Quantstamp Security Assessment Certificate

QUANTSTAMP VERIFIED SECURITY CERTIFICATE

December 4th 2020 — Quantstamp Verified

DerivaDEX

This smart contract audit was prepared by Quantstamp, the protocol for securing smart contracts.

Executive Summary

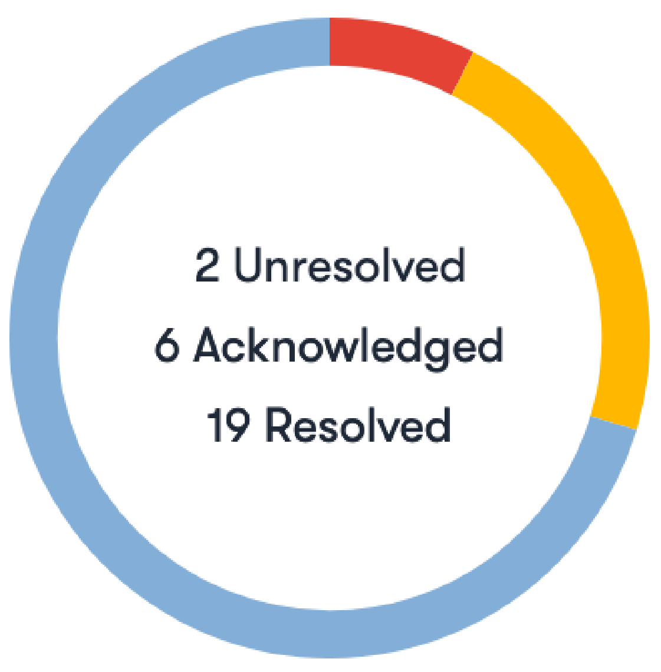
Type	Trading Platform
Auditors	Ed Zulkoski, Senior Security Engineer Fayçal Lalidji, Security Auditor Kacper Bąk, Senior Research Engineer
Timeline	2020-08-21 through 2020-10-27
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	AUDIT.md Diamond Standard (EIP-2535)
Documentation Quality	High
Test Quality	High
Source Code	

Repository	Commit
derivadex_v1	6bb9ac9

Goals

- Can funds be permanently locked or stolen?
- Do the contracts adhere to the diamond standard?
- Can the governance system be gamed or used maliciously?

Total Issues	27	(19 Resolved)
High Risk Issues	3	(3 Resolved)
Medium Risk Issues	5	(4 Resolved)
Low Risk Issues	7	(3 Resolved)
Informational Risk Issues	11	(8 Resolved)
Undetermined Risk Issues	1	(1 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

Overall, the code and documentation in the DerivaDEX smart contracts is of very high quality. Nonetheless, during the audit we uncovered several issues, both relating to the use of the diamond standard, as well as code in the facets themselves. Importantly, since certain contracts such as `InsuranceFund.sol` rely upon the security of external token contracts, caution should be used when adding new tokens to the system. We suggest addressing all issues found before using the code in production.
Update: The report has been updated to include updates in commit c4f974a, including `InsuranceFund.sol`. New findings have been appended to each section (in particular, QSP-20 through QSP-29, along with extensions to QSP-2 and QSP-4). As of this commit, the test suite has not been run; updated scripts will be used to run tests in future revised reports.
Update: All issues have been resolved, mitigated, or acknowledged as of commit 0dbe8788.
Update 2: The report has been extended to include commit 7194839, which primarily contains updates to the `InsuranceFund`. New findings were noted in QSP-26 and QSP-27, as well as appended to the Best Practices and Documentation sections.

ID	Description	Severity	Status
QSP-1	Cloneable wallet can be re-initialized	High	Fixed
QSP-2	DoS Attack in `getPriorVotes()`	High	Fixed
QSP-3	Diamond proxy pattern increases the attack surface of governance controlled contracts	High	Mitigated
QSP-4	Missing input validation	Medium	Fixed
QSP-5	`totalSupply()` does not denote the current total supply	Low	Fixed
QSP-6	Mapping Storage Layout	Low	Acknowledged
QSP-7	`skipRemainingVotingThreshold` may cause voting "races" if set below 50%	Low	Fixed
QSP-8	Flashloan Vote	Informational	Fixed
QSP-9	Nonstandard Naming in `transferFrom()`	Informational	Fixed
QSP-10	Unlocked Pragma	Informational	Fixed
QSP-11	Redundant Requirements	Informational	Fixed
QSP-12	Cannot retrieved ether deposited through `receive()`	Informational	Fixed
QSP-13	Contract will not pause if multiple diamond facet upgrade transactions occur	Informational	Acknowledged
QSP-14	Storage data packing can be optimized	Informational	Fixed
QSP-15	Variable type inconsistency	Informational	Fixed
QSP-16	Allowance Double-Spend Exploit	Informational	Mitigated
QSP-17	Gas Usage / `for` Loop Concerns	Medium	Fixed
QSP-18	`permit()` does not validate ECDSA parameters	Medium	Fixed
QSP-19	Staking may result in loss of funds when different multipliers are used	Medium	Fixed
QSP-20	Aave accrued interest model may be gamed by users staking toward the end of an interval	Medium	Acknowledged
QSP-21	Ignored exception cases in transfer functions	Low	Fixed
QSP-22	Unable to remove collateral types in `InsuranceFund`	Low	Acknowledged
QSP-23	External tokens should be added cautiously to `InsuranceFund`	Informational	Acknowledged
QSP-24	DDX reward computation dependent on USD-pegged stable coins	Low	Acknowledged
QSP-25	Unclear if-case in `claimDDXFromInsuranceMining()`	? Undetermined	Fixed
QSP-26	Users must check-in every rewards interval to avoid diluted COMP and Aave rewards	Low	Unresolved
QSP-27	Privileged Roles and Ownership	Informational	Unresolved

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 v0.6.6
- Mythril v0.2.7

Steps taken to run the tools:

1. Installed the Slither tool: `pip install slither-analyzer`
2. Run Slither from the project directory: `slither .s`
3. Installed the Mythril tool from Pypi: `pip3 install mythril`
4. Ran the Mythril tool on each contract: `myth -x path/to/contract`

Findings

QSP-1 Cloneable wallet can be re-initialized

Severity: *High Risk*

Status: Fixed

File(s) affected: `DDXWalletCloneable.sol`

Description: The `initialize()` function member of `DDXWalletCloneable` is used to set the wallets that hold the traders DDX fund. When creating the clone contract in Trader the constructor is not called, however, the `initialize()` function can be called later on to set the approval and delegation.
Once the clone created and initialized, any attacker can re-initialize the clone since initialize does not require to be intialized only once. As a result all DDX funds can be taken since the attacker can approve the tokens to his own address.

Recommendation: Add a requirements that checks if `initialize()` was not previously called; revert the transaction if so.

QSP-2 DoS Attack in `getPriorVotes()`

Severity: *High Risk*

Status: Fixed

File(s) affected: `DDX.sol`, `DIFundToken.sol`

Description: The function `getPriorVotes()` load an entire Checkpoints structure `cp = checkpoints[_account]` to memory from storage. The checkpoint structure contains two dynamic arrays, thus, an attacker can start an attack where he delegates his voting power to an important voting address with high weight, causing the delegatee Checkpoint arrays toincrease in size when the attacker make transfers back and forth between his addresses. If the array sizes increase enough the attacker can render voting inacessible for the delegatee address, since `getPriorVotes()` will consume excessive gas when loading the arrays into the memory and when the while loop will execute, and therefore increase his chances to pass a proposal.
`getPriorVotes()` checks the most recent balance before loading the storage to the memory and executing the fetching algorithm, however, this can be bypassed by making a token transfer from the attacker address that delegates its voting power to the attacked address when the proposal voting starts.
A more sophisticated attack can involve a malicous contract that own sub-contracts that attack multiple delegees addresses in a single transaction; please note that each token transfer transaction has to be mined in a different block to increase the arrays length.

Recommendation: As per the project description DDX token has taken heavy inspiration from Compound governance token. Using Compound logic will be enough to avoid this issues, mappings were used instead of arrays as shown here.
Note that the issue also exists in `DIFundToken.sol`.

QSP-3 Diamond proxy pattern increases the attack surface of governance controlled contracts

Severity: High Risk

Status: Mitigated

File(s) affected: `Governance.sol`, `DerivaDEX.sol`

Description: Compared to other governance logic, `Governance` is a facet of the `DerivaDEX` main proxy contract. All `DerivaDEX` facets share the same state (`InsuranceFund`, `Governance`, etc.).
Transfering ownership to self using `transferOwnershipToSelf()` and allowing `diamondCut()` to be called by a governance proposal is a risk since it will allow majority group or an attacker as explained in the "DoS Attack" issue to intoduce malicious code that will be executed in the context of the `DerivaDEX` contract, thus allowing access to insurance fund stakers assets or any other state variable in the contract.
On a more obvious way, a proposal can also contain a direct transaction that transfers users staked assets from insurance fund to any address.
The FUD created around such action is enough to hurt `DerivaDEX` image, even if the governance proposal fails. Please note that malicious proposals cannot be canceled unless the proposer address' voting threshold falls below the `proposalThreshold` as implemented here.

Recommendation: The governance was inspired by Compound governance, however, the diamond standard makes the attack surface higher as explained above.
To mitigate this issue, we recommend to implement both of following solutions:

1. Compound mitigated this issue (even if they used a regular proxy pattern) by using a guardian address that was allowed to cancel the proposals, it can be considered to be used in an initial phase then disabled later on.
2. Use different diamonds for Governance and InsuranceFund, or any other facets that handle users own assets. Governance is made to handle public actions not to manage users owned assets without any limitation.

Update from the Derivadex team: Economic structures in place that prevent this from happening practically. We will not be using an admin key embedded in the code, but proposals can be voted upon (and theoretically fast tracked to the queue) based on the pre-mine voting distribution. Just in case, however, an update was made such that all proposals (including malicious code upgrades) would take 3 days before execution, allowing users to withdraw their funds before they could be taken. There is a fastpath that allows for proposals to immediately execute. This fastpath is initialized with just the pause function (may be used early on if there are bugs, etc.), but this can be removed via governance in due time.

QSP-4 Missing input validation

Severity: Medium Risk

Status: Fixed

File(s) affected: `DDX.sol`, `Governance.sol`, `DerivaDEX.sol`, `InsuranceFund.sol`, `Trader.sol`

Description: Several functions do not fully sanitize for faulty arguments. Constructors and initializer functions should ensure that all relevant address arguments are non-zero to avoid faulty deployments. Setter functions should ensure that the underlying state of the contract remains consistent, e.g., variables that constitute lower bounds should be less than variables that denote upper bounds.
In the function `DDX.transferOwnershipToDerivaDEXProxy()` is allowed to be called only once and can be subject to input error. We recommend ensuring that `_derivaDEXProxy != address(0)`.
When setting `DDX` address in the `DerivaDEX` constructor, `DDX` is not checked to be different than `address(0)`.
In `Governance.sol`:

- `skipRemainingVotingThreshold` has to be higher than `quorumVotes` otherwise all proposal can eventually reach a defeated state following the `state()` function implementation. If the contract is self governed and a change is made where the `skipRemainingVotingThreshold` is lower than the `quorumVotes`, there will be a single way for reverting this situation since all future proposals will fail except if the last vote is high enough to reach both `skipRemainingVotingThreshold` and `quorumVotes` at the same time when the vote is casted.
- When setting governance parameters, some rules should be enforced programmatically such as the `minimumDelay` and `maximumDelay`. As an example, if `maximumDelay` is set lower than `minimumDelay`, all proposal will be rejected.
- The function `castVote()` does not revert if the `msg.sender` address votes are equal to zero.

In `InsuranceFund.sol`:

- The function `setAdvanceIntervalReward()` sets the weight drop between intervals. This should likely be bounded above by 100 through a require-statement.
- Similarly for `setWithdrawalFactor()`, this should be capped to 1000.
- The address parameters to `addInsuranceFundCollateral()` should at least be ensured to be non-zero.

In `Trader.sol`, `setRewardCliff()` may require a bound (possibly 100).

Recommendation: Add require-statements to each of the above functions to mitigate incorrect state variable updates.

QSP-5 `totalSupply()` does not denote the current total supply

Severity: Low Risk

Status: Fixed

File(s) affected: `DDX.sol`

Description: Total Supply is the total amount of coins in existence at any moment. The DDX token `totalSupply()` function returns the `TOTAL_SUPPLY` state variable (please note that `TOTAL_SUPPLY` is the max cap and not the existing supply and the state variable is set without being fully minted); `circulatingSupply` is used instead to count the existing token supply that is minted progressively.
When burning tokens, the amount is deducted only from `circulatingSupply` and not from `TOTAL_SUPPLY` making the contract return wrong estimation of the existing tokens when calling `totalSupply()`.

Recommendation: This issue can be solved by deducting the amount burned from `TOTAL_SUPPLY`. however, a requirement in the `mint()` function has to be changed since `TOTAL_SUPPLY` is used as max cap; a different state variable has to be used to check and store the max cap value.
The same logic should be used when minting, `TOTAL_SUPPLY` should be increased not `circulatingSupply`.

QSP-6 Mapping Storage Layout

Severity: Low Risk

Status: Acknowledged

File(s) affected: `DDX.sol`

Description: Using mappings with value types that require less than 32 bytes of storage will not save gas, instead, more gas will be required when the data is handled in memory.
DDX token uses `uint96` as mapping value type for allowances and balances, this is adding extra complexity when implementing ERC20 standard functions, more gas consumption and non-compliance with the ERC20 standard.
As an example using `safe96` inside `approve()` may cause function to revert for a reason not described in the standard; this can break some applications that interact with ERC20 tokens if they set any value between `uint96` and `uint256` (please not that `approve()` is intended to accept any value).

Recommendation: Change the allowances and balances value types to `uint256` and update the required code.
Update: Since certain functions do take advantage of the current layout, the current layout was not changed.

QSP-7 `skipRemainingVotingThreshold` may cause voting "races" if set below 50%

Severity: Low Risk

Status: Fixed

File(s) affected: `Governance.sol`

Description: Setting `skipRemainingVotingThreshold` value lower than 51% of the `DDX` supply or the total voting power will incentivize the voters to race against each other since the first group to reach that value will most probably win by either defeating the proposal of succeeding, since the voting deadline won't be necessary as seen here.
Following the importance of the proposal or if the proposal contains malicous code or actions, using `skipRemainingVotingThreshold` can help attackers reach their goal faster.

Recommendation: Disallow setting `skipRemainingVotingThreshold`to less than 50%.

QSP-8 Flashloan Vote

Severity: Informational

Status: Fixed

File(s) affected: `DDX.sol`

Description: Even if `getCurrentVotes()` is not used by the implemented Governance facet, using it in future iteration may make the system vulnerable to a flashloans vote, since it includes the user last CheckPoints even if it was created in the same block.

Recommendation: Remove the `currentVote()` function from the DDX token contract.

QSP-9 Nonstandard Naming in `transferFrom()`

Severity: Informational

Status: Fixed

File(s) affected: `DDX.sol`, `DIFundToken.sol`

Description: The function `transferFrom()` always allows the transfer of funds if the `msg.sender` is equal to the `_sender` param.

Recommendation: This is an uncommon implementation, however, the logic is correct but `_sender` naming should be changed to `_from` to be more clear.

QSP-10 Unlocked Pragma

Severity: Informational

Status: Fixed

File(s) affected: `Several Contracts`

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.4.*`. 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, it is recommended to remove the caret to lock the file onto a specific Solidity version.

QSP-11 Redundant Requirements

Severity: Informational

Status: Fixed

File(s) affected: `DDX.sol`, `DIFundToken.sol`

Description: Checking `msg.sender` to be different than `address(0)` is unecessary and consume extra gas only. This occurs in the functions `approve()`, `increaseAllowance()`, and `decreaseAllowance()`.
In `_transferTokens()`, the function already requires that the `_spender` and `_recipient` addresses must be different than `address(0)`; adding any extra requirements in the `transfer()` function is unecessary.

Recommendation: Unecessary requirements should be removed.

QSP-12 Cannot retrieved ether deposited through `receive()`

Severity: Informational

Status: Fixed

File(s) affected: `DerivaDEX.sol`

Description: The `DerivaDEX` contract implement receive method to handle transaction with empty `msg.data` to receive ether instead of the fallback function. However, no function is implemented to handle ether in the contract balance; any funds sent by mistake will be frozen except if a facet with a `withdraw()` function is added.

Recommendation: If the implemented `receive()` function is unnecessary, remove it and add a requirement in the fallback function to throw the transaction when the `msg.data` length is equal to zero.

QSP-13 Contract will not pause if multiple diamond facet upgrade transactions occur

Severity: Informational

Status: Acknowledged

File(s) affected: `DiamondFacet.sol`

Description: If during an upgrade `diamondCut()` is called multiple times, this situation can create issues since the fallback function does not pause, meaning that the users will be exposed to upgraded functions and old functions waiting to be upgraded at the same time.

Recommendation: Be sure to do all the necessary upgrade in a single transaction, otherwise implement a pause mechanism in the fallback function and enable it before starting the facets upgrade.

QSP-14 Storage data packing can be optimized

Severity: Informational

Status: Fixed

File(s) affected: `DiamondFacet.sol`, `DiamondStorageContract.sol`

Description: All data packing for `DiamondStorageContract` is done manually when structures can be used instead. The compiler will tightly pack any ordered state variable by group of 32 bytes and use only one storage slot (also applicable for struct members). Please note that all necessary operation for reading and writing the variables will be created and optimized automatically by the compiler.
The `DiamondFacet` contract handles all the data packing logic, however, the implemented code can be simplified for better reading and acceptance by the developers community.

Recommendation: As an example `mapping(bytes4 => bytes32) facets` maps to `address facet` (represented as a `bytes20`), `uint32 slotIndex` and `uint64 slotsIndex`. It can be replaced as follow:

```
struct Facet {
        bytes20 facetAddr;
        uint32 slotIndex;
        uint64 slotsIndex;
}
mapping(bytes4 => Facet) facets;
```

The same logic applies to all the other state variables.

QSP-15 Variable type inconsistency

Severity: Informational

Status: Fixed

File(s) affected: `DiamondStorageContract.sol`, `DiamondFacet.sol`

Description: `facets` should store `address facet`, `uint32 slotIndex` and `uint64 slotsIndex` in a single storage slot, however, the values saved here are loaded as `uint128`. Even if the total value will likely not be higher than `2^32`, the developers should keep a certain consistency when implementing their solutions.
The possibility of an overflow happening is very low, but it should always be considered.

Recommendation: As recommended in other issues above, use structure instead of self implementing the storage custom packing. Keep the same types and sizes for variables that represent the same value and usage.

QSP-16 Allowance Double-Spend Exploit

Severity: Informational

Status: Mitigated

File(s) affected: `DDX.sol`, `HDUMToken.sol`

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

Exploit Scenario: An example of an exploit goes as follows:

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-17 Gas Usage / `for` Loop Concerns

Severity: Medium Risk

Status: Fixed

File(s) affected: `InsuranceFund.sol`

Description: Gas usage is a main concern for smart contract developers and users, since high gas costs may prevent users from wanting to use the smart contract. Even worse, some gas usage issues may prevent the contract from providing services entirely. For example, if a `for` loop requires too much gas to exit, then it may prevent the contract from functioning correctly entirely. It is best to break such loops into individual functions as possible.
In particular:

- If the number of checkpoints is very large, `advanceInsuranceMineInterval()` may run into gas issues.
- In `claimDDXFromInsuranceMining()`, if the staker has not claimed rewards in many intervals, there may be gas issues.

Recommendation: Perform gas analysis to determine if these scenarios may be problematic in practice. If gas usage is indeed a concern, consider allowing users to claim up to a given interval (rather than up to the current interval, so that the reward claims may be split across multiple transactions.

QSP-18 `permit()` does not validate ECDSA parameters

Severity: Medium Risk

Status: Fixed

File(s) affected: `DDX.sol`, `DIFundToken.sol`

Description: The function does not validate either of the `s` and `v`. values. See ECDSA.sol.

Recommendation: Add checks for the `s` and `v` parameters.

QSP-19 Staking may result in loss of funds when different multipliers are used

Severity: Medium Risk

Status: Fixed

File(s) affected: `InsuranceFund.sol`

Description: When staking, `stakeToInsuranceFund()` will issue an amount of `DIFundToken` for the `msg.sender` equal to `_amount.proportion96(multiplier, 10)` (please note that the minted amount of `DIFundToken` is used to track the user deposited stake) where the multiplier is used as incentive. Users with higher multiplier value will be issued a higher value of `DIFundToken` and the opposite is true.
When withdrawing, `withdrawFromInsuranceFund()` will burn the `DIFundToken` input `_amount` and calculate the collateral amount to give back to the `msg.sender` (`underlyingToTransferNoFee`). If different multiplier values are used when the users stake, the users that get a higher multiplier can get a higher collateral amount withdrawal than his initial deposit (the opposite is also true for lower multiplier value), since they get the proportion of `claimCheckpoint.cap` divided by `stakeCollateral.diFundToken.getTotalPriorValues(block.number.sub(1))` of the input `DIFundToken` to be burned.
Please note that a users that stake `X` amount of collateral, in a normal scenario, it is expected that they get the same value of collateral returned when burning the totality of his issued `DIFundToken`.

Recommendation: Clarify that this functionality is as-intended. Ensure that users are aware of this possible scenario through documentation.

QSP-20 Aave accrued interest model may be gamed by users staking toward the end of an interval

Severity: Medium Risk

Status: Acknowledged

File(s) affected: `InsuranceFund.sol`

Description: Aave balances normally auto-increment through time with the accrued interest in the user's own balances. When Aave tokens are staked in the `InsuranceFund`, the staked value is saved at the moment of staking, the accrued interest will be still generated but for the total balance of the `InsuranceFund`, however the accrued interest for Aave tokens is calculated and divided when claiming later on.
As an example, if we take a staking interval equal to 1 week, a user that stakes 1K aUSDT at day 1 will receive the same value of the accrued interest as a staker that stakes at day 7; the total accrued interest is equally divided between the users following their staking collateral at the end of the interval as implemented in `transferTokensAave()` function. Aave instantaneously divides all accrued interest between the stakers, meaning that if both stakers kept their collateral and didn't stake it, the initial user will gain more interest than if he staked in the insurance fund.
A malicious user can stake a high amount of Aave tokens before the end of an interval to gain more interest, meaning that he can takes others stakers Aave accrued interest.
The implemented logic to distribute Aave accrued interest is dependent on transaction ordering.

Recommendation: Revise the accrual scheme for Aave token staking.

QSP-21 Ignored exception cases in transfer functions

Severity: Low Risk

Status: Fixed

File(s) affected: `InsuranceFund.sol`

Description: In `transferTokensAave()` and `transferTokensCompound()`, it is not clear why exceptions are ignored. If the transfer fails the transaction won't get reverted.

Recommendation: Revert in exception cases.

QSP-22 Unable to remove collateral types in `InsuranceFund`

Severity: Low Risk

Status: Acknowledged

File(s) affected: `InsuranceFund.sol`

Description: While new collateral types can be added, there is now way of removing an insurance fund collateral. If something goes wrong, it would stay in the contract forever, and could make loops over `collateralNames` more expensive.

Recommendation: Consider adding the ability to remove collateral types.
Update: As the cleanup for this feature would be non-trivial (e.g., ensuring funds are correctly returned to users), this functionality has been left out for now.

QSP-23 External tokens should be added cautiously to `InsuranceFund`

Severity: Informational

Status: Acknowledged

File(s) affected: `InsuranceFund.sol`

Description: The function `addInsuranceFundCollateral()` allows the admin (governance) to admit new collateral types, which could be arbitrary tokens or contracts.

Recommendation: Ensure that any added tokens are legitimate and secure contracts.

QSP-24 DDX reward computation dependent on USD-pegged stable coins

Severity: Low Risk

Status: Acknowledged

File(s) affected: `InsuranceFund.sol`

Description: DerivaDEX plans to list only USD stable coins, however all kind of tokens can be listed. The issue described below is applicable for both USD stable coins and other tokens (if listed).
As implemented here, the sum of the user staked collaterals and the total staked collaterals sum is used to compute the interval DDX reward for the `msg.sender`.
DDX reward is calculated following the underlying token unit in the case of compound or simply the aToken unit at the moment of staking for Aave or any other kind of tokens, The face value in USD of different stablecoins can drift from the target price, as an example if Token A face value is 1.05 USD and token B value is 0.95 USD, both stake will receive the same reward if the staked value is the same.
Further, extra care should be used if tokens are added that contain a decimal value different than 18.

Recommendation: Ensure only dependent USD-backed coins are added to the system.

QSP-25 Unclear if-case in `claimDDXFromInsuranceMining()`

Severity: Undetermined

Status: Fixed

File(s) affected: `InsuranceFund.sol`

Description: On L678, it is not clear why `(stakeCollateral.flavor == InsuranceFundDefs.Flavor.Vanilla)` is mentioned, since the only case handled in the code is for Aave, due to the if-statement on L680. It is not clear if additional code is required here for vanilla cases.

Recommendation: Clarify if this function is as-intended.

QSP-26 Users must check-in every rewards interval to avoid diluted COMP and Aave rewards

Severity: Low Risk

Status: Unresolved

File(s) affected: `InsuranceFund.sol`

Description: The function `advanceOtherRewardsInterval` checks how much COMP and Aave is owed to the `InsuranceFund` contract, which will be rewarded to stakers proportional to their deposits. If users have not claimed their owed rewards in previous intervals, the unclaimed rewards are aggregated into the same total. This unclaimed amount is then re-distributed proportionally across all users.
As an example, suppose there are two users, both have equal stakes, and the fund has earned 100 COMP during an interval. If one user invokes `claimDDXFromInsuranceMining`, they will receive the 50 COMP, and 50 COMP will remain unclaimed. Now, suppose another interval passes and the contract has earned another 100 COMP (now totaling 150 unclaimed COMP). If the same user invokes `claimDDXFromInsuranceMining`, they will now be rewarded 75 COMP, bringing their total to 125 COMP, which is more than 50% of the total earned rewards.
Thus, in order to maximize reward payouts, users must claim their rewards every interval, which may not always be desirable from a gas-cost perspective.

Recommendation: Ensure this approach to rewards is made clear to end users.

QSP-27 Privileged Roles and Ownership

Severity: Informational

Status: Unresolved

File(s) affected: `InsuranceFund.sol`, `HDUMToken.sol`

Description: The following additional privileged roles were noted as of commit `7194839`.
In `InsuranceFund.sol`:

1. The admin can invoke `extendInsuranceMining` to extend the mining final block number arbitrarily.

In `HDUMToken.sol`:

1. Users addresses can be blacklisted individually;
2. All main ERC20 functions can be paused;
3. Tokens can be minted for any selected address without limitation;
4. Users assets can be burned by coin factory admin using the `redeem` function.

Recommendation: Ensure that these roles are made clear to users through documentation.

Automated Analyses

Slither

Slither produced several warnings related to using strict equalities (e.g., `checkpoints[_user][userNum - 1].id == blockNumber` within `DIFundToken._writeCheckpoint()`, however these were classified as false positives.

Code Documentation

The code is generally very well documented. We have a few suggestions for improvements:

1. In `DiamondStorageContract.sol`, the comments should explicitly state how the storage variables should be broken down into their components. For example, `selectorSlotsLength` is 256-bits long, but claims to only store two 32-bit fields. The comment should mention that the first 128 bits are used for the first field, and the remaining 128 bits are used for the second. This was only apparent when examining `DiamondFacet.diamondCut()`.
2. Similar to above, `facets` maps to a `bytes32`, which is decomposed into "address facet, uint32 slotIndex, uint64 slotsIndex". However, this is only `20 + 5 + 6 = 31` bytes, so it is not clear where the remaining byte is allocated.
3. In L103 of `DiamondFacet.sol`, "at then end" should be "at the end".
4. On L152,157 of `DiamondFacet.sol`, `selectorSlotLength--;` occurs on both the `if` and `else` branches and could be moved after.
5. In `MathHelper.sol`, the comment on L35 should be "Returns the median of three numbers."

As of commit 7194839, we noted the following:

1. In `InsuranceFund.sol`, the numbers on L206: "1.189e18 ~ 5% liquidity mine (50mm tokens))" would benefit from more explanation. Similarly on L253.
2. The `HDUMToken.sol` contract does not contain any code documentation, Please note that without a clear description of the token usage we cannot estimate its overall impact on the system security.
3. The inline comment description of `InsuranceFund.isNotPaused` does not correspond to the modifier usage, since it is the owner of the governance that can initiate the pause mechanism, but the role of the modifier itself is to pause sensitive functions.

Adherence to Best Practices

The code generally adheres to best practices. We have the following minor recommendations below:

1. The conditions on L665 of `DDX._moveDelegates()`: `uint96 initDelOld = initDelNum > 0 ? checkpoints[_initDel].votes[initDelNum - 1] : 0;` is unnecessary, since even if the condition is false, the SafeMath operation `.sub()` just below will throw.

2. We generally recommend using `uint256` instead of `uint` throughout to make the size of the variables more apparent.

3. In `TraderInternalLib.sol`, the commented out import on L15 should be removed.

4. In `DiamondFacet.sol`, L113, 178, 182: spaces added at the end of the line.

5. In `InsuranceFund.sol`, there is a lot of overlap between the functions `getCurrentStakeByCollateralNameAndStaker()` and `getStakeByCollateralNameAndStakerByInterval()`, which could be abstracted to a helper function.

6. The constraint `msg.sender != address(0)` can never evaluate to false and can be removed. This occurs in `DIFundToken.sol` on L142, 172, 204, 247.

As of commit 7194839, we noted the following:

1. The `Blacklist` contract in `HDUMToken.sol` does not need to inherit ERC20 contract.

Test Results

Test Suite Results

We note that the insurance-related tests due to network timeouts.

```
    ✓ checks that ETH cannot be sent directly to the DerivaDEX contract
    ✓ checks pre-mine and liquidity mining balances (93ms)
    ✓ ensures everyone has the proper vote count (161ms)
    ✓ ensures delegation results in correct vote count (691ms)
    ✓ transfers from a trader to another (319ms)
    ✓ approves another users to transfer on behalf (500ms)
    ✓ fails to transfers ownership of DDX to another address from unauthorized address (42ms)
    ✓ fails to transfers ownership of DDX to the zero address (38ms)
    ✓ transfers ownership of DDX to another address (162ms)
    ✓ fails to transfers ownership of DDX since already done once
    ✓ fails to mint DDX from unauthorized issuer
    ✓ mints DDX to addresses (1499ms)
    ✓ fails to burn too many tokens
    ✓ fails to burn from with an unauthorized address
    ✓ fails to burn from too many tokens (163ms)
    ✓ burns DDX from addresses (1346ms)
    ✓ do some additional delegation downstream (415ms)
    ✓ approves using permit (229ms)


  18 passing (10s)

    ✓ adds Trader facet (1266ms)
    ✓ stakes DDX properly (1696ms)
    ✓ fails to maliciously reinitialize onchain DDX wallet (40ms)
    ✓ lifts governance cliff (884ms)
    ✓ withdraws DDX properly (786ms)


  5 passing (11s)

    ✓ fails to add Governance facet with invalid skipRemainingVotingThreshold below 50pct (409ms)
    ✓ fails to add Governance facet with invalid skipRemainingVotingThreshold and quorumVotes (371ms)
    ✓ adds Governance facet (1514ms)
    ✓ transfer ownership of Proxy to itself (170ms)
    ✓ checks quorum vote count, proposer vote count, and skip remaining voting threshold count (113ms)
    ✓ fails to propose when proposer not above threshold (69ms)
    ✓ fails to propose when proposal parity misaligned (77ms)
    ✓ fails to propose when proposal has no actions (58ms)
    ✓ fails to propose when proposal has too many actions (151ms)
    ✓ makes new proposal and fails to cast vote before voting delay (490ms)
    ✓ fails to propose when proposer already has active proposal (101ms)
    ✓ check vote receipts prior to voting for proposal 1 (136ms)
    ✓ casts vote (488ms)
    ✓ fails to cast another vote after already voting (42ms)
    ✓ fails to queue proposal since it has not succeeded yet (48ms)
    ✓ fails to cast a vote from participant with no voting power (69ms)
    ✓ casts another vote (196ms)
    ✓ fails to execute proposal that is not queued (40ms)
    ✓ queues successful proposal since voting period can be skipped with enough for votes (151ms)
    ✓ fails to queue proposal again
    ✓ fails to execute proposal that has not been in queue long enough (46ms)
    ✓ executes successful proposal (284ms)
    ✓ fails to set invalid skip remaining votes threshold (765ms)
    ✓ fails to set invalid quorum votes (732ms)
    ✓ checks fastpath delay for setIsPaused (940ms)


  25 passing (12s)

    ✓ fails to add InsuranceFund facet bypassing Governance
    ✓ adds new Insurance fund facet via Governance  (34373ms)
```

```
      ✓ check lock and unlock of stake (1577ms)
      ✓ fails to stake cUSDT to insurance fund since unsupported collateral (498ms)
      ✓ adds cUSDT and aUSDT to insurance fund as valid collateral types  (19031ms)
      ✓ checks supported collateral names have been added and addresses and flavors (172ms)
      ✓ Trader A - stakes USDT and cUSDT in the first interval (18314ms)
      ✓ Trader B - stakes USDT in the first interval (6613ms)
      ✓ fails to advance mine interval when called too early (60ms)
      ✓ distributes DDX to insurance miner A (40438ms)
      ✓ fails to distribute DDX to insurance miner A again (70ms)
      ✓ distributes DDX to insurance miner B (4878ms)
      ✓ Trader A - stakes USDT in the second interval (4117ms)
      ✓ Trader C - stakes AUSDT in the second interval (17864ms)
      ✓ distributes DDX to insurance miners A, B, and C (30624ms)
      ✓ withdraws USDT from insurance fund in third index (7658ms)
      ✓ distributes DDX to insurance miners A, B, and C again (37310ms)
      ✓ distributes DDX with the max loop (141966ms)

  18 passing (9m)
```

Code Coverage

Code coverage could not be performed with the current tech stack.

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

```
1e5c4522f06bec7bbc22435668067f0e6dd3b2890f8e48b62e42fb342bc07525 ./contracts/DerivaDEX.sol
481a1212b93d33c6a1c48d5479711521f63861b02de3e90527b6508d459661b9 ./contracts/Migrations.sol
df4e74052a0cb2c76858fcb3ac0ebe14e3e917e0384fea28d9d7dee1ce82eb19 ./contracts/tokens/DDX.sol
45a1c9acb92af69acfbc9dce30d2dc796757b3e0d705718d5ce3d73e702179d1 ./contracts/tokens/DDXWalletCloneable.sol
7b704c33b8fc2749956812fae0e1f5c3944ae6aa5f2ee0ffc7595eca71ce8382 ./contracts/tokens/DIFundToken.sol
11c78552d534d9dabf938dc0177c3953d489903202d02acdbb73933e86b17c4b ./contracts/tokens/DIFundTokenFactory.sol
fb7d15aa5ae62e402d00762a8e658c468476d9c385db90abb0bf7e10e2becf8a ./contracts/tokens/DummyToken.sol
899bdc976214f58d130eeb50ee81d1ec89298602763889e7e33e01fd34e4e013 ./contracts/tokens/HDUMToken.sol
f79b0857cd36d6d1a7ae9c68df14d753e5e7eeef1e27d1ca7577bd1e6a423de6 ./contracts/tokens/SafeERC20Wrapper.sol
e1ac7d23f312eb0e6c0caa2332ed3662a29f001150d68291fd7a3bbf813e0cbe ./contracts/tokens/interfaces/IDDX.sol
5ebeac479a279624cda3e3db6054f5d6965059bfd61e2c64fb5ec2e640e3e8a4 ./contracts/tokens/interfaces/IDDXWalletCloneable.sol
b847e12ac5cad25dd65f4ae8e33d44a3fc44958285cd71c410b1f917f0683655 ./contracts/tokens/interfaces/IDIFundToken.sol
e7fa4b09d9a51b8237a92bba4224c0d1dffdc88fa638a8848cf9f8f2c4b36fe4 ./contracts/tokens/interfaces/IDIFundTokenFactory.sol
969a11c5da0043aa81eda6b215fe49cb964372c053558db1b22737519983b815 ./contracts/storage/LibDiamondStorageDerivaDEX.sol
fd10ff12c4055ad76667b7d12bf71a24d8806e5d3a88565c772313bcf0b202ae ./contracts/storage/LibDiamondStorageGovernance.sol
5152dd4772adda0dd85005b8efe30a92874fc63129de3e10f38688972ef78f52 ./contracts/storage/LibDiamondStorageInsuranceFund.sol
e2ed1700204a79c966fa1269d38d3f42acd255acbbc24c177eef8a7c30088bba ./contracts/storage/LibDiamondStoragePause.sol
8f0253652be2ad0fc5ea9a8fb9bb282da65be7f874d23e48ab343ad2fa1ae57c ./contracts/storage/LibDiamondStorageTrader.sol
df3379c642aeae095a9b18cb25897695b7667d8b7d242b7f554819924f7a69d7 ./contracts/libs/LibBytes.sol
3e60a807dff5858d676444fca343d1face4ea19e62af3643fea338f043f20686 ./contracts/libs/LibClone.sol
696e29f8044c9057cf6190b67f1b3cd907d62479459e28ea0a177b8d931dc5d7 ./contracts/libs/LibDelegation.sol
dc4ec4ba7e47942da551593e7c660f7837ad76c31ed9bd4d8c92e0664d782652 ./contracts/libs/LibEIP712.sol
3373fce2fe4877e56f49b064d7476cef538391bfc096e17d08a4efcf704fe9bd ./contracts/libs/LibPermit.sol
5e615b2d291ffbdbfac59ec1b5ec158978c6f98d34289fd872ac5ab62f08ab11 ./contracts/libs/LibVoteCast.sol
8a99a8abe49bcf27b95a51a2ab218f49882a138530e1e11ae2a49a829b2b03e9 ./contracts/libs/MathHelpers.sol
21ff291e6164b47c22e5d9bcb5f82580dafe723b15fa78edb0c0bd14b136a284 ./contracts/libs/SafeMath128.sol
7bb1c3a391b7fe6359cc160778172fd8f61a3e52b61e46b70733c11cbc67f4d3 ./contracts/libs/SafeMath32.sol
e456c0ad257118542200159eb747408ddd1b3c2d6d537d16ee567fc8b9912af3 ./contracts/libs/SafeMath96.sol
5fa46a434dd8947c220e9a130a21bf327a8f3eee0d2dfc2e111dd5cfa9c998c8 ./contracts/libs/defs/GovernanceDefs.sol
ac3cfcfbb85baf24d51ca1206d8f53f758b57a1a15a823b98a40c5084f5c0748 ./contracts/libs/defs/InsuranceFundDefs.sol
914961730d8c283451d839b9df524ffeb7163ebd5955392531f0a0d54320b833 ./contracts/libs/defs/TraderDefs.sol
62f241598867a213fb7282c81a3f752c2998a260d51571f60bdaba52d942f820 ./contracts/facets/trader/LibTraderInternal.sol
8660ef4f0b33af00291a8946acb27b544a3d03149ccf4f14d6dbaec2502ab84b ./contracts/facets/trader/Trader.sol
9c7e14d662c097e3fa63ec47cc97a36d2ad4d2e1bf6ea42307a5a2887e9f4e0d ./contracts/facets/pause/Pause.sol
cd3f2c7528b0f040fec484711072f6178f3a8eba19657c662e773587bf5ff803 ./contracts/facets/interfaces/IAToken.sol
3a4f6eb27804723d93eca5be735ed6f1337de0b8de08233e375acaddaf45e489 ./contracts/facets/interfaces/IComptroller.sol
d3ee37cb7665fa8b993c032412bf7480e7406a275908bff481b5f1124ddc985f ./contracts/facets/interfaces/ICToken.sol
8de11743e6a86d975fb126722ca1fe616a6d0c9cb06145e58c3c0982052dcc9e ./contracts/facets/interfaces/IInsuranceFund.sol
6b793835c51afc91826987c330d6bdeaf698ab399817130a2ef83ab4251ff084 ./contracts/facets/insurance-fund/InsuranceFund.sol
b9415f5c53d5fa3b5551e84df50e4f02e75c7dbb718ed19a08a37e2662063459 ./contracts/facets/governance/Governance.sol
ad7e49f503fa4c573eba4510de73297c220e83712dc5d8c726be9ccb4314ef53 ./contracts/diamond/DiamondFacet.sol
a9a65608a416ea9658d72edd2eb4affa42c4a237052a2cf86955d6fe1f7f759d ./contracts/diamond/IDiamondCut.sol
577e386d85b76a9bbf1a569bd8e9bd649862db7e71fdbdef3188204c9ddce00a ./contracts/diamond/IDiamondLoupe.sol
3ea19697318cbcda14d0ea9d574c08ea2b15f999213435b698f26cd4d16fd4d3 ./contracts/diamond/IERC165.sol
6c4b96481828ac2efde4b55377b3dbf5d868f67443a1fbe57799f0dcb6c76487 ./contracts/diamond/LibDiamondCut.sol
0a7db742a932f80df2cdeb5f03a750a237929e4bedbfaff7e5a9bbc734dd5828 ./contracts/diamond/LibDiamondStorage.sol
eecc5d30cdea610b220300caa7d17f0f5014a719c578797868f9ac95efc0aa30 ./contracts/diamond/OwnershipFacet.sol
```

Tests

```
15006ec8023e2c7b30a56324647e41e78bbe3fa44343a648b6abecdda0580320 ./test/fixtures.ts
c95180275ed52917a1bc869323620a6973ab2327777f53c499ad0eaa8d8f30c4 ./test/misc.ts
62e169e2640c0cfc57298cbe2c36e3585350fd7b49f53a4ae43b98eb7f34ef51 ./test/setup.ts
c9f9b257235c8d98fe424dd7bdcf6b0b3fe41c6975cf61e636fa7cb17ee49b0b ./test/setupKovan.ts
b095c5fe285e5940782db1e77d910dad7cd05e2e30277bfcf16d9e76d7de0450 ./test/derivadex/TestDDX.ts
16891300d3b9e719a975f1a1784c0464289d52cc195745b9b3c4f2d131b0f626 ./test/derivadex/TestDiamond.ts
a85135ca101f6a820ef2486f89b224f8c41509a1ddf2db741fc9e05c16480087 ./test/derivadex/TestGovernance.ts
b2d492149eb591afdfc20d65ed98c95441e0d9b1335b2bbd07a605b76958f406 ./test/derivadex/TestInsuranceMining.ts
10809f846c0f8f8f35fe2eba1540b6481601813eb50f30ff257143fc34a99c4d ./test/derivadex/TestTrader.ts
```

Changelog

- 2020-09-04 - Initial report
- 2020-09-17 - Updated report based on commit c4f974a
- 2020-10-08 - Updated report based on commit 0dbe8788
- 2020-10-26 - Updated report based on commit 7194839

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.