Quantstamp Security Assessment Certificate

August 31st 2021 — Quantstamp Verified

Badger ibBTC

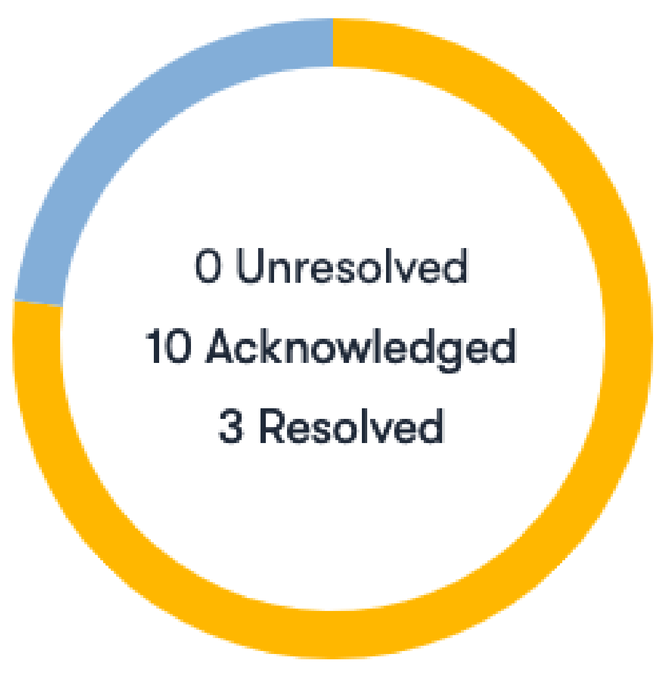
This audit report was prepared by Quantstamp, the leader in blockchain security.

Executive Summary

Type	DeFi
Auditors	Jake Goh Si Yuan, Senior Security Researcher Mohsen Ahmadvand, Senior Research Engineer
Timeline	2021-06-08 through 2021-08-05
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Documentation Quality	Low
Test Quality	Low
Source Code	

Repository	Commit
ibBTC	6860dd8
ibBTC Reaudit	3cad810

Total Issues	13	(3 Resolved)
High Risk Issues	1	(0 Resolved)
Medium Risk Issues	5	(2 Resolved)
Low Risk Issues	2	(1 Resolved)
Informational Risk Issues	3	(0 Resolved)
Undetermined Risk Issues	2	(0 Resolved)



High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	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.

Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
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).
Resolved	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

We have performed an audit and discovered 13 issues, ranging from High to Undetermined. The distribution of severity is detailed both above and below this text. In this audit, we found that the implicit trust towards external contracts to be problematic, and recommend that the ibBTC team consider the security model of "trust, but verify" whenever handling any external interactions. We have also found the documentation to be imprecise and lacking in some areas, which we have noted in one of the issues. There was also a lack of tests for the contracts, and we heavily recommend raising the coverage level to the minimum of 100%. We urge the ibBTC team to strongly consider the issues and the recommendations, and make the appropriate fixes and/or official acknowledgements about it, in a speedy manner, especially as it seems that the system has already been launched before the audit has taken place.

ID	Description	Severity	Status
QSP-1	Implicit unbounded trust in peaks	High	Acknowledged
QSP-2	Unbounded `mint`ing when `guestList` is not set	Medium	Acknowledged
QSP-3	Peaks may be duplicated and cause `totalSystemAssets` to miscount	Medium	Fixed
QSP-4	Funds may be locked by peak extinction	Medium	Mitigated
QSP-5	`Core` may never be switched in current implementation	Medium	Acknowledged
QSP-6	Lack of validation for arbitrary input `token` may lead to stolen token credit	Medium	Acknowledged
QSP-7	Setter functions may set to same variables	Low	Acknowledged
QSP-8	Potential precision loss in division	Low	Fixed
QSP-9	Privileged Roles and Ownership	Informational	Acknowledged
QSP-10	Misleading usage of scaled, and under-informative function names	Informational	Acknowledged
QSP-11	Magic numbers	Informational	Acknowledged
QSP-12	Dormant peaks may `redeem`	Undetermined	Acknowledged
QSP-13	Cross peak withdrawal is allowed	Undetermined	Acknowledged

Quantstamp Audit Breakdown

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

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

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
    i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    iii. 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:
    i. 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.
    ii. 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, clarify, 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.

Toolset

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

Setup

Tool Setup:

- Slither v0.8.0

Steps taken to run the tools:

> Installed the Slither tool: `pip install slither-analyzer` Run Slither from the project directory: `slither .`

Findings

QSP-1 Implicit unbounded trust in peaks

Severity: High Risk

Status: Acknowledged

File(s) affected: `Core.sol`

Description: The current flow in the essential external functions of `mint` and `redeem` works by having most of the critical operations being performed and started in the peak contracts. The operations include the storing and collection/return of external tokens, retrieval of exchange rates, calculation of the actual exchange, and the emitting of event. The interaction with the `Core` contract comes in the form of the peak contracts telling it how many `bBTC` to `mint` or `redeem`, and the former following it without limits or any form of verification. Note also that the peak contracts may also `redeem` and therefore `burn` for any arbitrary account, even those that are not associated with it.
This is a security model that relies on the peak contracts to be secure and correct. Given the definition of a peak in the `README.md` to be `any third party integration in the protocol`, we find that this security model is too trusting, and any malicious peak may either ruin the economics through excessive minting or loss of reputation from arbitrary `burn`ing.

Recommendation: There should be an accounting system within the `Core` contract that retains information about the peak and associated users, that can be verified internally from the `Core` perspective. Likewise, there should be a governance set limit to how many `bBTC` each peak contract is able to mint at any time.
There should also be a consideration to whether it might be more secure to have the collection and verification of funds done at the `Core` level.

Update: From the team: "There can't be malicious peaks. Every peak is a trusted contract; not a random 3rd party integration that anyone can plug into the system. It's as simple as breaking the core logic of the system into several contracts instead of having a huge single contract."

QSP-2 Unbounded `mint`ing when `guestList` is not set

Severity: Medium Risk

Status: Acknowledged

File(s) affected: `Core.sol`

Description: The `mint` function only validates for whether or not to `mint` to a particular `account` if `address(guestList) != address(0)`. The other access control present is the check for whether `peaks[msg.sender] == PeakState.Active`. This means that if `address(guestList) == address(0)`, an active peak can mint any arbitrary amounts of `bBTC`. Given that peaks are considered to be third party integrations, caution should be given to unbounded powers like this.

Recommendation: Either create an alternate access control method when `address(guestList) != address(0)` and/or cap the total amount that can be minted by a peak.

Update: From the team: "This is the intended behaviour"

QSP-3 Peaks may be duplicated and cause `totalSystemAssets` to miscount

Severity: Medium Risk

Status: Fixed

Description: Peaks are added to the `Core` contract by `whitelistPeak`, which first validates for whether `peaks[peak] == PeakState.Extinct` L155, before adding it to `peakAddresses`, setting the status at `peaks[peak]` to `PeakState.Active` and firing an event.
The status of peaks can be adjusted at `setPeakStatus`, which allows for any arbitrary state to be set as long as the current status is not `PeakState.Extinct`. This means that it is possible to have whitelisted a peak, then set its status to `PeakState.Extinct` after, then whitelist the same peak again, ad infinitum. Given that the peak address is not removed from `peakAddresses` when the peak is set to `PeakState.Extinct`, nor does it perform an existence check when adding it, this means that duplicates can happen. This means that `totalSystemAssets`, which relies on iterating through all `peakAddresses`, will count the duplicates and produce a miscounted value.

Recommendation: Do not allow peaks to be set to `PeakState.Extinct` and allow for deactivation of peaks to be set to `PeakState.Dormant` instead. Otherwise, remove peak address from `peakAddresses` when set to `PeakState.Extinct`

Update: Check added to assert that peak is not already whitelisted in order to avoid duplicates.

QSP-4 Funds may be locked by peak extinction

Severity: Medium Risk

Status: Mitigated

File(s) affected: `Core.sol`

Description: The `redeem` function prevents extinct peaks from utilizing it. The governor may set a peak to extinct status without ensuring that all of the associated funds have already been `redeem`ed. This has the unfortunate effect of causing accounting issues as the funds will also be removed from the `totalSystemAssets` calculation, which is essential for a lot of important calculations. Whilst it may be possible to `whitelistPeak` again to bring it back to a non extinct status so that the funds can be retrieved, it will result in the other issue (QSP-3) being activated.

Recommendation: There are a couple of ways this can be resolved, depending on what the actual definition of the extinct status is, which is not too clear at this point of the audit as there is no specification about it. The requirement can be removed from `redeem`, or setting a peak status to extinct could require that there are no existing funds still `redeem`able, or one can simply remove the ability for peaks to be set to extinct.

Update: The team added a validation that checked for whether the amount remaining is less than `1e15`, or "near-0" as described.

QSP-5 `Core` may never be switched in current implementation

Severity: Medium Risk

Status: Acknowledged

File(s) affected: `bBTC.sol`, `Core.sol`

Description: In the `bBTC` contract, there is a method `switchCore` which is designed to be used to upgrade and switch `Core` contracts. This method is protected by the `onlyCore` modifier which allows only the existing `Core` contract to call it. However, given that the current `Core` implementation does not have any method that calls `switchCore`, it is likely that this method will never be callable, therefore rendering it useless.

Recommendation: Set up a method in `Core` that targets this method in `bBTC` such that it is not rendered ineffectual.

Update: From the team: "Intended behavior".

QSP-6 Lack of validation for arbitrary input `token` may lead to stolen token credit

Severity: Medium Risk

Status: Acknowledged

File(s) affected: `Zap.sol`

Description: Currently in `Zap`, the `mint` method does not check for whether the input `token` are the desired `wBTC` or `renBTC` addresses. Therefore, given that the only check for `token` is in the form of `safeTransferFrom` a specified amount, it is possible to set `token` to some arbitrary worthless token address to steal any underlying `wBTC` or `renBTC` on the `Zap` contract.

Recommendation: Validate that the input token is either in the expected `renBTC` or `wBTC` address.

Update: From the team: "Given that zap contract isn't expected to hold funds, we are ok with users being able to steal it, if some1 sends funds to zap. Doesn't affect the core system."

QSP-7 Setter functions may set to same variables

Severity: Low Risk

Status: Acknowledged

File(s) affected: `Guestlist.sol`

Description: There are some setter functions for the contracts involved that does not compare the incoming parameter to the current state variable. This may result in a misleading event emitted or successful result being returned, and may confuse components monitoring for these changes.
The following examples were spotted:

1. In Guestlist.sol, `setGuestRoot` does not validate for whether `guestRoot_` has changed.
2. In Guestlist.sol, `setUserDepositCap` does not validate for whether `cap_` has changed.
3. In Guestlist.sol, `setTotalDepositCap` does not validate for whether `cap_` has changed.

Recommendation: Validate for whether the incoming parameter is equal to the current state variable, and if so, revert the transaction.

Update: From the team: "Out of scope for audit.".
This file was actually in scope as per the original agreement, but after discussions with the team, they have decided to consider it removed.

QSP-8 Potential precision loss in division

Severity: Low Risk

Status: Fixed

Description: In the `_btcTobyvWBTC` method, a division of `1e20` is performed before the next division of `byvWBTC.pricePerShare`. This is probably done for the purpose of reducing the decimals to an intended `8`. However, given that the useful work is done in `byvWBTC.pricePerShare`, it would make more sense for the purpose of information preservation to perform that first, before `1e20`

Recommendation: Switch the order of the operations such that `byvWBTC.pricePerShare` comes first, then `1e20`.

Update: Recommendation adhered to.

QSP-9 Privileged Roles and Ownership

Severity: Informational

Status: Acknowledged

File(s) affected: `GuestList.sol`, `Core.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. In this audit, we note the following cases:

1. In GuestList.sol, the `owner` is able to set the `guestRoot` either to `0` to completely remove any need for proof verification, or change it to anything else to invalidate all the previously working invitation proofs.
2. In GuestList.sol, the `owner` is able to set the `userDepositCap`and/or `totalDepositCap` to `0` or low enough such that any further user deposit will not be allowed.
3. In Core.sol, the `owner` is able to set any existing peak to the extinct state arbitrarily, therefore removing it from `totalSystemAssets` and heavily affecting the return for the `bBTC` token.

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.

Update: No updates were given by the team besides the Acknowledged status.

QSP-10 Misleading usage of scaled, and under-informative function names

Severity: Informational

Status: Acknowledged

Description: The `Core` contract receives and sends out amounts of `btc` in different decimal places to the peak contracts. We have identified the following problems that has caused us confusion in this audit:

1. Core.sol::L52 states that `BTC amount supplied, scaled by 1e18`, which should be `BTC amount supplied, scaled by 1e10` or `BTC amount supplied, in 18 decimal places` instead.
2. Core.sol::L92 states that `btc amount redeemed, scaled by 1e18`, which should be `BTC amount redeemed, scaled by 1e18` or `BTC amount redeemed, in 36 decimal places` instead, given that the `btc` returned is in 36 decimal places from `bBtcToBtc`.
3. Core.sol::L104 states that `btc amount redeemed, scaled by 1e36`, which should be `BTC amount expected, scaled by 1e18` or `BTC amount expected, in 36`

`decimal places`. It also does not appropriately name the `fee` that is returned.

4. Core.sol::L106 function name of `bBtcToBtc` should be renamed `bBtcToBtcScaled` to more accurately and pre-emptively warn any users of the function of the uniquely scaled effect.
5. BadgerSettPeak.sol::L143 states that `BTC amount, scaled by 1e36`, which should be `BTC amount, scaled by 1e18` or `BTC amount, in 36 decimal places`.
6. BadgerSettPeak.sol::L152 states that `is already scaled by 1e36` which should be `is already scaled by 1e18`.
7. BadgerYearnWbtcPeak.sol::L119 states that `BTC amount, scaled by 1e36` which should be `BTC amount, scaled by 1e18` or `BTC amount, in 36 decimal places`.
8. BadgerYearnWbtcPeak.sol::L126 states that `this value is scaled by 1e36` which should be `this value is scaled by 1e18` or `this value, in 36 decimal places`.
9. BadgerYearnWbtcPeak.sol::L134 states that `btc value, scaled by 1e18` which should be `BTC value, scaled by 1e10` or `BTC value, in 18 decimal places`.
10. BadgerYearnWbtcPeak.sol::L141-142 states that `wBTC and byvWBTC are scaled by 8 decimals. Multiply by 100 to return a value scaled by 1e18` which should be `wBTC and byvWBTC are in 8 decimals. Raise the result by 1e2 to return a value in 18 decimal places.`

Recommendation: Follow the recommendations stated in the description.

Update: From the team: "We can work with the understanding that in the comments scaled by 1e18 means 18 decimals of precision. Fixed 10.2"

QSP-11 Magic numbers

Severity: *Informational*

Status: Acknowledged

File(s) affected: `Core.sol`, `BadgerSettPeak.sol`, `BadgerYearnWbtcPeak.sol`, `Zap.sol`

Description: Magic numbers are constant numbers used in the middle, usually without description and requires some inference to understand what is going on. It would be far better as a security and software engineering practice to have it declared as a named state constant, much like it was done for `Core::PRECISION`.
The following examples were discovered:

1. Core.sol::L114 | `1e18`
2. Core.sol::L116 | `1e18`
3. Zap.sol::L89 | `poolId + 2`
4. Zap.sol::L236 | `1e18`
5. BadgerSettPeak.sol::L118 | `1e18`
6. BadgerSettPeak.sol::L153 | `1e18`
7. BadgerSettPeak.sol::L172 | `1e36`
8. BadgerYearnWbtcPeak.sol::L102 | `1e18`
9. BadgerYearnWbtcPeak.sol::L127 | `1e20`
10. BadgerYearnWbtcPeak.sol::L145 | `100`

Recommendation: Set the magic numbers to appropriately named state constants and use those state constants instead.

Update: From the team: "This could have made it clearer. Not changing now because I dont wanna risk adding new variables (even constant ones) because that comes at the risk of messing up the storage slots."

QSP-12 Dormant peaks may `redeem`

Severity: *Undetermined*

Status: Acknowledged

Description: It is currently possible for peaks with status of `PeakState.Dormant` to `redeem` in `Core`, and it is not clear, due to the lack of technical documentation around that, if this is intentional.

Recommendation: Have clear technical documentation around `PeakState` and the different possible versions and what is allowed and what is not. It may not be intended for dormant peaks to be able to perform these actions, and if so, validate against that.

Update: From the team: "Intended behavior"

QSP-13 Cross peak withdrawal is allowed

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `Core.sol`, `BadgerSettPeak.sol`, `BadgerYearnWbtcPeak.sol`

Description: ibBTC resembles an index token across the different peaks that target different liquidity pools. Users currently can deposit into any of the pools to mint ibBTC and redeem it from any other arbitrary pools. This operates under the assumption that all BTC-derived tokens are equivalent in value, which may not always hold, and can allow for arbitrage of unbounded values. Building an index token does not necessarily require cross pool withdrawals as long as the value in shares and total supply calculations are done across pools.

Recommendation: Accounting for and limiting the withdrawals to the deposit pools can contain the exposure to the compromised pools only. Given the various inherent risk levels of different third-party pools (e.g Curve vs Yearn), containing the exposure can have security benefits.

Update: From the team: "Intended behavior"

Automated Analyses

Slither

Slither analyzed all the contracts, and 122 results were found, of which all of them were deemed to be false-positives.

Test Results

Test Suite Results

We were only able to run the test suite as per the instructions on one of the auditor's setup. It required access to an external node, in this case, an Alchemy API key was necessary.

```
  BadgerSettPeak
    ✓ mint (188ms)
    ✓ setPeakStatus (46ms)
    ✓ redeem (135ms)
    ✓ redeem fails for Extinct peak (71ms)
    ✓ collectFee (55ms)
    ✓ modifyWhitelistedCurvePools (276ms)

  Zero fee and redeem all
    ✓ setConfig
    ✓ mint (134ms)
    ✓ redeem (127ms)
    ✓ collectFee reverts when fee=0

  Core
    ✓ can't add duplicate peak
    ✓ whitelistPeak fails from non-admin account
    ✓ whitelistPeak fails for non-contract account
    ✓ setPeakStatus fails from non-admin account
    ✓ setPeakStatus (63ms)
    ✓ mint fails from unwhitelisted peak
    ✓ redeem fails from unwhitelisted peak
    ✓ can't set null fee sink

  BadgerSettPeak + SaddlePeak (mainnet-fork)
    ✓ saddlePeak.modifyWhitelistedCurvePools (558ms)
    ✓ whitelist saddle peak (12310ms)
    ✓ badgerPeak.modifyWhitelistedCurvePools (41ms)
    ✓ setConfig
    ✓ mint with saddleTWRenSBTC (5352ms)
    ✓ mint with bcrvRenWBTC (85411ms)
    ✓ mint with bcrvRenWSBTC (10597ms)
    ✓ mint with b-tbtc/sbtcCrv (21327ms)
    ✓ pricePerShare should increase after a trade (4550ms)
    ✓ redeem in bcrvRenWBTC (891ms)
    ✓ redeem in bcrvRenWSBTC (903ms)
    ✓ redeem in b-tbtc/sbtcCrv (808ms)
    ✓ redeem in saddleTWRenSBTC (342ms)
    ✓ sanity checks (155ms)

  BadgerSettPeak + BadgerYearnWbtcPeak (mainnet-fork)
    ✓ BadgerYearnWbtcPeak is whitelisted
    ✓ badgerPeak.modifyWhitelistedCurvePools
    ✓ setConfig
    ✓ mint with byvWBTC (2127ms)
    ✓ mint with bcrvRenWSBTC (779ms)
    ✓ mint with bcrvRenWBTC (683ms)
    ✓ mint with b-tbtc/sbtcCrv (633ms)
    ✓ pricePerShare should increase after a trade (2068ms)
    ✓ redeem in bcrvRenWSBTC (777ms)
    ✓ redeem in bcrvRenWBTC (752ms)
    ✓ redeem in b-tbtc/sbtcCrv (704ms)
    ✓ redeem in byvWBTC (492ms)
    ✓ sanity checks

  BadgerSettPeak (mainnet-fork)
    ✓ modifyWhitelistedCurvePools
    ✓ mint with bcrvRenWBTC (327ms)
    ✓ mint with bcrvRenWSBTC (735ms)
    ✓ mint with b-tbtc/sbtcCrv (620ms)
    ✓ redeem in bcrvRenWBTC (311ms)
    ✓ redeem in bcrvRenWSBTC (326ms)
    ✓ redeem in b-tbtc/sbtcCrv (283ms)

  BadgerYearnWbtcPeak (mainnet-fork)
    ✓ BadgerYearnWbtcPeak is whitelisted
    ✓ mint with byvWBTC (95ms)
    ✓ redeem in byvWBTC (109ms)

  Zap (mainnet-fork)
BigNumber.toString does not accept any parameters; base-10 is assumed
    ✓ admin whitelists (24722ms)
    ✓ mint with renbtc (118787ms)
    ✓ mint with wbtc (23350ms)
    ✓ approveContractAccess (2957ms)

  GuestList
    ✓ setup GuestList (476ms)
    ✓ mint sett LP
    ✓ invited guest (alice) can mint (55ms)
    ✓ alice cannot mint more than userDepositCap
    ✓ raise userDepositCap
    ✓ alice mints after userDepositCap raise (64ms)
    ✓ alice cannot mint more than totalDepositCap
    ✓ raise totalDepositCap
    ✓ alice mints after totalDepositCap raise (46ms)
    ✓ uninvited guest (bob) cannot mint
    ✓ include bob in GuestList
    ✓ newly invited guest (bob) mint (41ms)
    ✓ include pete in GuestList without merkleProof manually
    ✓ manually added guest (pete) mint (47ms)
    ✓ remove pete (no merkleProof invitation) from GuestList manually
    ✓ removed guest (pete) cannot mint
    ✓ remove bob (merkleProof invitation) from GuestList manually
    ✓ removed guest (bob) cannot mint


  77 passing (6m)
```

Code Coverage

We were only able to run the code coverage on one of the auditor's setup. The code coverage can and should be pushed to a much higher percentage, ideally at 100%, given the relatively brief nature of the codebase.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	38.41	33.33	60	38.85	
Core.sol	92.31	62.5	91.67	90.57	… 116,136,205
Zap.sol	0	0	0	0	… 235,236,237
bBTC.sol	62.5	33.33	66.67	66.67	35,36,40
contracts/common/	8.33	12.5	18.75	12.82	
AccessControlDefended.sol	60	50	60	71.43	27,31
GuestList.sol	0	0	0	0	… 144,149,150
contracts/common/proxy/	84.21	60	83.33	88.46	
GovernableProxy.sol	85.71	75	80	90	33
IERCProxy.sol	100	100	100	100	
Proxy.sol	0	100	50	50	38
UpgradableProxy.sol	90.91	50	100	92.86	40
contracts/interfaces/	100	100	100	100	
ICore.sol	100	100	100	100	
IPeak.sol	100	100	100	100	
ISett.sol	100	100	100	100	
ISwap.sol	100	100	100	100	
IbBTC.sol	100	100	100	100	
IbyvWbtc.sol	100	100	100	100	
contracts/peaks/	64	75	52.94	64.71	
BadgerSettPeak.sol	100	75	100	100	
BadgerYearnWbtcPeak.sol	0	100	0	0	… 112,126,143
All files	**42.8**	**34.38**	**52.86**	**45.1**	

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

```
015938371fdadb4c8996235f7fca667e7722f989a18519e1c2855c2914ee2325 ./contracts/bBTC.sol
ae55300f0371e8517b57b9d6f36b79a13cb5c47e0abdda8fcfb2023d5acda926 ./contracts/Zap.sol
d94dafda4dca1d75a39476a54cd8313de0ec3ffa726faa5738a5a7537cd861e8 ./contracts/Core.sol
c105a8e559cb09525cb19d7b595cb030ea19ef3a48e83ab87912e2f0ac6c833a ./contracts/interfaces/ICore.sol
6fcaf6149fb36650adaaa9d85df7caf80ea51e915dfa3ff6189fc2b0b5c618d3 ./contracts/interfaces/IbyvWbtc.sol
d352b1cc93ee68b7c4762a6f533560f7856ea3953cd91b23caf07241acc0809d ./contracts/interfaces/IbBTC.sol
adfb513defcf20c4df4b1e5aa0af0557c7d60fe6858c7cd20b7829f1eb82e0de ./contracts/interfaces/ISwap.sol
bc83c41872721406a7b15d66b8c8f8676f31147a10661638336062981fdc3e87 ./contracts/interfaces/IPeak.sol
75faa01063e2048216911a66f2406bd78191ac2bfa5176cf8f72d6eee9a14b77 ./contracts/interfaces/ISett.sol
db265fe377b8b811a35b024fa7b85997ee134186a0cb334ee0f7ae2177038bf0 ./contracts/common/GuestList.sol
6d9a408bf846ea26ffce2bd815d0ff8be12721262b4b5e3cf7fa657ba859fc5b ./contracts/common/AccessControlDefended.sol
5dc74be3c385188261936272c88e1f175843680058b9e3a56f8902040a642e50 ./contracts/common/proxy/IERCProxy.sol
56ee6a702666324d2ff33aa1a3c40a3cd65c5a5ab5921c5a3aefefb2106555f6 ./contracts/common/proxy/Proxy.sol
68ee4ff42892c257b9d9fc49263cb0f2efb6813a18f552023a369014d377308e ./contracts/common/proxy/GovernableProxy.sol
b2826b81a0c4d4f59db04b194bf548cb2caa98871e41ea411423c6971938c4f2 ./contracts/common/proxy/UpgradableProxy.sol
4b0cfb824e0015b80c964772797858dc9984844f15d81a5ab3d4436a7488c85b ./contracts/peaks/BadgerYearnWbtcPeak.sol
1a96b7c0e36221b7e4426fdb292dae1b99136a74de1bfbbaf01f4f22e3bc1c72 ./contracts/peaks/BadgerSettPeak.sol
```

Tests

```
f61dfef3f6a3682d56b69970b3b16dbbe5965c3e3e56494ba7566f1970d8ed4f ./contracts/test/ZapCall.sol
6538077a7cfc1932c0bb4b06ca7ef6f98f897bfe3655b7ac292ea310bd6b4b38 ./contracts/test/SaddlePeak.sol
cd993a67ef28bae871088d5b68a8cc1fcf58d1142c5cf32c723147e11318cc17 ./test/GuestList.js
b301d0f0bb0a34c77768cba530e65c0dd7dbb08a35bd22d7be1a6a38e60464c1 ./test/deployer.js
19761dd9f1b07a6325488cc3b911493489ae7a6c7bc731eb73d256eea07ab3b9 ./test/BadgerSettPeak.js
4a38787cb91a5d0ce68ab19a63dff494f089273a2ab09086dca74bf537c6df76 ./test/utils.js
51becc9371e38eb3d38d730f3ccc8ab456777c1d9cb3bef556e2b792678c9f3e ./test/Core.test.js
960b4949ca2819b4c679435c899124241475c0e123ba719c643cd6a4e0ddd725 ./test/fork/Zap.js
9ca56297519c539edf07be0c501d71b51bde1776fb33ebab0acf5f612a142619 ./test/fork/BadgerSettPeak.js
06ec88d728879e3befc2686bec9481481968e240530de8f46cf93ca8c3b12226 ./test/fork/YearnWbtcPeak.js
1dec0ba747efcebdb0e5f4855ce4afd67af59e631894caec125db865e14e9c71 ./test/fork/Badger-YearnWbtc-Peak.js
4a4d2ba822d893dc6c9b6dc8fa6cd6ecc400b8711874426cd1802670a87848b6 ./test/fork/Badger-Saddle-Peak.js
```

Changelog

- 2021-06-08 - Initial report
- 2021-08-05 - Final report

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp's team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp's collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.