Quantstamp Security Assessment Certificate

QUANTSTAMP VERIFIED SECURITY CERTIFICATE

May 5th 2021 — Quantstamp Verified

B20 Buyout Contracts

This security assessment was prepared by Quantstamp, the leader in blockchain security

Executive Summary

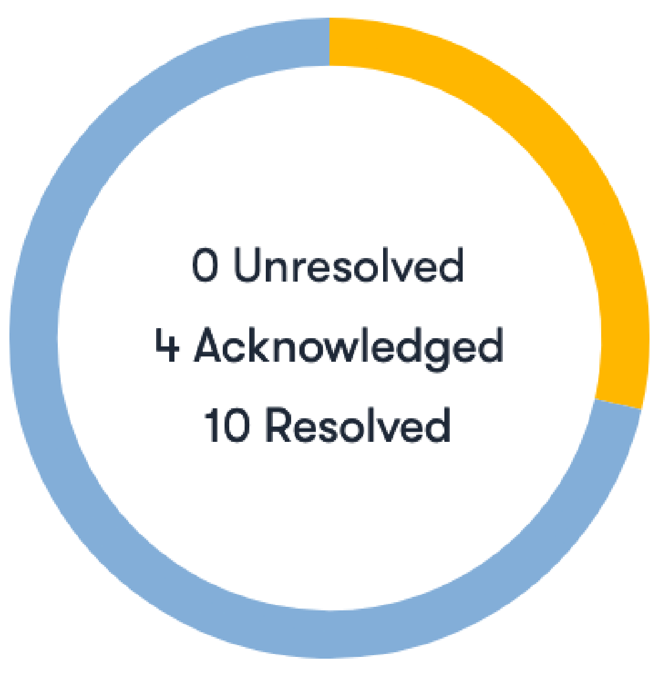
Type	NFT protocol
Auditors	Kacper Bąk, Senior Research Engineer Christoph Michel, Research Engineer Martin Derka, Senior Research Engineer
Timeline	2021-04-01 through 2021-05-03
EVM	Muir Glacier
Languages	Solidity, Javascript
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Documentation Quality	Low
Test Quality	Low
Source Code	

Repository	Commit
b20-contracts	d312f8a

Goals

- Is the implementation vulnerable to DoS attacks?
- Can tokens get locked up in a contract?
- Are computations implemented correctly?

Total Issues	14	(10 Resolved)
High Risk Issues	5	(5 Resolved)
Medium Risk Issues	3	(3 Resolved)
Low Risk Issues	4	(1 Resolved)
Informational Risk Issues	2	(1 Resolved)
Undetermined Risk Issues	0	(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 found a large number of issues in the reviewed code, although the codebase itself is relatively small. Notably, we consider majority of the issues high and medium severity. Under no circumstances do we recommend deploying the code as is. We recommend addressing all the issues, and improving the test suite quality and adding integration tests.
Update: the team addressed all the issues either by fixing, mitigating, or acknowledging them in the following:

- commit cada41d,
- commit f6a473e,
- PR#3,
- PR#5,
- PR#7,
- PR#8,
- PR#9.

All of these are merged in commit 8566c93.

ID	Description	Severity	Status
QSP-1	Anyone can veto bids and do a denial of service for the auction	High	Fixed
QSP-2	Wrong start threshold is used after a bid has been revoked	High	Fixed
QSP-3	Anyone can call `burnFrom()`	High	Fixed
QSP-4	`setDecentralandOperator` uses wrong asset ID	High	Fixed
QSP-5	Flashloan DoS	High	Mitigated
QSP-6	Wrong veto power is used when extending vetoes using `veto`	Medium	Fixed
QSP-7	Missing check if `SimpleBuyout` is vault owner	Medium	Mitigated
QSP-8	No reclaim mechanism for token0 in `SimpleMarket`/`SimpleMarket2`	Medium	Fixed
QSP-9	`epochFromTimestamp` returns wrong period on `HEART_BEAT_START_TIME`	Low	Acknowledged
QSP-10	Privileged Roles and Ownership	Low	Acknowledged
QSP-11	External interactions	Low	Acknowledged
QSP-12	Individual cap in `SimpleMarket2` can easily be circumvented	Low	Fixed
QSP-13	Allowance Double-Spend Exploit	Informational	Mitigated
QSP-14	Network jam susceptibility	Informational	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 v0.7.0

Steps taken to run the tools:

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

Findings

QSP-1 Anyone can veto bids and do a denial of service for the auction

Severity: *High Risk*

Status: Fixed

File(s) affected: `SimpleBuyout.sol`

Description: Anyone can acquire a small amount of token0s and call `veto(amount)` followed by `withdrawStakedToken0()` and repeat this process. On each iteration, the `amount` is added to `currentBidToken0Staked` until it reaches the threshold and the current bid is revoked, causing a denial of service.

Recommendation: Disallow withdrawals if the auction has not ended and the user has vetoed in the current period.

Update: Fixed in commit cada41d

QSP-2 Wrong start threshold is used after a bid has been revoked

Severity: *High Risk*

Status: Fixed

File(s) affected: `SimpleBuyout.sol`

Description: After a successful veto, the threshold is set to `startThreshold = highestBidValues[2].mul(108).div(100);`, where `highestBidValues[2]` is the token2 amount of the highest bid.

Recommendation: This should be the *overall* amount `highestBidValues[0]` instead, which is also compared against in `placeBid()`.

Update: Fixed in commit cada41d

QSP-3 Anyone can call `burnFrom()`

Severity: *High Risk*

Status: Fixed

File(s) affected: `Token0.sol`

Description: Only owner can call `burn()` but anyone can call `burnFrom()` and skip the ownership check. Note that token0's total supply is used in `SimpleBuyout.sol`

Recommendation: Unless there is a good reason not to, protect `burnFrom()` via the modifier `onlyOwner`.

Update: Fixed in PR#3. It is important to note that the owner the `token0` needs to be set to `SimpleBuyout`.

QSP-4 `setDecentralandOperator` uses wrong asset ID

Severity: *High Risk*

Status: Fixed

File(s) affected: `SimpleVault.sol`

Description: The function `setDecentralandOperator()` invokes `setUpdateOperator()` with the variable `assetId` which is not the asset ID but the asset *index* in the contract's `assets` array.

Recommendation: This was probably supposed to be `IDecentralandLandRegistry(registryAddress).setUpdateOperator(assets[assetIds[i]].tokenId), operatorAddress);`. In addition, rename `assetId` to a more accurate name, e.g., `assetIndex`.

Update: Fixed in PR#9

QSP-5 Flashloan DoS

Severity: *High Risk*

Status: Mitigated

File(s) affected: `SimpleBuyout.sol`

Description: A user can cause a DoS by submitting a veto (maybe using flash loans and mints) to evict the highest bidder and reset the highest bid to 0, and then withdrawing the staked tokens.

Recommendation: We recommend adding protection against flashloan DoS attacks, e.g., by disallowing staking and unstaking in the same block.

Update: The issue is mitigated in commit f6a473e.

QSP-6 Wrong veto power is used when extending vetoes using `veto`

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `SimpleBuyout.sol`

Description: Method `extendVeto()` cannot be used in combination with a regular veto. While `extendVeto()` requires a unique last vetoed bid, the `veto()` method does not. If a user calls `veto(amount)` for the current bid, the bid may be revoked and a new bid starts. If the user would like to veto the new bid again with more veto power and calls `veto(amount2)` (instead of calling `extendVeto()` and `veto()` first). Only the additional `amount2` is used for the veto and `extendVeto()` cannot be called again.
This differs from the specification as `veto()` should take into account the already existing stake:

> If the user had vetoed an earlier bid, the staked B20 tokens and the veto amount are added to the veto stack

Recommendation: We recommend adjusting the code so that `veto()` considers the already staked amount if the user hasn't vetoed the given bid. Also, explain use cases for both methods and/or consider removing one of them.

Update: Fixed in commit cada41d. Furthermore, the team explained that the reason for the existence of both methods is mainly to optimize gas costs for the end user if they choose to extend their veto. According to their simulations `SimpleBuyout.veto()` costs 110,541 gas on average while `SimpleBuyout. extendVeto()` costs 65,846 gas on average. The latter is about 60% cost reduction for the end user.

QSP-7 Missing check if `SimpleBuyout` is vault owner

Severity: Medium Risk

Status: Mitigated

File(s) affected: `SimpleBuyout.sol`

Description: It's important that the `SimpleBuyout` contract is the vault owner, otherwise the auction would fail in `endBuyout()` with the `vault.transferOwnership(highestBidder);` call and the highest bidder would not receive the vault's asset.

Recommendation: Check if the `SimpleBuyout` contract is the vault owner before placing bids. Add an option to restore original ownership in case no bid was accepted.

Update: The issue is mitigated in PR#6. Furthermore, the team informed us that they have decided to burn the admin keys of the Buyout contract, and therefore, Vault transfer by the Buyout owner shall not be possible.

QSP-8 No reclaim mechanism for token0 in `SimpleMarket`/`SimpleMarket2`

Severity: Medium Risk

Status: Fixed

File(s) affected: `SimpleMarket.sol`, `SimpleMarket2.sol`

Description: The simple market contracts require a deposit of token0s to be able to trade. However, they cannot be withdrawn again if the cap is reached or the market closed.

Recommendation: Ensure enough token0s are deposited after creating the contract and add a way to retrieve token0s after the market is closed.

Update: Fixed in PR#7

QSP-9 `epochFromTimestamp` returns wrong period on `HEART_BEAT_START_TIME`

Severity: Low Risk

Status: Acknowledged

File(s) affected: `Pacemaker.sol`

Description: The epochs are supposed to start *on* the start times but this fails for the first epoch which should start at `HEART_BEAT_START_TIME`. The `epochFromTimestamp` returns 0 instead of 1.

Recommendation: Change `if (timestamp > HEART_BEAT_START_TIME)` to `if (timestamp >= HEART_BEAT_START_TIME)`

QSP-10 Privileged Roles and Ownership

Severity: Low Risk

Status: Acknowledged

File(s) affected: `SimpleVault.sol`, `SimpleMarket.sol`, `SimpleMarket2.sol`, `Token0.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. For example, the vault owner can withdraw assets and transfer ownership at any time using the `escapeHatchERC721()` and `transferOwnership()` functions.

Recommendation: Inform users about potential risks.

QSP-11 External interactions

Severity: Low Risk

Status: Acknowledged

File(s) affected: `SimpleBuyout.sol`, `SimpleVault.sol`, `SimpleMarket.sol`, `SimpleMarket2.sol`

Description: B20 contracts interact with external contracts. This may have unintended consequences if little attention is paid to the functionality of external contracts.
For example, placing bids can fail for ERC777 tokens. The functions `placeBid()` and `_resetHighestBidDetails()` refund the latest bid to the highest bidder. A malicious highest bidder can deploy a token contract and revert the transaction upon receiving the transfer which breaks the auction functionality.

Recommendation: Make sure that any external contracts are reviewed carefully and behave as assumed.

QSP-12 Individual cap in `SimpleMarket2` can easily be circumvented

Severity: Low Risk

Status: Fixed

File(s) affected: `SimpleMarket2.sol`

Description: The individual cap in `SimpleMarket2` can easily be circumvented by using a different account to buy and is more or less meaningless as there's no whitelist.

Recommendation: Remove the individual cap or add a whitelist of buyers.

Update: Fixed in PR#5

QSP-13 Allowance Double-Spend Exploit

Severity: Informational

Status: Mitigated

File(s) affected: `Token0.sol`

Description: As it presently is constructed, the contract is vulnerable to the allowance double-spend exploit, as with other ERC20 tokens.

Exploit Scenario:

1. Alice allows Bob to transfer `N` amount of Alice's tokens (`N>0`) by calling the `approve()` method on `Token` smart contract (passing Bob's address and `N` as method arguments)
2. After some time, Alice decides to change from `N` to `M` (`M>0`) the number of Alice's tokens Bob is allowed to transfer, so she calls the `approve()` method again, this time passing Bob's address and `M` as method arguments
3. Bob notices Alice's second transaction before it was mined and quickly sends another transaction that calls the `transferFrom()` method to transfer `N` Alice's tokens somewhere
4. If Bob's transaction will be executed before Alice's transaction, then Bob will successfully transfer `N` Alice's tokens and will gain an ability to transfer another `M` tokens
5. Before Alice notices any irregularities, Bob calls `transferFrom()` method again, this time to transfer `M` Alice's tokens.

Recommendation: The exploit (as described above) is mitigated through use of functions that increase/decrease the allowance relative to its current value, such as `increaseAllowance()` and `decreaseAllowance()`.
Pending community agreement on an ERC standard that would protect against this exploit, we recommend that developers of applications dependent on `approve()` / `transferFrom()` should keep in mind that they have to set allowance to 0 first and verify if it was used before setting the new value. Teams who decide to wait for such a standard should make these recommendations to app developers who work with their token contract.

QSP-14 Network jam susceptibility

Severity: Informational

Status: Acknowledged

File(s) affected: `SimpleBuyout.sol`

Description: The contract relies on liveness of the network for bidding and vetoing. If the network is not alive, or is purposely jammed, the auction can expire.

Recommendation: We recommend informing users about this possible risk.

Automated Analyses

Slither

Slither reported that:

1. `SimpleMarket.sol#83` and `SimpleMarket2.sol#88` sets array length with a user-controlled value. We consider it a false positive.
2. `SimpleBuyout._veto()` performs a multiplication on the result of a division (Line 239). We recommend changing the order of these operations.
3. Reentrancies in `SimpleBuyout._resetHighestBidDetails()`, `SimpleBuyout.placeBid()`, and `SimpleVault.safeTransferAsset()` due to `safeTransfer()` and `safeTransferFrom()` calls. We recommend vetting external contracts before using them to eliminate risks related to reentrancies.

Code Documentation

1. `README.md` mentions `SimpleRedeem.sol` but the file does not belong to the repo. **Update:** fixed.
2. `README.md` does not mention `SimpleMarket2.sol`. **Update:** fixed. **Update:** issues fixed in PR#8.

Adherence to Best Practices

1. `SimpleVault.safeTransferAsset()` could be optimized to use less gas by storing the `assets` array in memory first; lots of `assets` storage reads. **Update:** acknowledged.
2. Naming of variables could be more expressive, for example `token0` -> `shareToken`, `token1`, `token2` -> `paymentToken`, `assetId` -> `assetIndex`. Also `uint256Values` is not descriptive; use the actual parameter names. **Update:** partially fixed.
3. The comment is wrong `// uint256Values[1], aka, bidIntervalInEpochs can be zero, so no checks required.`; should be `uint256Values[2]`. **Update:** fixed.
4. Replace the hardcoded 8% and 5% literals in `SimpleBuyout.sol` with named constants. **Update:** fixed.
5. The `buyers` array in `SimpleMarket.sol` might not be needed and could save gas. **Update:** acknowledged.
6. In `SimpleVault.sol#129``== !` can be replaced with `!=`. **Update:** fixed.
7. `SimpleMarket2.sol` is a code clone of `SimpleMarket.sol`. **Update:** fixed.
8. No draining functions for ERC20 token contracts. The contract assumes that ERC20 tokens will be sent around. We recommend adding draining functions for tokens that were sent accidentally. **Update:** fixed. **Update:** unless stated otherwise, issues fixed in PR#8.

Test Results

Test Suite Results

All tests executed successfully, however, given the amount of issues we have found during the audit, we consider the quality of tests low. The tests do not catch any potential integration problems. We recommend further testing using some fork of the mainnet.

```
Contract: SimpleVault
  safeAddAsset
     ✓ [require] - caller needs to be the owner (24054 gas)
     ✓ [require] - vault needs to be unlocked
     ✓ [revert] - when empty tokenAddresses (25009 gas)
     ✓ [revert] - when tokenAddresses and tokenIds lengths are not equal (25974 gas)
     ✓ [revert] - when tokenAddresses and categories lengths are not equal (25637 gas)
     ✓ [revert] - when invalid tokenAddress (27117 gas)
     ✓ [revert] - when invalid tokenId (3143749 gas)
     ✓ [success] - supports adding NFT with deprecated onERC721Received()
     ✓ [success] - when owner adds his assets (2481356 gas)
  safeTransferAsset
     ✓ [require] - caller needs to be the owner (23069 gas)
     ✓ [require] - vault needs to be unlocked
     ✓ [revert] - when empty assetIndices (24024 gas)
     ✓ [revert] - when invalid assetIndices (25092 gas)
     ✓ [success] - works when called by owner (2481356 gas)
     ✓ [success] - updates storage correctly (399328 gas)
     ✓ [fails] - when transfer already transferred asset (111684 gas)
  lockVault
     ✓ [require] - caller needs to be the owner (22498 gas)
     ✓ [require] - vault needs to be unlocked
     ✓ [success] - when called by owner (29108 gas)
     ✓ [fail] - when vault already locked (23440 gas)
     ✓ [prevent] - when add assets after locking vault (26020 gas)
     ✓ [prevent] - when transfer assets after locking vault (24115 gas)
  unlockVault
     ✓ [require] - caller needs to be the owner (22431 gas)
     ✓ [require] - vault needs to be locked
     ✓ [success] - when called by owner (29041 gas)
     ✓ [fail] - when vault already unlocked (23373 gas)
     ✓ [allow] - when add assets after unlocking vault (225791 gas)
     ✓ [allow] - when transfer assets after unlocking vault (75693 gas)
  transferOwnership
     ✓ [require] - caller needs to be the owner (23151 gas)
     ✓ [revert] - when invalid new owner (22967 gas)
     ✓ [success] - when called by owner (32183 gas)
     ✓ [prevent] - add/transfer assets after ownership transfer (48323 gas)
     ✓ [allow] - add/transfer assets with new owner (3350002 gas)
  escapeHatchERC721
     ✓ [require] - caller needs to be the owner (23506 gas)
     ✓ [revert] - when invalid tokenAddress (23322 gas)
     ✓ [revert] - when invalid tokenId (28881 gas)
     ✓ [success] - when called by owner (80891 gas)
  setDecentralandOperator
     ✓ [require] - caller needs to be the owner (24070 gas)
     ✓ [revert] - when invalid addresses (47810 gas)
     ✓ [revert] - when invalid assetIndex (24995 gas)
     ✓ [success] - when called by owner (28515 gas)

Contract: SimpleMarket
  createMarket
     ✓ deploys with owner
     ✓ cannot be called by non-owner (25825 gas)
     ✓ fails with invalid token0Address (55370 gas)
     ✓ fails with invalid token1Address (57064 gas)
     ✓ fails with invalid fundsWalletAddress (28494 gas)
     ✓ fails with invalid marketStart (57502 gas)
     ✓ fails with invalid totalCap (276475 gas)
     ✓ fails with invalid token1PerToken0 (28829 gas)
     ✓ configures market parameters correctly (324178 gas)
  pay
     ✓ fails when contribution amount is 0 (23794 gas)
     ✓ succeeds
     ✓ fails if contribution amount exceeds totalCap (607874 gas)
     ✓ fails if market is closed (153147 gas)
  closeMarket
     ✓ fails if called by non-owner (22268 gas)
     ✓ works even if totaltoken1Paid is 0
     ✓ succeeds (559165 gas)
     ✓ fails if called more than once (23394 gas)

Contract: SimpleMarket2
  whitelistAddresses
     ✓ [require] - caller needs to be the owner (23959 gas)
     ✓ [fail] - invalid params (59585 gas)
     ✓ [success] - valid inputs (89376 gas)
  createMarket
     ✓ deploys with owner
     ✓ cannot be called by non-owner (26279 gas)
     ✓ fails with invalid token0Address (56786 gas)
     ✓ fails with invalid token1Address (58480 gas)
     ✓ fails with invalid fundsWalletAddress (29202 gas)
     ✓ fails with invalid marketStart (58918 gas)
     ✓ fails with invalid totalCap (277891 gas)
     ✓ fails with invalid token1PerToken0 (29537 gas)
     ✓ fails with invalid individualCap (26423 gas)
     ✓ configures market parameters correctly (346546 gas)
  pay
     ✓ fails when contribution amount is 0 (24692 gas)
     ✓ fails when user not whitelisted (22937 gas)
     ✓ succeeds
     ✓ fails if contribution amount exceeds individualCap (316077 gas)
     ✓ fails if market is closed (124111 gas)
  closeMarket
     ✓ fails if called by non-owner (22334 gas)
     ✓ works even if totaltoken1Paid is 0
     ✓ succeeds (564649 gas)
     ✓ fails if called more than once (23460 gas)

Contract: SimpleBuyout
  constructor
     ✓ deploys with owner
     ✓ [config] - buyout parameters correctly
  togglePause
     ✓ [require] - caller needs to be the owner (22697 gas)
     ✓ [success] - toggle pause (61117 gas)
  placeBid
     ✓ [require] - contract is not paused
     ✓ [require] - buyout not ended
     ✓ fails with totalBidAmount < minimum threshold (25116 gas)
     ✓ fails with insufficient token2 balance (29293 gas)
     ✓ fails with insufficient token0 balance (194141 gas)
     ✓ fails with less than 5% of token0 totalSupply (139118 gas)
     ✓ success when inputs are correct (225847 gas)
     ✓ fails with totalBidAmount < previous bid (192011 gas)
     ✓ fails if expiry has been reached or buyout ended (95363 gas)
     ✓ updates storage correctly when bids are placed (173875 gas)
  withdrawBid
     ✓ [require] - contract is paused
     ✓ fails if called by non-highest-bidder (23349 gas)
     ✓ succeeds if called by highest-bidder on paused contract (57096 gas)
  veto
     ✓ [require] - contract is not paused and is actuve
     ✓ [require] - token0 amount cannot be zero (22916 gas)
     ✓ fails if buyout expiry has been reached
     ✓ updates storage correctly when token0s are staked (766052 gas)
  extendVeto
     ✓ [require] - contract is not paused
     ✓ fails if token0 has not been staked (23405 gas)
     ✓ fails on already vetoed bid (564920 gas)
     ✓ succeeds on non-vetoed bid (339284 gas)
  withdrawStakedToken0
     ✓ fails if token0Amount is zero (22853 gas)
     ✓ fails if token0Amount > staked amount (23870 gas)
     ✓ fails until current bid expires (28105 gas)
     ✓ updates storage correctly when buyout is active and staked token0s are unstaked (101460 gas)
  endBuyout
     ✓ [require] - contract is not paused
     ✓ [require] - highestBidder should exists
     ✓ fails if no bid was placed at all (25898 gas)
     ✓ fails if currentEpoch has not surpassed endEpoch (249907 gas)
     ✓ updates storage correctly when successful (65524 gas)
     ✓ [fail] - contract is already ended (25024 gas)
  requiredToken0ToBid
     ✓ fails if token2Amount >  totalBidAmount
     ✓ returns correct values
  transferVaultOwnership
     ✓ [require] - contract is paused
```

```
    ✓ fails if called by non-owner (23033 gas)
    ✓ fails if newOwner address is zero (23899 gas)
    ✓ succeeds if called by non-owner on paused contract (37193 gas)
  redeem
    ✓ fails if status is not ENABLED (23008 gas)
    ✓ fails with invalid token0 amount (insufficient) (92709 gas)
    ✓ fails with invalid token0 amount (zero) (27124 gas)
    ✓ works as expected (67990 gas)

Contract: SimpleVault2
  safeAddAsset
    ✓ [require] - caller needs to be the nftKey owner (29365 gas)
    ✓ [require] - vault needs to be unlocked
    ✓ [revert] - when empty tokenAddresses (30302 gas)
    ✓ [revert] - when tokenAddresses and tokenIds lengths are not equal (31267 gas)
    ✓ [revert] - when tokenAddresses and categories lengths are not equal (30930 gas)
    ✓ [revert] - when invalid tokenAddress (32410 gas)
    ✓ [revert] - when invalid tokenId (3164991 gas)
    ✓ [success] - supports adding NFT with deprecated onERC721Received()
    ✓ [success] - when owner adds his assets (2561244 gas)
  safeTransferAsset
    ✓ [require] - caller needs to be the owner (28402 gas)
    ✓ [require] - vault needs to be unlocked
    ✓ [revert] - when empty assetIndices (29339 gas)
    ✓ [revert] - when invalid assetIndices (30407 gas)
    ✓ [success] - works when called by owner (2561244 gas)
    ✓ [success] - updates storage correctly (436563 gas)
    ✓ [fails] - when transfer already transferred asset (127642 gas)
  lockVault
    ✓ [require] - caller needs to be the owner (27809 gas)
    ✓ [require] - vault needs to be unlocked
    ✓ [success] - when called by owner (49407 gas)
    ✓ [fail] - when vault already locked (28733 gas)
    ✓ [prevent] - when add assets after locking vault (31313 gas)
    ✓ [prevent] - when transfer assets after locking vault (29430 gas)
  unlockVault
    ✓ [require] - caller needs to be the owner (27742 gas)
    ✓ [require] - vault needs to be locked
    ✓ [success] - when called by owner (19340 gas)
    ✓ [fail] - when vault already unlocked (28666 gas)
    ✓ [allow] - when add assets after unlocking vault (241740 gas)
    ✓ [allow] - when transfer assets after unlocking vault (81014 gas)
  transferOwnership
    ✓ [revert] - when invalid new owner (22027 gas)
    ✓ [revert] - even if called by key owner (21986 gas)
    ✓ [prevent] - add/transfer assets after key ownership transfer (145323 gas)
    ✓ [allow] - add/transfer assets with new key owner (3376594 gas)
  escapeHatchERC721
    ✓ [require] - caller needs to be the owner (28817 gas)
    ✓ [revert] - when invalid tokenAddress (28615 gas)
    ✓ [revert] - when invalid tokenId (34174 gas)
    ✓ [success] - when called by owner (91512 gas)
  setDecentralandOperator
    ✓ [require] - caller needs to be the owner (29381 gas)
    ✓ [revert] - when invalid addresses (58396 gas)
    ✓ [revert] - when invalid assetIndex (30288 gas)
    ✓ [success] - when called by owner (33808 gas)


165 passing (2m)
```

Code Coverage

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	96.71	90.7	94	96.76	
IDecentralandLandRegistry.sol	100	100	100	100	
IToken0.sol	100	100	100	100	
IVault.sol	100	100	100	100	
SimpleBuyout.sol	98.18	84.15	93.33	98.18	94,95
SimpleMarket.sol	100	100	100	100	
SimpleMarket2.sol	100	94.44	100	100	
SimpleMarketBase.sol	91.18	95.45	87.5	91.18	50,51,64
SimpleVault.sol	94.74	100	92.31	95	117,118
SimpleVault2.sol	100	83.33	100	100	
Token0.sol	100	100	100	100	
contracts/access/	90	25	80	81.82	
Ownable.sol	90	25	80	81.82	46,47
All files	**96.41**	**89.2**	**92.73**	**96.04**	

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

```
277067bfc30175d61a17adf18b69461ad7d50b3e6dd48e3cf47920f7df9d68f2 ./contracts/SimpleBuyout.sol

c8a5d84df178ece9e6929c2f19ece1b5f95698531e71ec837e2e3a8e26aa385a ./contracts/IToken0.sol

0a74a74ff34d1a1209df6834286f21f915ca4c644a98f56d5abb7a5a4cd02319 ./contracts/Token0.sol

d4167d0a5c51541b3d6e94f3daa2a087ea12a840f8139bc8ee4f21e1fb9df35e ./contracts/IDecentralandLandRegistry.sol

086d825fa7dce563b59822c57dc2a82cf09846295a659ccd23d01c93e876d90b ./contracts/SimpleVault2.sol

dc7ae510ccc6ada76aba539a3cd3089bd318637c41e1a3683a192d659d48cb53 ./contracts/IVault.sol

9484dcc847cafe8935685c1f087ac82424585507ea0cce59e2a88da73456ba7e ./contracts/SimpleMarketBase.sol

e0eb43174a0e73712a2529f9adadaac0bc9de9617966db38a902e229d571eb36 ./contracts/SimpleMarket2.sol

a0bca864b3fe7de95c3a76b2fb5ae30a3b8bbd77efdd048d2b156c41f4b7ce4c ./contracts/SimpleMarket.sol

9b0fced0981ed7a684f427e9882dd0dbc647a942924f08a5aff90e7d821c0df2 ./contracts/SimpleVault.sol

684bf0255dcadbfa5cb351ccf0e62a5ea57abbebdd0f12214077f8f90f1e9cbf ./contracts/heartbeat/Pacemaker.sol

729218f53c89d6304415701fc8305f96c307fb49f0a00be36c68b6b318d07cfd ./contracts/access/Ownable.sol

9d333ee5a43475507fc062e4f56e21c0189ea94ee0e75c0b477bf6156bf4df32 ./contracts/mocks/MockPacemaker.sol

3ce0b86a88a35d081b44589c3945db6f943a472c5b131e3fd3e0ea61e4ae2f86 ./contracts/mocks/MockMintableERC721.sol

15ada510077da6ea9d081b296e2ee2e6b4d1035d54012526d2aa7a9540e8db94 ./contracts/mocks/MockSimpleWallet.sol

ae9aaf54c56e504aa0e038f0e29b60dc45945ee2510ace967970336bf965eab6 ./contracts/mocks/MockNFTB20.sol

24bdb1315d7ccd96342db249338b3cabd8746b54d64079b9999ecd80e541cba3 ./contracts/mocks/MockToken2.sol

9ba3696445a9e88f76852299a9368bbcd6b5d816502d0281be093050e558ab97 ./contracts/mocks/MockToken0.sol

9ed2b46722c0abc125fd213d275e935ff20df12cdae63233286e77b0c087f6a6 ./contracts/mocks/MockToken1.sol

0662b9a65b3f9b8e1b11510b5e9c668715352090606991478641 20f6079414f2 ./contracts/mocks/MockNFT1.sol

3eda326d199789d3360d1012a69299fc58a4af14bcf19b671aebf2ce754877af ./contracts/mocks/MockNFT3.sol

e0e328373f7049e69cbfd72e41216f40943193d4758c20e7658324f37e58d3c3 ./contracts/mocks/MockNFT2.sol
```

Tests

```
d76a6646357e50ef3e1d7e9e43f8fdca961f66ff69d8bc2ed44efcd605e64157 ./test/test.Market.js

2b409ea601461b75e27f306804665ea66fb15443831746c5cf5c5e3d90953ac2 ./test/test.Vault2.js

86978c83496b66f573b6e9a3cc66d5ddc4e42ab9151810748cfa96ab15333805 ./test/contracts.test.js

5936266455de70f78aeb125696bed271ffb03953a4ad3e43e7c8d5baf42d250b ./test/test.Market2.js

bc594421c03be12ad0c71837e2b4090f8b05a4cd33f27166c074ec5eb1a5c70a ./test/test.Buyout.js

c31aa2cc7576805a31694e74e7391a1a7e374d8709b35a6a092e2d1ed8462ba1 ./test/test.Vault.js

a78ad0627e59f6f6531443a94e0b41ec0627f7d3530f9720391c95f0ebcdb537 ./test/helpers/currentEpoch.js
```

Changelog

- 2021-04-07 - Initial report
- 2021-05-03 - Reaudit based on commit 8566c93 that includes fixes in commit cada41d, commit f6a473e, PR#3, PR#5, PR#7, PR#8, PR#9.

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.