related functions such as `approve()` and `setApprovalForAll()` have no purpose. Consider overriding these functions to revert whenever called. Fixed

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

- `096...13a ./contracts/interfaces/Interfaces.sol`
- `48b...a12 ./contracts/interfaces/external/IWormhole.sol`
- `a70...fcb ./contracts/interfaces/external/IHashflowRouter.sol`
- `618...6c3 ./contracts/interfaces/external/IWormholeStructs.sol`
- `3e9...db5 ./contracts/interfaces/external/erc/IERC4906Upgradeable.sol`
- `b2d...1a3 ./contracts/interfaces/staking/IStakingVault.sol`
- `25e...401 ./contracts/interfaces/wormhole/IWormholeBaseUpgradeable.sol`
- `bf5...e2f ./contracts/interfaces/nft/IRenovaItemSatellite.sol`
- `9bc...541 ./contracts/interfaces/nft/IRenovaAvatarBase.sol`
- `68f...115 ./contracts/interfaces/nft/IRenovaAvatar.sol`

- 38a...4d8 ./contracts/interfaces/nft/IRenovaAvatarSatellite.sol
- 3c1...4eb ./contracts/interfaces/nft/IRenovaItem.sol
- 302...9cc ./contracts/interfaces/nft/IRenovaItemBase.sol
- 27e...5fa ./contracts/interfaces/core/IRenovaCommandDeckBase.sol
- 3ff...3c7 ./contracts/interfaces/core/IRenovaQuest.sol
- 62d...1da ./contracts/interfaces/core/IRenovaCommandDeckSatellite.sol
- 9b0...4c7 ./contracts/interfaces/core/IRenovaCommandDeck.sol
- 66c...78f ./contracts/staking/StakingVault.sol
- 372...7db ./contracts/wormhole/WormholeBaseUpgradeable.sol
- 27b...bd9 ./contracts/nft/RenovaItem.sol
- 2b4...c53 ./contracts/nft/RenovaAvatarSatellite.sol
- 22b...b3c ./contracts/nft/RenovaItemSatellite.sol
- 6bc...7fa ./contracts/nft/RenovaAvatarBase.sol
- c84...bea ./contracts/nft/RenovaAvatar.sol
- 2bd...93f ./contracts/nft/RenovaItemBase.sol
- 8d4...dac ./contracts/test/MockHashflowRouter.sol
- d38...e20 ./contracts/test/HFT.sol
- e26...b3d ./contracts/test/TestERC1271.sol
- 577...c55 ./contracts/test/TestERC20.sol
- 8cf...4ae ./contracts/test/MockWormhole.sol
- bb0...1c8 ./contracts/core/RenovaCommandDeckBase.sol
- 263...023 ./contracts/core/RenovaCommandDeck.sol
- 14d...40d ./contracts/core/RenovaCommandDeckSatellite.sol
- 238...a11 ./contracts/core/RenovaQuest.sol

Tests

- 4ef...45f ./test/renova.spec.ts
- 9fa...d6f ./test/utils.ts
- 59d...e33 ./test/stakingVault.spec.ts
- 890...ca1 ./test/integration/wormhole.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 detected issues concerning missing input validation, use of `block.timestamp`, contracts locking ether, and unused return values. These issues were reported above. The remainder were not determined to be security issues of note.

Test Suite Results

The existing tests are well-implemented. We also appreciate that there are a great number of integration tests written to assess the ability of the contracts to interact with Wormhole. However, these integration tests are necessary but not sufficient in general, and more unit tests remain to be written.

```
Renova
  Avatar
    ✔ should support appropriate interfaces
    ✔ should mint avatar (61ms)
    ✔ should render correct metadata link
    ✔ should not double-mint
    ✔ should be soul-bound
  Command Deck
    ✔ should mint item via admin
    ✔ should mint item via Merkle root
    ✔ should create quest
    ✔ should not double create quest
  Quest
    ✔ should not create quest that starts in the past
    ✔ should not create quest that ends before it starts
    ✔ should not create quest that lasts too long
    ✔ should allow players to enter before quest starts (51ms)
    ✔ should not allow players without an avatar to enter (42ms)
    ✔ should not allow players to enter twice
    ✔ should respect player caps (44ms)
    ✔ should enter quest, load items, and deposit tokens
    ✔ should load items (118ms)
    ✔ should deposit assets (41ms)
    ✔ should trade (80ms)
    ✔ should withdraw assets (52ms)
    ✔ should unload items
  Item
    ✔ should support appropriate interfaces
    ✔ should correctly handle royalties
    ✔ should render correct metadata link

StakingVault
  Stake
    ✔ should stake (45ms)
    ✔ should increase HFT staked
    ✔ should extend HFT lock
    ✔ should stake with permit (50ms)
    ✔ should stake from ERC1271
  Withdraw
    ✔ should withdraw HFT (38ms)
  Transfer
    ✔ should transfer stake (62ms)
    ✔ should transfer ERC1271 stake


33 passing (2s)
```

Code Coverage

Coverage was obtained using `yarn hardhat coverage` . Branch coverage, as measured by Hardhat, is relatively poor overall in the current instance. We recommend increasing branch coverage to 100%, noting however that this metric does not include the functionality tested by the existing integration tests. These integration tests are, however, necessary.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lin
core/	93.4	61.29	81.48	91.97	
RenovaCommandDeck.sol	100	56.25	100	100	
RenovaCommandDeckBase.sol	80.95	42.31	50	75.86	... 162,164,16
RenovaCommandDeckSatellite.sol	0	0	0	0	25

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lin
RenovaQuest.sol	97.22	70	100	96.77	108,217,396
interfaces/	100	100	100	100	
Interfaces.sol	100	100	100	100	
interfaces/core/	100	100	100	100	
IRenovaCommandDeck.sol	100	100	100	100	
IRenovaCommandDeckBase.sol	100	100	100	100	
IRenovaCommandDeckSatellite.sol	100	100	100	100	
IRenovaQuest.sol	100	100	100	100	
interfaces/external/	100	100	100	100	
IHashflowRouter.sol	100	100	100	100	
IWormhole.sol	100	100	100	100	
IWormholeStructs.sol	100	100	100	100	
interfaces/external/erc/	100	100	100	100	
IERC4906Upgradeable.sol	100	100	100	100	
interfaces/nft/	100	100	100	100	
IRenovaAvatar.sol	100	100	100	100	
IRenovaAvatarBase.sol	100	100	100	100	
IRenovaAvatarSatellite.sol	100	100	100	100	
IRenovaItem.sol	100	100	100	100	
IRenovaItemBase.sol	100	100	100	100	
IRenovaItemSatellite.sol	100	100	100	100	
interfaces/staking/	100	100	100	100	
IStakingVault.sol	100	100	100	100	
interfaces/wormhole/	100	100	100	100	
IWormholeBaseUpgradeable.sol	100	100	100	100	
nft/	49.21	33.33	56.25	55	
RenovaAvatar.sol	28.57	16.67	66.67	44.44	46,51,56,65,6
RenovaAvatarBase.sol	77.78	50	63.64	83.33	89,119,124,12

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lin
RenovaAvatarSatellite.sol	0	0	0	0	... 36,55,60,6
RenovaItem.sol	80	33.33	66.67	77.78	51,53
RenovaItemBase.sol	42.31	31.82	58.33	41.94	... 163,168,17
RenovaItemSatellite.sol	0	0	0	0	21
staking/	89.39	59.72	76.92	89.66	
StakingVault.sol	89.39	59.72	76.92	89.66	... 256,258,2
test/	72.22	50	90	70	
HFT.sol	100	100	100	100	
MockHashflowRouter.sol	87.5	62.5	100	77.78	34,40
MockWormhole.sol	100	100	100	100	
TestERC1271.sol	42.86	0	80	50	24,26,27,29
TestERC20.sol	100	100	100	100	
wormhole/	37.5	7.14	37.5	38.71	
WormholeBaseUpgradeable.sol	37.5	7.14	37.5	38.71	... 123,128,13
All files	76.17	49.66	68.89	77.18	

Changelog

- 2023-03-03 - Initial report
- 2023-03-22 - 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, Aave, 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.