Quantstamp

Hemi - Re-Audit

Executive Summary

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

Type	Cross Chain BTC
Timeline	2024-12-13 through 2024-12-20
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Official Documentation Internal Documentation
Source Code	• hemilabs/bitcoin-tunnel-contracts #6497933
Auditors	• Julio Aguilar Auditing Engineer • Gereon Mendler Auditing Engineer

Documentation quality	High
Test quality	Low
Total Findings	14 Fixed: 11 Acknowledged: 2 Mitigated: 1
High severity findings	5 Fixed: 5
Medium severity findings	2 Fixed: 2
Low severity findings	4 Fixed: 3 Mitigated: 1
Undetermined severity findings	0
Informational findings	3 Fixed: 1 Acknowledged: 2

Summary of Findings

The Hemi BTC Tunnel is an innovative bridge connecting the Bitcoin network to the Hemi ecosystem, enabling users to mint `hBTC` through over-collateralized vaults. Vaults can be created and administrated by operators who compete with each other by offering different fees to users for their bridging service.

This is a re-audit that ran in parallel to the last part of the original audit. Although the codebase is well-documented, testing is insufficient, and several issues could have been avoided with more thorough testing of all processes. Due to the project's complexity, implementing robust monitoring systems at launch is highly recommended. Furthermore, since the vault involves intricate challenge mechanisms, users should stay vigilant and monitor operators for potential fraud.

Fix-Review Update: The Hemi team addressed all issues by fixing or acknowledging them providing a detailed explanation for each. However, the test coverage remains very low; we urge the team to continuously improve their tests before launching their service.

ID	DESCRIPTION	SEVERITY	STATUS
HEM-1	**Initial Bidders of Partial Collateral Liquidations Always Win and Get Other Bidders' Bids Due to the Usage of `memory` Instead of `storage` Keyword**	High	Fixed
HEM-2	**Inconsistency in the Withdrawals Output's Length Disrupts Proper Functioning of the Vaults**	High	Fixed
HEM-3	**Challenging a Withdrawal Is Not Possible Putting User Funds at Risk**	High	Fixed
HEM-4	**Price Changes Not Accounted for During Full Liquidation Could Lead to Funds Locked**	High	Fixed

ID	DESCRIPTION	SEVERITY	STATUS
HEM-5	`increasetotaldepositsheld()` Does Not Correctly Increase the `totalDepositsHeld` in some Cases	High	Fixed
HEM-6	Operator Might Not Be Able to Withdraw the Desired and Possible Collateral	Medium	Fixed
HEM-7	Undesired Behavior During Partial Collateral Withdrawal	Medium	Fixed
HEM-8	2-Step Admin Transfer Pattern Incorrectly Implemented	Low	Fixed
HEM-9	Lack of Access Control in `factory.createVault()` Allows for Phishing Attacks	Low	Fixed
HEM-10	Unnecessary Collateral Loss for the Operator	Low	Mitigated
HEM-11	Withdraw Fees Can Exceed Minimum Withdraw Amount	Low	Fixed
HEM-12	Insufficient `BTC` Security Confirmations	Informational	Acknowledged
HEM-13	`initiatewithdrawal()` Always Returns Fee as Zero	Informational	Fixed
HEM-14	Pending Unminted Fees Are Collateralized	Informational	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

The scope includes all the following files:

Files Included

- contracts/
  - governance/
  - AddressWhitelist.sol
  - GlobalConfig.sol
  - vaults/
  - SimpleBitcoinVault/
    - SimpleBitcoinVault.sol
    - SimpleBitcoinVaultFactory.sol
    - SimpleBitcoinVaultFactoryHelper.sol
    - SimpleBitcoinVaultState.sol
    - SimpleBitcoinVaultStateFactory.sol
    - SimpleBitcoinVaultStructs.sol
    - SimpleBitcoinVaultUTXOLogicHelper.sol
    - SimpleGlobalVaultConfig.sol
  - CommonStructs.sol
  - BitcoinTunnelManager.sol
  - BTCToken.sol

Files Excluded

- contracts/bitcoinkit/*
- contracts/oracle/*

Operational Considerations

1. The depositor must format the deposit Bitcoin transaction to contain the Bitcoin output corresponding to this vault and the OP_RETURN with their EVM address to credit for the deposit within the number of outputs returned by the BitcoinKit's built-in `getTransactionByTxId()`. We assume the protocol returns the BTC back to the sender on the Bitcoin chain if the deposit is deemed invalid.
2. The SimpleBitcoinVault contract assumes that the collateral tokens are "standard tokens" in the way that it should not have transfer fee nor able to rebase...etc. We assume the vault configuration is set correctly with this assumption being considered.
3. The implementation assumes that the hVM will be lagging two bitcoin blocks and thus the finalization threshold is set to 4 in a sense that it will wait 6 blocks in total.
4. Our review assumes that a robust oracle is used to provide the real-time BTC price to the system, and not the hBTC price. Any oracle implementation is out of scope of the audit and we trust that oracles undergo a separate audit and behave honestly.
5. Several updates in the codebase, such as minCollateralAssetAmount in SimpleGlobalVaultConfig and various delays in GlobalConfig, use "only-increase" logic without upper bounds, which could unintentionally block system flows if excessively high values are set due to manual errors. The team or the operator should be very careful whenever updating these configurations to avoid unintended disruptions.
6. We trust this protocol to be configured honestly and accurately. For example, it is possible that the SimpleBitcoinVaultFactory could be assigned a bitcoinTokenContract that differs from the one assigned in the BitcoinTunnelManager. However, we trust the team will deploy this with a matching token address.

Key Actors And Their Capabilities

One of the main actors is the **operator**, which is used in the `SimpleBitcoinVault` and `SimpleBitcoinVaultState`. Each vault is controlled by an operator who must deposit collateral and manage operations related to deposits, withdrawals, and sweeps. The operator is economically incentivized to act honestly through the collateral. This role has the privilege to call several functions in the `SimpleBitcoinVault`, such as `updateOperatorAdmin()`, `goLive()`, `closeVault()`, `depositCollateral()`, `windDownVault()`, and others.

Another important actor is the **minter** of the `hBTC` token which as the name suggests is allowed to mint but also to burn them. This role is assigned to the `BitcoinTunnerManager` contract which will mint and burn tokens based on certain actions from the vaults like deposits, withdrawals, and liquidations.

Additionally, from the `GlobalConfig` contract, there are several roles with different permissions to update or tweak global configurations:

1. **globalConfigAdmin**: has the highest authority and can manage all configuration settings, permissions, and latches in the system. The role has the privilege to call several functions in `GlobalConfig` such as `updateVaultFactoryUpgradeAdmin()`, `initiateGlobalConfigAdminUpgrade()`, `permDisableVaultUpgrades()`, `increas`

`eGlobalConfigAdminUpgradeDelay()`, `updateVaultCreationPauseAdmin()`, `updateWithdrawalPauseAdmin()`, and many more.
2. **vaultFactoryUpgradeAdmin**: manages upgrades to the `VaultFactory` implementation and controls the upgrade delay. The role has the privilege to call several functions in `GlobalConfig` such as `initiateVaultFactoryUpgrade()`, `updateVaultFactoryUpgradeDelay()`, and `rejectPendingVaultFactoryUpgrade()`.
3. **vaultFactoryUpgradeBypassAdmin**: allows bypassing the activation delay for `VaultFactory` upgrades. The role has the privilege to call the function `finalizeVaultFactoryUpgrade()` in `GlobalConfig` with the `force=true` parameter.
4. **vaultCreationPauseAdmin**: can pause the creation of new vaults in the system. The role has the privilege to call the `pauseVaultCreation()` function in `GlobalConfig`.
5. **vaultCreationUnpauseAdmin**: can unpause the creation of new vaults in the system. The role has the privilege to call the `unpauseVaultCreation()` function in `GlobalConfig`.
6. **vaultCreationWhitelistEnableAdmin**: can enable address whitelisting for vault creation. The role has the privilege to call the `enableVaultCreationWhitelist()` function in `GlobalConfig`.
7. **vaultCreationWhitelistDisableAdmin**: can disable address whitelisting for vault creation. The role has the privilege to call the `disableVaultCreationWhitelist()` function in `GlobalConfig`.
8. **vaultCreationWhitelistModificationAdmin**: can modify (add or remove) addresses in the vault creation whitelist. The role has the privilege to call several functions in `GlobalConfig` such as `addAddressToVaultCreationWhitelist()` and `removeAddressFromVaultCreationWhitelist()`.
9. **vaultCreationWhitelistAdditionAdmin**: can only add addresses to the vault creation whitelist. The role has the privilege to call the `addAddressToVaultCreationWhitelist()` function in `GlobalConfig`.
10. **withdrawalPauseAdmin**: can globally pause withdrawals from vaults. The role has the privilege to call the `pauseWithdrawals()` function in `GlobalConfig`.
11. **withdrawalUnpauseAdmin**: can globally unpause withdrawals from vaults. The role has the privilege to call the `unpauseWithdrawals()` function in `GlobalConfig`.
12. **withdrawalWhitelistEnableAdmin**: can enable address whitelisting for withdrawals. The role has the privilege to call the `enableWithdrawalWhitelist()` function in `GlobalConfig`.
13. **withdrawalWhitelistDisableAdmin**: can disable address whitelisting for withdrawals. The role has the privilege to call the `disableWithdrawalWhitelist()` function in `GlobalConfig`.
14. **withdrawalWhitelistModificationAdmin**: can modify (add or remove) addresses in the withdrawal whitelist. The role has the privilege to call several functions in `GlobalConfig` such as `addAddressToWithdrawalWhitelist()` and `removeAddressFromWithdrawalWhitelist()`.
15. **withdrawalWhitelistAdditionAdmin**: can only add addresses to the withdrawal whitelist. The role has the privilege to call the `addAddressToWithdrawalWhitelist()` function in `GlobalConfig`.
16. **bitcoinKitAdmin**: manages upgrades to the `IBitcoinKit` implementation. The role has the privilege to call several functions in `GlobalConfig` such as `initiateBitcoinKitAddrUpgrade()`, `rejectBitcoinAddrKitUpgrade()`, and `increaseBitcoinKitUpgradeDelay()`.

Findings

HEM-1

Initial Bidders of Partial Collateral Liquidations Always Win and Get Other Bidders' Bids Due to the Usage of `memory` Instead of `storage` Keyword

• High ⓘ Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `d17929d443cf4175f646bc037f96ad1c3671b53a`.
The client provided the following explanation:

```
(Note: this commit was done before addressing feedback in this round, hence commit does not call out HEM-1). There were two additional instances of memory instead of storage being used for PartialLiquidation which were not appropriately fixed previously.
```

File(s) affected: `SimpleBitcoinVaultState.sol`

Description: NOTE: this issue was part of the original audit which was fixed. However, since it was still present at the commit used for this re-audit, we added it here for consistency's sake.

In Solidity, creating a local variable of an item taken from a storage mapping using the updates after the end of the function. For that, the keyword `storage` must be used instead. This mechanism negatively impacts the mechanism of partial collateral liquidations:

- in `beginFullCollateralLiquidation()`, the update of `pl.finished = true` will not persist after the end of the function.
- in `finalizePartialCollateralLiquidation()`, the update of `pl.finished = true` will not persist after the end of the function.
- in `bidOnPartialCollateralLiquidation()`, the three updates of `pl` will not persist after the end of the function:

```
pl.currentBidAmount = newBid;
pl.currentBidTime = block.timestamp;
pl.currentBidder = msg.sender;
```

The last scenario has a high impact because the initial bidder remains the `currentBidder`, the `currentBidTime` remains the initial one so it will not last more than `PARTIAL_LIQUIDATION_BID_TIME` seconds, and the initial bidder will also receive the reimbursement of any future `n+1` bid that would be replaced by another future `n+2` bid.

Recommendation: Consider replacing `PartialLiquidation memory pl =` with `PartialLiquidation storage pl =`.

HEM-2
Inconsistency in the Withdrawals Output's Length Disrupts Proper Functioning of the Vaults

• High ⓘ Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `f2f1d814ddea09258fd9fcb5de3b5ceacda174d9`.
The client provided the following explanation:

```
After adding a required OP_RETURN output to the format of withdrawal BTC transactions, functions in UTXOLogicHelper needed to be updated to account for expected higher tx output counts.
```

File(s) affected: `SimpleBitcoinVaultUTXOLogicHelper.sol`

Description: The functions `checkConfirmedDepositSpendInvalidity()` and `checkSweepSpendInvalidity()` expect a withdrawal to have 2 outputs, otherwise, they mark the TX as invalid.

For example:

```
if (txInputCount == 1) {
    // Must be a withdrawal or invalid

    if (txOutputCount != 2) {
        // Can only traverse backwards through outputs which output a sweep UTXO,
        // meaning they would have exactly two outputs. If not exactly two outputs,
        // then this is not a valid withdrawal transaction and therefore the potential
        // sweep we are analyzing must be invalid.
        return true;
    }
...
```

But the `checkWithdrawalFinalizationValidity()` function expects the withdrawal to have 3 in normal conditions or 2 if a full withdrawal was made:

```
require(btcTx.outputs.length >= 2 && btcTx.outputs.length < 4, "withdrawal transaction must have at least two outputs and no more than three");
```

This incongruency makes it hard for the protocol to function properly since either the withdrawals or the withdrawal challenges won't work. If the withdrawals don't work, funds could get stuck. If the invalidation checks don't work, then users could lose funds.

Recommendation: Confirm the correct number of outputs in a withdrawal and update the corresponding part of the code accordingly.

HEM-3
Challenging a Withdrawal Is Not Possible Putting User Funds at Risk

• High ⓘ Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `99990e5ab732d846b098fd2f89ea15683ca46d35`.
The client provided the following explanation:

```
(Note: this commit was done before addressing feedback in this round, hence commit does not call out HEM-3). A check in challengeWithdrawal() which should have checked that a withdrawal is NOT already
```

```
challenged was instead checking that a withdrawal IS already challenged which was incorrect and was fixed.
```

Description: The function `challengeWithdrawal()` attempts to check that the given withdrawal id was not already challenged. However, it does so incorrectly by missing the `!` operator:

```
require(vaultStateChild.isWithdrawalAlreadyChallenged(uuid), "withdrawal has already been challenged");
```

This affects the correct functioning of a critical part of the system that ensures the proper behaviour of the operator. Without it, funds could get stolen.

Recommendation: We recommend adding tests to cover this issue. The correct check would be:

```
require(!vaultStateChild.isWithdrawalAlreadyChallenged(uuid), "withdrawal has already been challenged");
```

HEM-4

Price Changes Not Accounted for During Full Liquidation Could Lead to Funds Locked

• High ⓘ Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `6a84938a17bcde976bfbf0ab0c5b17e90c0aaf25`.
The client provided the following explanation:

```
The liquidation price for a full vault liquidation was originally set only once based on the current price of collateral at the time the full liquidation was triggered, meaning rapid price changes in either direction would not be accounted for, which could either result in no liquidators being interested in liquidating the vault or in the vault liquidating collateral at a higher price than was required. Fixed by updating the full liquidation price calculation to be based on the price reported by the oracle at the time the liquidation purchase occurs.
```

File(s) affected: `SimpleBitcoinVaultState.sol`

Description: When a full liquidation starts, the liquidation price gets set with a 5% discount and then the discount gets bigger and bigger with time and based on the given ratio. However, the real price of the collateral is no longer taken into account during the liquidation process.

If the real collateral price drops faster than the liquidation price, then users might not want to liquidate, leaving the vault's collateral and hBTC stuck. Another possibility is the operator stealing the BTC since its value would be higher than the collateral.

Recommendation: We recommend including the real collateral as a reference during the liquidation process while maintaining the discount to make it profitable for liquidators.

HEM-5

`increasetotaldepositsheld()` Does Not Correctly Increase the `totalDepositsHeld` in some Cases

• High ⓘ Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `3c3ba0b7b3a7fd85e281544af36a6ece0f3fe8ac`.
The client provided the following explanation:

```
There was a missing else{} branch that meant if a full liquidation was started but no operator reserves existed then the total deposits held would not be incremented correctly, added the appropriate else{} condition in so total deposits are always increased as expected regardless of liquidation status and operator reserves.
```

File(s) affected: `SimpleBitcoinVaultState.sol`

Description: The `increaseTotalDepositsHeld()` function increases the total custodied amount while respecting liquidation processes and operator reserves. However, if a full liquidation has been started, and the operator does not have any reserves, no action is taken and the deposit is not accounted for.

Recommendation: Make sure that the `totalDepositsHeld` are increased correctly by removing the `if (fullLiquidationOperatorReserves > 0)` or adding an else branch with the computation `totalDepositsHeld = totalDepositsHeld + increase`.

HEM-6 Operator Might Not Be Able to Withdraw the Desired and Possible Collateral

• Medium ⓘ Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `4f2caba15965365f152abc8f31d6e28d69a092d6`.
The client provided the following explanation:

```
The getFreeCollateral() function was always calculating free collateral by subtracting out pending
withdrawals, so the pending collateral withdrawal was incorrectly deducted from the free collateral
during finalizePartialCollateralWithdrawal(). To fix, now getFreeCollateral() accepts a boolean for
whether pending collateral withdrawals should be considered in the calculation, and when called by
finalizePartialCollateralWithdrawal() this boolean is set to false, so the free collateral calculation
is done only based on the deposited and utilized collateral.
```

File(s) affected: `SimpleBitcoinVaultState.sol`

Description: When a partial collateral withdrawal is initiated, the variable `pendingCollateralWithdrawal` is set to the withdrawal amount if it is less than the free collateral which is calculated as `uint256 freeCollateral = depositedCollateralBalance - getUtilizedCollateralSoft()`.

However, durring the call to `finalizePartialCollateralWithdrawal()`, the function `getFreeCollateral()` calculates the free collateral as `conservativeTotalCollateral = depositedCollateralBalance - pendingCollateralWithdrawal - getUtilizedCollateralSoft()`. However, since the intention is to withdraw `pendingCollateralWithdrawal` which was already deemed as free during the withdrawal initialization, it should not be used to determine the free collateral.

This leads to the operator not being able to withdraw the desired and possible amount.

Recommendation: We recommend using the same formula as in the withdrawal initialization to use the right collateral balance.

HEM-7 Undesired Behavior During Partial Collateral Withdrawal

• Medium ⓘ Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `1737eeb4598a8a330ec85bb9d8d65ec57a32f32f`.
The client provided the following explanation:

```
A partial collateral withdrawal can only be initiated when a partial collateral withdrawal is not
already in progress, which is determined by checking whether pendingCollateralWithdrawal is equal to
zero. However, finalizePartialCollateralWithdrawal() does not appropriately set this value back to
zero, blocking subsequent initiations of new partial collateral withdrawals. This has been fixed by
setting pendingCollateralWithdrawal and pendingCollateralWithdrawalRequestTime to zero upon successful
finalization of a partial collateral withdrawal.
```

File(s) affected: `SimpleBitcoinVaultState.sol`

Description: When a partial collateral withdrawal is initiated, the variable `pendingCollateralWithdrawal` is set to the withdrawal amount. However, once the withdrawal is finalized, said variable is not reset back zero.

There are two possible issues:

1. Not reseting `pendingCollateralWithdrawal` back to zero will **block** the operator from initiating another partial collateral withdrawal because the contract checks that there are no pending withdrawals in process.
2. The operator could call `finalizePartialCollateralWithdrawal()` multiple times since both the `pendingCollateralWithdrawal` and `pendingCollateralWithdrawalRequestTime` would pass the checks. Fortunately, there is a limit to what the operator could withdraw.

Recommendation: Reset the `pendingCollateralWithdrawal` back to zero once the withdrawal has been finalized. Additionally, reset `pendingCollateralWithdrawalRequestTime` for consistency.

HEM-8 2-Step Admin Transfer Pattern Incorrectly Implemented

• Low ⓘ Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `6293990b66c5f207186301bb66f81f45c00a9404`.
The client provided the following explanation:

```
The 2-step upgrade pattern for the configAdmin in SimpleGlobalVaultConfig was improperly setting the configAdmin upon initialization, which has been resolved by removing this assignment and requiring the finalization of configAdmin upgrade before the configAdmin variable is updated.
```

File(s) affected: `SimpleGlobalVaultConfig.sol`

Description: The function `initiateConfigAdminUpdate()` should only set the `pendingConfigAdmin` but it also sets the `configAdmin` basically bypassing the 2-step pattern.

Recommendation: Do not set the `configAdmin` during the initialization step.

HEM-9 Lack of Access Control in `factory.createVault()` Allows for Phishing Attacks

• Low ⓘ Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `e98c3b779c92bb4a5de515af0b81b1f857389d25`.
The client provided the following explanation:

```
Anyone could deploy a vault through the vault factory. This would not directly affect protocol solvency because that vault would not be recognized by the BitcoinTunnelManager and hence deposits to it would not be credited as hBTC in the system, however it could be used for phishing and confuse users to steal funds if the users did not check the vault's deployment source or availability in BitcoinTunnelManager. This has been resolved by only allowing a permittedVaultCreator to create vaults, which is set as the BitcoinTunnelManager in the GlobalConfig and passed into the vault factory upon activation.
```

Description: Creating a vault through the `BitcoinTunnelManager` ensures proper tracking of vaults, including their count and addresses. However, the `SimpleBitcoinVaultFactory.createVault()` function is publicly accessible, enabling anyone to create a vault that bypasses the intended tracking system. This function also allows the caller to specify the address of the tunnel admin.

A malicious user could exploit this by providing their own implementation of the `BitcoinTunnelManager`, potentially enabling unexpected attacks such as duplicating withdrawal UUIDs. While the exact consequences of such actions are unclear, they introduce unnecessary risks and should be avoided.

Moreover, even if the correct tunnel is provided during vault creation, bypassing the tunnel tracking system could facilitate phishing attacks. A malicious actor could deploy a vault, specify the correct tunnel, and deceive users into depositing BTC into the attacker's Bitcoin address under the guise of a legitimate vault. Since the tunnel cannot track these bypassed vaults, users would be unable to confirm their deposits, as such confirmations require tunnel integration. This would allow attackers to steal BTC without transferring collateral.

Recommendation: We recommend adding a check to `factory.createVault()` that verifies that the caller is the `BitcoinTunnelManager`.

HEM-10 Unnecessary Collateral Loss for the Operator

• Low ⓘ Mitigated

Update
Marked as "Fixed" by the client.
Addressed in: `6a84938a17bcde976bfbf0ab0c5b17e90c0aaf25`.
The client provided the following explanation:

```
This was fixed alongside HEM-4; updated collateral asset price to BTC is fetched whenever the full liquidation price is calculated to prevent the vault liquidating collateral at a higher value than required.
```

Update
Marked as "Mitigated". Even though the function now retrieves the current collateral price, it also limits the final price if it is lower than the starting price (price at the liquidation time 5% discount) to the starting price. This limitation allows liquidators to receive more than they should, and therefore, operators are forced to sell even cheaper than expected.

File(s) affected: `SimpleBitcoinVaultState.sol`

Description: When a full liquidation starts, the liquidation price gets set with a 5% discount and then the discount gets bigger and bigger with time and based on the given ratio. However, the real price of the collateral is no longer taken into account during the liquidation process.

If the real collateral price increases considerably, even making the vault's position no longer liquidatable, the liquidation still takes place and liquidators would take more collateral than they should, leaving the operator with unnecessary collateral loss.

Recommendation: We consider two things could be added to fix the issue:
1. We recommend including the real collateral as a reference during the liquidation process while maintaining the discount to make it profitable for liquidators.
2. Since the price of the collateral could jump back up making the vault's position healthy again, you could consider stopping the liquidation process. A way to avoid jumping back and forth between healthy and unhealthy positions is to use hysteresis thresholding.

HEM-11 Withdraw Fees Can Exceed Minimum Withdraw Amount

• Low ⓘ Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `e892278226774b42ec66efd976e2b39120839e42`.
The client provided the following explanation:

```
An operator could set the minimum deposit fee charged in sats higher than the MINIMUM_WITHDRAWAL_SATS
constant. Fixed by ensuring that any value set for minDepositFeeSats does not exceed this constant,
even if permitted by bounds set in the SimpleBitcoinVaultConfig contract.
```

File(s) affected: `SimpleBitcoinVaultState.sol`

Description: The withdrawal fee can be set by the operator within the bounds defined in the `SimpleBitcoinVaultConfig` contract. Additionally, the `SimpleBitcoinVault` defines `MINIMUM_WITHDRAWAL_SATS` as the smallest amount that can be withdrawn. If the `maxWithdrawalFeeAdmin` changes the maximum allowed fee to be higher than the minimum withdrawal amount, and this change is adopted by the operator, this can break the wind-down procedure of the vault which ensures that another withdrawal is possible. This also applies to deposit fees.

Recommendation: Ensure that `SimpleBitcoinVaultState.sol` cannot set the `minDepositFeeSats` higher than `MINIMUM_WITHDRAWAL_SATS`.

HEM-12 Insufficient `BTC` Security Confirmations

• Informational ⓘ Acknowledged

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

```
hVM functions by lagging the indexer 2 blocks behind the current consensus tip, but that current
consensus tip is based on all available public Bitcoin consensus information. As a result, an hVM BTC
confirmations of 4 is (at a minimum) equivalent to a normal BTC confirmations of 6, because  achieving
4 BTC blocks of confirmation in hVM requires that the lightweight BTC consensus tracked by the Hemi
protocol has determined that the canonical chain tip  is 6 BTC blocks ahead of the confirmed
transaction with 4 confirmations. If a reorg were to occur when the transaction had effectively 5
confirmations (3 in hVM), then that reorg would be recognized by the Hemi protocol before the protocol
advanced onto the BTC block which gives the transaction 4 confirmations in hVM, unless that reorg
occurred after that 6th BTC block was communicated to the Bitcoin P2P network, which is the same
failure scenario as a 6-block reorg with standard confirmations. No changes made, but may consider
bumping up confirmations to a higher value for more than effectively 6 blocks of BTC security given the
importance of protocol solvency.
```

File(s) affected: `SimpleBitcoinVault.sol`

Description: The vault requires `MIN_BITCOIN_CONFIRMATIONS` of 4, with the motivation that the hVM on-chain is 2 blocks behind the actual Bitcoin ledger resulting in 6 blocks in total. While this ensures that 6 blocks have passed, this does not ensure 6 blocks of security. For this purpose all 6 blocks need to be checked, since otherwise the ledger could have adopted a different longest-chain consensus without the relevant transaction.

Recommendation: Decide and enforce the desired security level on Hemi. We recommend at least 6 confirmations, some sources recommend more for higher-value transactions.

HEM-13 `initiatewithdrawal()` Always Returns Fee as Zero

• Informational ⓘ Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `bb439beeaa3532745437bed9dee5c63061364416`.
The client provided the following explanation:

```
initiateWithdrawal() was not properly returning the feeSats returned by vault.initiateWithdrawal().
This does not effect any functionality but this data should be returned to the caller for their own
information. Fixed by correctly returning the fee returned from vault.initiateWithdrawal().
```

File(s) affected: `BitcoinTunnelManager.sol`

Description: The return value `feeSats` of the `initiateWithdrawal()` function is never updated and thus always zero.

Recommendation: Return `fee` instead or update the variable.

HEM-14 Pending Unminted Fees Are Collateralized

• Informational ⓘ Acknowledged

Update

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

```
This is the intended behavior; when confirming a deposit the vault wants to make sure that confirming
the deposit does not exceed the soft collateralization threshold assuming the fees would be claimed
immediately after, but we do not want to block minting of operator fees afterwards if subsequent
collateral value changes would later result in the operator fee minting violating the soft
collateralization threshold.
```

File(s) affected: `SimpleBitcoinVault.sol`

Description: The `confirmDeposit()` function checks whether the supplied deposit is collateralized. The security only extends to custodied BTC, so per documentation, the pending fee on BTC should not be considered, only after being minted into `hBTC`. However, the check below extends to the entire deposit.

```
require(!vaultStateChild.doesDepositExceedSoftCollateralThreshold(totalDepositSats), "deposit would
exceed soft collateralization threshold");
```

The internal accounting correctly increases the net deposits. Similarly, in `mintOperatorFees()`, the accounting correctly adds the minted fees to the net deposits, but foregoes the collateralization check.

Recommendation: Decide on the intended behavior and correct the code and documentation accordingly.

Auditor Suggestions

S1 Documentation Discrepancies

Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `f2f42f3aba7a7cd4e86f47676a161cef55dc1d71`.
The client provided the following explanation:

```
(Note: this commit was done before addressing feedback in this round, hence the commit includes other
items and does not callout S1 by name): The new vault deployments were not being properly saved by the
factory. Updated to properly save them with appropriate permissions to prevent others from adding
vaults that are not part of the system.
```

File(s) affected: `SimpleGlobalVaultConfig.sol`

Description: 1. The comment above `saveNewVaultDeployment()` states that the `SimpleBitcoinVaultFactory` should call it after a new vault deployment, however, the factory does not call it, and the implementation seems to expect `vaultDeploymentAdmin` to be the caller. If this is not called by anyone, consider removing it along with the `deployedVaults` mapping.

2. `depositCollateral()` contradicts documentation: Initially, a minimum collateral amount is needed for the vault to go live. According to the documentation, this can be achieved via separate supply transactions, however the code below forces the first collateral deposit to be larger than the currently required minimum collateral amount. This is contradicted by the natspec comment of the function as well as other comments within that function.

```
if (vaultStatus == Status.CREATED) {
    require(amount > vaultConfig.getMinCollateralAssetAmount(),
    "when depositing collateral into a new vault, the initial deposit must be at least the minimum
collateral asset amount");
}
```

Recommendation: Consider the recommendations above.

S2 Critical Role Transfer Not Following Two-Step Pattern

Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `f2f42f3aba7a7cd4e86f47676a161cef55dc1d71`.
The client provided the following explanation:

```
(Note: this commit was done before addressing feedback in this round, hence the commit includes other
items and does not callout S2 by name): The upgrade to the admin in SimpleBitcoinVault was not
following a 2-step pattern which has been added in.
```

File(s) affected: `SimpleBitcoinVault.sol`

Description: The `operatorAdmin` can transfer the ownership to a new address. If an uncontrollable address is accidentally provided as the new operator admin, it could lead to unauthorized modifications to the protocol.

Recommendation: We recommend updating the `operatorAdmin` using the 2-step pattern.

S3 General Suggestions

Acknowledged

Update

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

```
Will perform code cleanup and incorporate these suggestions shortly.
```

Description: This is a non-exhaustive list of general recommendations we would like to suggest:

1. Consider using custom errors instead of strings for more efficiency. See `AddressWhitelist`, `GlobalConfig`.
2. If a variable is set at construction time only, consider declaring the variable as `immutable` which would make the intent clearer and save on gas consumption. The following is a non-exhaustive list of variables that can be set `immutable`:
    1. `GlobalConfig.minimumVaultFactoryUpgradeDelay`.
    2. In `SimpleGlobalVaultConfig`: `softCollateralizationThreshold`, `hardCollateralizationThreshold`, `vaultDeprecationAdmin`, `vaultDeploymentAdmin` and `permittedCollateralAssetContract`.
3. Avoid using magic numbers, they are error prone and reduce code readability. Instead, create const variables with a describing name that conveys the idea behind it.
4. Remove dead or unused code since it bloats the codebase and could increase deployment costs. In `SimpleBitcoinVault`: the event `CollateralWithdrawn` and the variable `MAX_WITHDRAWAL_QUEUE_SIZE` are not used.
5. The function `initiateWithdrawal()` checks `require(amountSats <= getNetDeposits(), "withdrawal exceeds net deposits")`. However, it unnecessarily checks it again by doing `require(pendingWithdrawalAmount + amountSats <= depositsHeld, "cannot withdraw more sats than vault holds")`.
6. The Solidity compiler automatically creates getters for public variables. Additional getters like `SimpleGlobalVaultConfig.getPriceOracle()` only increase the contract size. Consider removing them wherever possible.

Recommendation: Consider the suggestions above.

S4 Unlocked Pragma

Acknowledged

Update

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

```
Will update this as part of future code cleanup.
```

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.*`. The caret (`^`) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend to remove the caret to lock the file onto a specific Solidity version.

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).

Test Suite Results

To run the tests: `npm install && npx hardhat test`

```
Warning: Contract code size is 24766 bytes and exceeds 24576 bytes (a limit introduced in Spurious
Dragon). This contract may not be deployable on Mainnet. Consider enabling the optimizer (with a low
"runs" value!), turning off revert strings, or using libraries.
   --> contracts/vaults/SimpleBitcoinVault/SimpleBitcoinVault.sol:292:1:
    |
292 | contract SimpleBitcoinVault is IBitcoinVault, VaultUtils, SimpleBitcoinVaultStructs,
ReentrancyGuard {
    | ^ (Relevant source part starts here and spans across multiple lines).


Warning: Contract code size is 25712 bytes and exceeds 24576 bytes (a limit introduced in Spurious
Dragon). This contract may not be deployable on Mainnet. Consider enabling the optimizer (with a low
"runs" value!), turning off revert strings, or using libraries.
 --> contracts/vaults/SimpleBitcoinVault/SimpleBitcoinVaultFactoryHelper.sol:7:1:
  |
7 | contract SimpleBitcoinVaultFactoryHelper is ISimpleBitcoinVaultFactoryHelper {
  | ^ (Relevant source part starts here and spans across multiple lines).


Compiled 39 Solidity files successfully (evm target: cancun).


  AddressWhitelist
    Deployment
      ✔ Should set the right owner (9831ms)
      ✔ No address should be whitelisted
      ✔ Should fail if owner is zero address (51ms)
    Whitelisting
```

```
    Whitelist Additions
      ✔ Owner adding a regular address to whitelist should be accepted
      ✔ Owner adding a zero address to whitelist should be rejected (51ms)
      ✔ Address other than owner adding a regular address to whitelist should be rejected
      ✔ Any address should be able to check if an address is whitelisted
    Whitelist Removals
      ✔ Owner removing a regular address from whitelist should be removed (52ms)
      ✔ Address other than owner should not be able to remove a whitelisted address

  GlobalConfig
    Deployment
      ✔ Should set the right initial admins (172ms)
      ✔ Should set the right bitcoin kit address
      ✔ Should set the right upgrade delays
    Vault Factory Modifications
      ✔ Should allow global config admin to set initial vault factory (53ms)
      ✔ Should allow global config admin to delegate setting of initial vault factory to vault factory
upgrade admin (67ms)
      ✔ Should not allow address that is not global config admin or vault factory upgrade admin to set
initial vault factory (69ms)
      ✔ Should not allow global config admin to set initial vault factory twice (69ms)
      ✔ Should check that vault factory's status admin is set correctly (42ms)
      ✔ Should check that vault factory's children count is zero (51ms)
      ✔ Should check that vault factory is not active (48ms)
      ✔ Should check that vault factory is not deprecated (44ms)
      ✔ Should prevent an unpermitted address from initiating a vault factory upgrade (152ms)
      ✔ Should allow global config admin to initiate a vault factory upgrade (58ms)
      ✔ Should allow global config admin to set vault upgrade admin who can initiate a vault factory
upgrade (57ms)
      ✔ Should allow global config admin to reject a pending vault factory upgrade (168ms)
      ✔ Should allow vault upgrade admin admin to reject a pending vault factory upgrade
      ✔ Should prevent an unpermitted address from rejecting a pending vault factory upgrade (43ms)
      ✔ Should allow a second vault factory upgrade to overwrite an existing pending vault factory
upgrade (46ms)
      ✔ Should reject a vault factory upgrade finalization if not enough time has elapsed (88ms)
      ✔ Should allow anyone to finalize the vault upgrade if enough time has elapsed
      ✔ Should allow vault factory upgrade bypass admin to bypass the vault factory upgrade delay
      ✔ Should allow global config admin to bypass the vault factory upgrade delay
      ✔ Lowering vault factory upgrade delay should not apply to a vault upgrade already in progress
(87ms)

  SimpleBitcoinVaultState
    Deployment
      ✔ Should set operator admin correctly (215ms)
      ✔ Should set parent simple bitcoin vault correctly
      ✔ Should set soft collateralization threshold correctly
      ✔ Should not have pending soft collateralization threshold
      ✔ Should not have pending soft collateralization threshold update time
      ✔ Should set hard collateralization threshold correctly
      ✔ Should set min deposit fee sats correctly
      ✔ Should not have pending min deposit fee sats
      ✔ Should not have pending min deposit fee sats update time
      ✔ Should set deposit fee bps correctly
      ✔ Should not have pending deposit fee bps
      ✔ Should not have pending deposit fee bps update time
      ✔ Should set min withdrawal fee sats correctly
      ✔ Should not have pending min withdrawal fee sats
      ✔ Should not have pending min withdrawal fee sats update time
      ✔ Should set withdrawal fee bps correctly
      ✔ Should not have pending withdrawal fee bps
      ✔ Should not have pending withdrawal fee bps update time
    Fee Calculation
      ✔ Should calculate fee on small deposit to be minimum sats fee
      ✔ Should calculate fee on large deposit to be correct based on bps
      ✔ Should calculate fee on small withdrawal to be minimum sats fee
      ✔ Should calculate fee on large withdrawal to be correct based on bps
    Vault Administration
      ✔ Should update operator admin from parent vault correctly
      ✔ Should not allow an account other than operator admin to change soft collateralization threshold
      ✔ Should not allow operator admin to set soft collateralization threshold lower than threshold set
in shared config
      ✔ Should not allow operator admin to set soft collateralization threshold that is same as current
```

```
      ✔ Should allow operator to raise and lower soft collateralization threshold immediately if vault is not live
      ✔ Should allow operator to immediately set a lower soft collateralization update when vault is live
      ✔ Should allow operator to initiate a pending higher soft collateralization update when vault is live
      ✔ Should allow operator to initiate a pending higher soft collateralization update then replace with lower update (49ms)
      ✔ Should reject operator setting min deposit fee sats lower than minimum in config
      ✔ Should reject operator setting min deposit fee sats higher than maximum in config
      ✔ Should reject operator setting min deposit fee sats to its existing value
      ✔ Should reject non-operator attempting to change min deposit fee sats
      ✔ Should allow operator to immediately increase min deposit fee sats before vault is live
      ✔ Should allow operator to immediately decrease min deposit fee sats before vault is live
      ✔ Should allow operator to immediately decrease min deposit fee sats when vault is live
      ✔ Should allow operator to increase min deposit fee sats after vault is live only after delay (42ms)
      ✔ Should reject min deposit fee sats finalization if no update is in progress
      ✔ Should reject operator setting min withdrawal fee sats lower than minimum in config (41ms)
      ✔ Should reject operator setting min withdrawal fee sats higher than maximum in config
      ✔ Should reject operator setting min withdrawal fee sats to its existing value
      ✔ Should reject non-operator attempting to change min withdrawal fee sats
      ✔ Should allow operator to immediately increase min withdrawal fee sats before vault is live
      ✔ Should allow operator to immediately decrease min withdrawal fee sats before vault is live
      ✔ Should allow operator to immediately decrease min withdrawal fee sats when vault is live
      ✔ Should allow operator to increase min withdrawal fee sats after vault is live only after delay (40ms)
      ✔ Should reject min withdrawal fee sats finalization if no update is in progress
      ✔ Should reject operator setting deposit fee bps lower than minimum in config
      ✔ Should reject operator setting deposit fee bps higher than maximum in config
      ✔ Should reject operator setting deposit fee bps to its existing value
      ✔ Should reject non-operator attempting to change deposit fee bps
      ✔ Should allow operator to immediately increase deposit fee bps before vault is live
      ✔ Should allow operator to immediately decrease deposit fee bps before vault is live
      ✔ Should allow operator to immediately decrease deposit fee bps when vault is live
      ✔ Should allow operator to increase deposit fee bps after vault is live only after delay (38ms)
      ✔ Should reject deposit fee bps finalization if no update is in progress
      ✔ Should reject operator setting withdrawal fee bps lower than minimum in config
      ✔ Should reject operator setting withdrawal fee bps higher than maximum in config
      ✔ Should reject operator setting withdrawal fee bps to its existing value
      ✔ Should reject non-operator attempting to change withdrawal fee bps
      ✔ Should allow operator to immediately increase withdrawal fee bps before vault is live
      ✔ Should allow operator to immediately decrease withdrawal fee bps before vault is live
      ✔ Should allow operator to immediately decrease withdrawal fee bps when vault is live
      ✔ Should allow operator to increase withdrawal fee bps after vault is live only after delay (38ms)
    Pending Withdrawal Storage
      ✔ Should save single withdrawal correctly
      ✔ Should save multiple withdrawals correctly
      ✔ Should allow fulfillment of single withdrawal correctly (48ms)
      ✔ Should allow fulfillment of multiple withdrawals correctly (192ms)

  SimpleBitcoinVaultUTXOLogicHelper
    Deployment
      ✔ Should expose MAX_SWEEP_UTXO_WALKBACK (39ms)
    Extract Withdrawal Index From OP_RETURN
      ✔ Should extract indexes correctly from 6-byte OP_RETURN (150ms)
      ✔ Should extract indexes correctly from 7-byte OP_RETURN (148ms)
      ✔ Should extract indexes correctly from 75-byte OP_RETURN (47ms)
      ✔ Should extract indexes correctly from 76-byte OP_RETURN (48ms)
    Check Deposit Confirmation Validity
      ✔ Should reject deposit that is already acknowledged (109ms)
      ✔ Should reject deposit with a claimed output index >= 8
      ✔ Should reject deposit where bitcoin transaction does not have enough outputs for claimed output index (174ms)
      ✔ Should reject deposit without an op_return containing an extractable evm address (110ms)
      ✔ Should reject deposit where claimed deposit utxo does not deposit funds to the vault custodianship script hash (98ms)
      ✔ Should reject deposit where the deposited sats are lower than the minimum deposit sat amount (107ms)
      ✔ Should reject deposit where the calculated deposit fee is higher than the deposited sats (112ms)
      ✔ Should accept 1-input 2-output deposit with address bytes in output 0 and deposit in output 1
      ✔ Should accept 1-input 2-output deposit with address lower-case hex in output 0 and deposit in output 1
```

```
      ✔ Should accept 1-input 2-output deposit with address mixed-case hex in output 0 and deposit in
output 1
      ✔ Should accept 1-input 2-output deposit with deposit in output 0 and address bytes in output 1
      ✔ Should accept 1-input 2-output deposit with deposit in output 0 and address lower-case hex in
output 1
      ✔ Should accept 1-input 2-output deposit with deposit in output 0 and address mixed-case hex in
output 1
      ✔ Should accept 2-input 2-output deposit with deposit in output 0 and address bytes in output 1
      ✔ Should accept 2-input 2-output deposit with address lower-case hex in output 0 and deposit in
output 1
      ✔ Should accept 2-input 2-output deposit with deposit in output 0 and address mixed-case hex in
output 1
      ✔ Should accept 4-input 7-output deposit with deposit in output 3 and address bytes in output 5
      ✔ Should accept 4-input 7-output deposit with address bytes in output 3 and deposit in output 5
      ✔ Should accept 9-input 8-output deposit with address bytes in output 6 and deposit in output 7
      ✔ Should accept 9-input 8-output deposit with deposit in output 6 and address bytes in output 7
      ✔ Should accept 9-input 10-output deposit where only 8 outputs are given with deposit in output 6
and address bytes in output 7
    Check Withdrawal Finalization Validity
      ✔ Should reject withdrawal fulfillment for btc tx that is not available in hVM yet
      ✔ Should reject withdrawal fulfillment for btc tx that has more than one input (106ms)
      ✔ Should reject withdrawal fulfillment for btc tx that does not spend current sweep utxo
      ✔ Should reject withdrawal fulfillment for btc tx that has no outputs
      ✔ Should reject withdrawal fulfillment for btc tx that has more than three outputs (99ms)
      ✔ Should reject withdrawal fulfillment for withdrawal that does not exist (46ms)
      ✔ Should reject withdrawal fulfillment for btc tx that does not send to the correct withdrawal
script in output 0 (73ms)
      ✔ Should reject withdrawal fulfillment for btc tx that does not send the correct amount (53ms)
      ✔ Should reject withdrawal fulfillment for btc tx that outputs change to the wrong custodian
      ✔ Should accept withdrawal fulfillment for valid btc withdrawal tx with change that paid btc fees
lower than withdrawal fees collected
      ✔ Should accept withdrawal fulfillment for valid btc withdrawal tx with change that paid btc fees
exactly equal to withdrawal fees collected
      ✔ Should accept withdrawal fulfillment for valid btc withdrawal tx with change that paid btc fees
higher than withdrawal fees collected
      ✔ Should accept withdrawal fulfillment for valid btc withdrawal tx without change that paid btc
fees lower than withdrawal fees collected
      ✔ Should accept withdrawal fulfillment for valid btc withdrawal tx without change that paid btc
fees exactly equal to withdrawal fees collected
      ✔ Should accept withdrawal fulfillment for valid btc withdrawal tx without change that paid btc
fees higher than withdrawal fees collected
    Check Sweep Validity
      ✔ Should reject empty sweep txid
      ✔ Should reject missing sweep transaction
      ✔ Should reject sweep transaction with one input
      ✔ Should reject sweep transaction with more than 8 inputs
      ✔ Should reject sweep transaction with more than 1 output
      ✔ Should reject sweep transaction that doesn't match current sweep utxo
      ✔ Should reject sweep with output to script that does not match custodian
      ✔ Should reject sweep with sweep input that is not an acknowledged deposit
      ✔ Should reject sweep with sweep input that does not match acknowledged deposit output index
      ✔ Should calculate return values correctly with 1 input where all collectable fees are spent on btc
tx fee
      ✔ Should calculate return values correctly with 1 input where operator collects nonzero fees
      ✔ Should calculate return values correctly with 1 input where more fees paid on btc fees than
collected
      ✔ Should calculate return values correctly with 1 input where entire deposit input is spent on btc
tx fees
      ✔ Should calculate return values correctly with 2 inputs where all collectable fees are spent on
btc tx fee
      ✔ Should calculate return values correctly with 2 inputs where operator collects nonzero fees
      ✔ Should calculate return values correctly with 2 inputs where more fees paid on btc than collected
      ✔ Should calculate return values correctly with 7 inputs where all collectable fees are spent on
btc tx fee (67ms)
      ✔ Should calculate return values correctly with 7 inputs where operator collects nonzero fees
(67ms)
      ✔ Should calculate return values correctly with 7 inputs where more fees paid on btc than collected
(66ms)
    Check Confirmed Deposit Spend Invalidity
      ✔ Should reject invalid spend claim of missing current sweep utxo spend transaction
      ✔ Should reject invalid spend claim where identified input index does not exist
      ✔ Should reject invalid spend claim where identified input is not a confirmed deposit
```

```
      ✔ Should reject invalid spend claim where identified input is a confirmed deposit but wrong index
is spent
      ✔ Should reject invalid spend claim where transaction has more than 8 inputs available in tx list
but input is not a confirmed deposit
      ✔ Should reject invalid spend claim where transaction has more than 8 inputs available in tx list
but input is a confirmed deposit but wrong index is spent
      ✔ Should reject invalid spend claim where transaction has more than 1 output available in tx list
but input is not a confirmed deposit
      ✔ Should reject invalid spend claim where transaction has more than 1 output with only 1 in tx list
and input is a confirmed deposit but wrong index is spent
      ✔ Should accept invalid spend claim where transaction has more than 8 inputs available in tx list
and confirmed deposit is spent
      ✔ Should accept invalid spend claim where transaction has more than 8 inputs with only 8 in tx list
and confirmed deposit at index 1 is spent (55ms)
      ✔ Should accept invalid spend claim where transaction has more than 8 inputs with only 8 in default
tx list but targeted confirmed deposit is 9th input (52ms)
      ✔ Should accept invalid spend claim where transaction has more than 1 output available in tx list
and confirmed deposit is spent
      ✔ Should accept invalid spend claim where transaction has more than 1 output with only 1 in tx list
and confirmed deposit is spent
      ✔ Should accept invalid spend claim where transaction spends a confirmed deposit as first and only
input
      ✔ Should accept invalid spend claim where transaction spends a confirmed deposit after sweep utxo
input but outputs to a different script than the custodian
      ✔ Should accept invalid spend claim where transaction spends a confirmed deposit at index 1
      ✔ Should accept invalid spend claim where transaction spends a confirmed deposit at index 8 (48ms)
      ✔ Should accept invalid spend claim where transaction spends an input which is not a confirmed
deposit but a valid confirmed deposit spend is blamed
      ✔ Should accept invalid spend claim where transaction spends an input which spends a confirmed
deposit txid but uses the wrong index and a valid confirmed deposit spend is blamed
      ✔ Should reject invalid spend claim where all self-contained validity checks pass and input index 0
is the current sweep utxo
      ✔ Should accept invalid spend claim where all self-contained validity checks pass but input index 0
spends the sweep utxo with the wrong source index
      ✔ Should accept invalid spend claim where all self-contained validity checks pass but input 0 is
from a transaction with 1 input but only 1 output so cannot be valid withdrawal (or sweep)
      ✔ Should accept invalid spend claim where all self-contained validity checks pass but input 0 is
from a transaction with 1 input but 3 outputs so cannot be valid withdrawal (or sweep)
      ✔ Should accept invalid spend claim where all self-contained validity checks pass but input 0 is
from a transaction with withdrawal geometry but input index is 0 which isn't withdrawal sweep output
index
      ✔ Should accept invalid spend claim where all self-contained validity checks pass but input 0 is
from a transaction with more than 8 inputs
      ✔ Should accept invalid spend claim where all self-contained validity checks pass but input 0 is
from a transaction with more than 1 input and more than 1 output
      ✔ Should reject invalid spend claim where all self-contained validity checks pass and input 0 is
from a potentially valid withdrawal transaction
      ✔ Should reject invalid spend claim where all self-contained validity checks pass and input 0 is
from a potentially valid sweep transaction


  189 passing (16s)
```

Code Coverage

The test coverage is low at the moment. We recommend adding tests ensuring a branch coverage greater than 90%. Specifically, important contracts like: `SimpleBitcoinVault`, `SimpleBitcoinVaultState`, and `SimpleBitcoinVaultUTXOLogicHelper` should be thoroughly tested.

To run the coverage, we removed the optimizer setting in the `hardhat.config.js` and turned on the `viaIR: true` flag to avoid getting the stack-too-deep error.

The coverage is run by: `npx hardhat coverage`.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	1.67	1.67	7.69	2.74	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
BTCToken.sol	16.67	16.67	25	20	... 32,33,36,41
BitcoinTunnelManager.sol	0	0	0	0	... 380,382,383
contracts/bitcoinkit/	0	0	0	0	
BitcoinKit.sol	0	0	0	0	... 392,393,395
BitcoinKitStructs.sol	100	100	100	100	
IBitcoinKit.sol	100	100	100	100	
Utils.sol	0	0	0	0	... 27,28,29,31
contracts/governance/	28.68	19.92	31.75	36.09	
AddressWhitelist.sol	100	90	100	100	
GlobalConfig.sol	27.07	17.19	24.56	33.78	... 8,1169,1171
contracts/oracles/	0	0	16.67	33.33	
DummyPriceOracle.sol	0	0	16.67	33.33	... 69,70,77,78
IAssetPriceOracle.sol	100	100	100	100	
contracts/test/	29.56	11.43	40.43	33.72	
MockBitcoinKit.sol	45.45	27.78	30.77	54.26	... 262,268,273
MockBitcoinVault.sol	0	0	3.13	0	... 337,338,342
MockCollateralToken.sol	0	100	33.33	0	13,18
MockSimpleBitcoinVault.sol	100	50	100	100	
MockSimpleBitcoinVaultState.sol	55.56	100	78.57	62.5	... 75,76,80,81
MockVaultFactory.sol	78.57	33.33	81.82	85.71	60,78,79
TestUtils.sol	100	100	100	100	
contracts/vaults/	92.31	83.33	100	94.12	
CommonStructs.sol	100	100	100	100	
IBitcoinVault.sol	100	100	100	100	
IGlobalVaultConfig.sol	100	100	100	100	
IVaultFactory.sol	100	100	100	100	
VaultUtils.sol	92.31	83.33	100	94.12	34

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/vaults/SimpleBitcoinVault/	33.88	28.18	30.12	37.27	
ISimpleBitcoinVaultFactoryHelper.sol	100	100	100	100	
ISimpleBitcoinVaultStateFactory.sol	100	100	100	100	
SimpleBitcoinVault.sol	0	0	0	0	... 9,1491,1492
SimpleBitcoinVaultFactory.sol	0	0	0	0	... 270,279,289
SimpleBitcoinVaultFactoryHelper.sol	0	100	0	0	9
SimpleBitcoinVaultState.sol	37.1	36.88	41.54	43.42	... 5,1813,1821
SimpleBitcoinVaultStateFactory.sol	0	100	0	0	9
SimpleBitcoinVaultStructs.sol	100	100	100	100	
SimpleBitcoinVaultUTXOLogicHelper.sol	73.43	72.39	83.33	77.06	... 747,756,762
SimpleGlobalVaultConfig.sol	49.18	20.16	43.9	53.04	... 611,620,647
All files	28.23	23.37	31.23	31.16	

Changelog

- 2024-12-20 - Initial report
- 2025-01-21 - 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 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 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 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 and may not be represented as such. No third party is entitled to rely on the report in any any way, including for the purpose of making any decisions to buy or sell a product, 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 or any open source or third-party software, code, libraries, materials, or information to, to, called by, referenced by or accessible through the report, its content, or any related 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 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