Quantstamp Security Assessment Certificate

QUANTSTAMP VERIFIED SECURITY CERTIFICATE

July 8th 2021 — Quantstamp Verified

Clipper

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

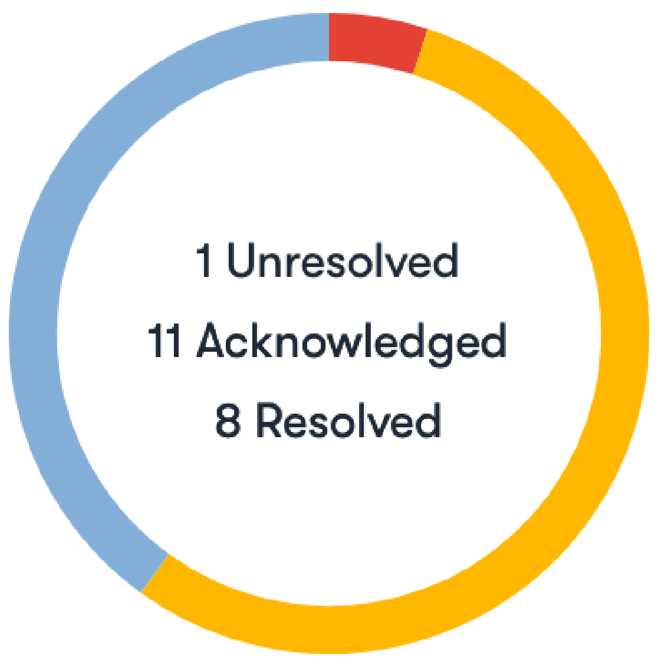
Executive Summary

Type	Cryptocurrency Exchange
Auditors	Sebastian Banescu, Senior Research Engineer Fayçal Lalidji, Security Auditor Joseph Xu, Technical R&D Advisor
Timeline	2021-04-19 through 2021-05-25
EVM	Berlin
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	New Invariants for Automated Market Making README.md
Documentation Quality	Medium
Test Quality	Low

Source Code

Repository	Commit
galleon-dex	0dff47e

Total Issues	20	(8 Resolved)
High Risk Issues	2	(1 Resolved)
Medium Risk Issues	8	(4 Resolved)
Low Risk Issues	3	(0 Resolved)
Informational Risk Issues	5	(2 Resolved)
Undetermined Risk Issues	2	(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

Quantstamp has performed a security audit of the Galleon DEX smart contracts and has determined 20 issues ranging from High to Informational severity and 2 issues have an Undetermined severity level. The exchange pool has an above-average level of owner privileges, one of which has been marked as a High severity issue. Another concerning issue regards the low number of tests in the test suite. In addition to the aforementioned issues, 8 best practice deviations were detected. We recommend fixing all of these issues before deploying the code in production.

Update after reaudit: Quantstamp has performed a reaudit according to commit hash `72f2dad` and the official responses obtained from the development team. The report has been updated accordingly.

ID	Description	Severity	Status
QSP-1	The Galleon DEX Owner May Arbitrarily Mint LP Tokens	High	Mitigated
QSP-2	Insuficient Test Suite	High	Unresolved
QSP-3	Loss Of Funds If Given Procedures For Deposits & Swaps Not Followed	Medium	Acknowledged
QSP-4	Loss Of Funds During Withdrawals	Medium	Acknowledged
QSP-5	Liquidity Provider Fees	Medium	Fixed
QSP-6	Possibly Incorrect Invariant Check	Medium	Fixed
QSP-7	Oracle Prices Used Could Be Stale Or Manipulated	Medium	Fixed
QSP-8	ERC20 Compliance	Medium	Fixed
QSP-9	Transfer/Transaction Mechanism Not Required In All Contracts	Medium	Acknowledged
QSP-10	Missing OFAC Blocked Wallets	Medium	Acknowledged
QSP-11	Privileged Roles and Ownership	Low	Acknowledged
QSP-12	Existing Vested Deposits May Get Locked Again	Low	Acknowledged
QSP-13	Missing Input Validation	Low	Acknowledged
QSP-14	Strong Assumption About Number of Decimals Returned by Oracle	Informational	Fixed
QSP-15	Important State Changes Are Difficult To Monitor	Informational	Fixed
QSP-16	Ambiguous Naming	Informational	Acknowledged
QSP-17	PLP Compliance	Informational	Acknowledged
QSP-18	Custom Square-Root Function Does Not Work For Input Value `2`	Informational	Acknowledged
QSP-19	Blocked Addresses Can Still Deposit & Withdraw	? Undetermined	Fixed
QSP-20	The `GalleonPool.constructor()` Mints 10x More Tokens	? 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.

Findings

QSP-1 The Galleon DEX Owner May Arbitrarily Mint LP Tokens

Severity: High Risk

Status: Mitigated

File(s) affected: `GalleonPool.sol`

Description: The owner of the exchange pools may perform many privileged actions as described in the "Privileged Roles and Ownership" finding from this report. However, one privilege that stands out is that of arbitrarily minting LP tokens. Most of the other privileges are common in DeFi applications, but the arbitrary LP token minting makes this a High severity issue, because of the impact it may have on end-users holding LP tokens. By minting a large amount of LP tokens, the owner could withdraw a large number of deposited tokens from the Galleon pool, leaving depositors short-handed. This scenario is not necessarily assuming a malicious owner; it could very well be the case that the private key of the owner is stolen and the attacker does this. From the existing documentation and code comments, it is not clear to the auditors which use-case this arbitrary minting of LP tokens would be needed for.

Recommendation: Remove `GalleonPool.mint()` function.

Update: From dev team:

> "While the ability to mint tokens is important and will not be removed, our code now limits mint calls to only once every five days, with a maximum mint amount of 5% of the token base."

It is unclear why pool owners are able to mint. Max 5% minting every 5 days annualized is max 3522%.

QSP-2 Insuficient Test Suite

Severity: High Risk

Status: Unresolved

Description: Branch coverage of some of the core contracts in this project is too low. For example, the branch coverage for the `GalleonPool` is 40% and 55% for the `GalleonExchangeInterface`. This means that there are several potential code execution flows, which are not tested to work as expected and could therefore work differently than expected.

Recommendation: We strongly recommend writing a more comprehensive test suite in order to ensure that all the functionality and use-cases of this project are properly tested. Additionally, we recommend testing using Mainnet forking and adding a few ERC20 tokens which do not have typical behaviour such as USDC and USDT.

QSP-3 Loss Of Funds If Given Procedures For Deposits & Swaps Not Followed

Severity: Medium Risk

Status: Acknowledged

File(s) affected: `GalleonDeposit.sol`, `GalleonExchangeInterface.sol`, `GalleonPool.sol`

Description: Users need to process swaps (deposits) by first transferring ETH or one of the supported ERC20 tokens to the liquidity pool contract and then calling the proper API functions on the exchange interface (the `deposit()` function in the `GalleonDeposit` contract), which provides the proper output after re-calculating the invariant based on the actual liquidity pool contract balances. Note that this makes the DEX susceptible to front-running or transaction order sequencing by validators.

Since the transfer and the API calls are separate from each other, users must ensure that the strict procedure is followed to achieve the correct, intended output. Transaction ordering also matters so users may need to complete the transfer and the API call in the same tx. Otherwise, end-user funds could be stolen by malicious actors. For example, malicious actors could perform front-running attacks if the end-user uses an EOA to directly deposit funds before exchanging. There are currently few safeguards against users interacting with the smart contracts directly and there are also no features that can facilitate these actions to be processed in the same tx.

For example, the PLP API functions implemented `GalleonPool` and the functions implemented in `GalleonExchangeInterface` are allowed to be called by any address or contract, making front-running attacks accessible in case of misuse of the transfer mechanism by a contract (bad implementation) or an EOA. The vulnerable functions use the same naming for both contracts and are listed below:

```
    function sellTokenForToken(address inputToken, address outputToken, address recipient, uint256 minBuyAmount, bytes calldata auxiliaryData) external returns (uint256 boughtAmount);
    function sellEthForToken(address outputToken, address recipient, uint256 minBuyAmount, bytes calldata auxiliaryData) external payable returns (uint256 boughtAmount);
    function sellTokenForEth(address inputToken, address payable recipient, uint256 minBuyAmount, bytes calldata auxiliaryData) external returns (uint256 boughtAmount);
```

Exploit Scenario: Below are certain scenarios that may lead to incorrect outputs:

- An incorrect amount of LP tokens might be given on deposit if two users deposit into the pool simultaneously, with the first user calling `GalleonDeposit.deposit()` before the second one. In this case, the first user would receive the LP tokens equivalent to both deposits and the second user would receive no LP tokens.
- An incorrect amount of tokens might be swapped if two users deposit the same tradable asset into the pool and one user makes the API call before the other.

Recommendation: Provide a smart contract or functions to facilitate the transfer and API call in the same tx. Communicate the risks associated with this design clearly to the users through proper documentation. This includes informing third-party contract developers about the importance of sending the tokens and calling the required functions in a single transaction.

Update: From dev team:

> "We acknowledge this issue. We expect most users will interact with our exchange through an Aggregator or through our website."

QSP-4 Loss Of Funds During Withdrawals

Severity: Medium Risk

Status: Acknowledged

File(s) affected: `GalleonExchangeInterface.sol`

Description: The function `GalleonExchangeInterface.withdrawInto()` allows withdrawals as long as the invariant does not decrease too much as the result of the withdrawal. The parameter of output amount needs to be specified by the user, but the user may miscalculate and withdraw too few tokens in exchange for the LP tokens. Notably, the function allows withdrawals with `outputTokenAmount = 0`, which would lead to the user not receiving any assets in exchange for the LP tokens.

Recommendation: Add basic checks in the `withdrawInto()` function to revert if the `outputTokenAmount` is too small relative to the difference in the invariant.

Update: From dev team:

> "We are fine with users inputting the amount they would like back and acknowledge it could result in a potential loss. In practice, we expect users to perform the calculation offline and then add in a modest buffer. The alternative (on-chain binary search for the invariant) was deemed too gas-expensive."

QSP-5 Liquidity Provider Fees

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `GalleonExchangeInterface.sol`

Description: The fee logic inside the `withdrawInto()` and `withdraw()` functions will always leave assets in the contract, meaning that the last user to withdraw his asset will see his calculated fees being left out in the contract. Depending on the amount left, an attacker can deposit an extremely small amount of tokens (compared to the left out fees) and withdraw the majority of the remaining asset in the pool.

Recommendation: Change the fee mechanism such that fees are correctly paid upon withdrawals. Clarify why fees are applied to the LPs in end-user-facing documentation.

QSP-6 Possibly Incorrect Invariant Check

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `GalleonExchangeInterface.sol`

Description: On L197, the `withdrawInto()` function computes the `invariantFractionIBurned` using `theExchange.fullyDilutedSupply()`, which has been updated after `amount` LP tokens have been burned through `GalleonPool.swapBurn()`. Based on the code comments and the implementation of `GalleonExchangeInterface.withdraw()`, it seems like the correct invariant check needs to be using the initial fully diluted supply before it is updated.

Recommendation: Double-check the invariant condition to clarify if it should be calculated using the initial fully diluted supply or the updated fully diluted supply.

QSP-7 Oracle Prices Used Could Be Stale Or Manipulated

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `libraries/AggregatorInterface.sol`, `GalleonExchangeInterface.sol`, `GalleonPool.sol`

Description: The `GalleonPool.findBalanceAndMultiplier()` and the `GalleonExchangeInterface.invariant()` functions use the price from the call to the deprecated API function `oracle.latestAnswer()` of Chainlink. This approach is vulnerable to price manipulation and stale prices according to the Chainlink documentation:

1. under current notifications: "if answeredInRound < roundId could indicate stale data."
2. under historical price data: "A timestamp with zero value means the round is not complete and should not be used."

Recommendation: We recommend using the `latestRoundData()` function as recommended by the Chainlink documentation and adding require statements that check for the aforementioned conditions in all the occurrences of those functions, while also checking the `startedAt` and `updatedAt` values to check if the price returned by the oracle is stale since Chainlink aggregators rely on third party project to be updated. Develop contingency plans in the case of oracle failure (pause swaps, use a fall-back oracle, etc.)

Update: Transactions will revert if the oracle is stale due to the `require()` statement in `SafeAggregatorInterface.safeUnsignedLatest()`.

QSP-8 ERC20 Compliance

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `GalleonPool.sol`

Description: `GalleonPool.upsertAsset` and `GalleonPool.getMarketShare` assume that the listed tokens' decimals are always lower than 18 decimals. Such implementation will prevent listing tokens with higher decimals since the transactions will throw due to underflow. Please note that the ERC20 standard does not require the decimals value to be fixed or lower than any specific value.

Recommendation: The developers must be compliant with the ERC20 standard to avoid any future issues.

QSP-9 Transfer/Transaction Mechanism Not Required In All Contracts

Severity: *Medium Risk*

Status: Acknowledged

Description: The 0x PLP requires only the following function to handle direct transfers to `GalleonPool`:

```
function sellTokenForToken(address inputToken, address outputToken, address recipient, uint256 minBuyAmount, bytes calldata auxiliaryData) external returns (uint256 boughtAmount);
function sellEthForToken(address outputToken, address recipient, uint256 minBuyAmount, bytes calldata auxiliaryData) external payable returns (uint256 boughtAmount);
function sellTokenForEth(address inputToken, address payable recipient, uint256 minBuyAmount, bytes calldata auxiliaryData) external returns (uint256 boughtAmount);
```

However, the same mechanism is used by `GalleonPool.deposit()` where it is not required by 0x since the 0x protocol will not interact with any function other than the functions stated above.

Recommendation: We strongly recommend using the safest way possible to handle user assets, in this case, `approve/transferFrom` for any other function that does not require any compatibility with the PLP API.

Update: From the dev team:

> We prefer the "deposit, then check" modality for its simplicity and modularity. We acknowledge that users interacting with our contracts outside of our own onramps or the onramps of trusted DEX aggregator partners may lose funds if they behave carelessly.

QSP-10 Missing OFAC Blocked Wallets

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `BlacklistAndTimeFilter.sol`

Description: The `BlacklistAndTimeFilter.constructor()` blocks 3 OFAC addresses. However, there are other addresses listed by OFAC which are not included. For example this post indicates 2 different Ethereum addresses which are not listed:

1. `0xd882cfc20f52f2599d84b8e8d58c7fb62cfe344b`
2. `0x7F367cC41522cE07553e823bf3be79A889DEbe1B` Also, the current version of the OFAC SDN List contains 23 Ethereum addresses (some of which are duplicates), many of which are not included in the `BlacklistAndTimeFilter.constructor()`.

Recommendation: While it is possible to block new addresses post-deployment, using the `BlacklistAndTimeFilter.blockAddress()` function, it is our understanding that the intention of the developers is to include all addresses on the OFAC SDN List in the `BlacklistAndTimeFilter.constructor()`. Therefore, we recommend adding all unique addresses from that document in the constructor.

Update: From the dev team:

> The `BlacklistAndFilter` contract is intended to be a demonstration implementation of the approval interface for testing. We will be sure to block all listed OFAC wallets prior to launch.

QSP-11 Privileged Roles and Ownership

Severity: Low Risk

Status: Acknowledged

File(s) affected: `GalleonPool.sol`

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

1. Mint any amount of Galleon Pool Tokens (GLNPL) at any time to any address.
2. Transfer any amount of any token (including ETH) from the `GalleonEscapeContract`
3. Add and/or remove ERC20 tokens to the asset set. Removing ERC20 tokens has a 5 day period in which end-users can withdraw their deposits in that token. Afterward withdrawing a deposit in that token is no longer possible by an end-user.
4. Withdraw the total balance of an ERC20 token that has been removed from the pool.
5. Withdraw the total balance of ETH from the pool only if there are no ERC20 tokens in the `assetSet`.
6. Change the address of the exchange interface contract to any address at any time.
7. Change the address of the deposit contract to any address at any time.
8. Change the triage address to any address at any time.
9. Change the approval contract of the `GalleonExchangeInterface` contract to any address at any time.
10. Change the swap fee in the `GalleonExchangeInterface` contract to any value below `500` (representing 5%).

The `owner` of the `GalleonExchangeInterface` contract is allowed to set the `GalleonPool theExchange` address once.
The `BlacklistAndTimeFilter` contract owner can:

- Change the pool address.
- Allow and deny swaps for the entire pool.
- Allow and deny deposits to the entire pool.
- Block and unblock address.
- Modify the minimum deposit days.

Recommendation: Clearly document all the different user roles, capabilities, and impact of those capabilities in end-user-facing documentation.

Update: From the dev team:

> We acknowledge that the `owner` of the contract does, indeed, wield a great deal of power.

QSP-12 Existing Vested Deposits May Get Locked Again

Severity: Low Risk

Status: Acknowledged

File(s) affected: `GalleonDeposit.sol`

Description: If a user does not first unlock a fully vested deposit and makes a new deposit, the vested deposits will get locked up along with the new deposit.

Exploit Scenario: One example of how this issue would affect users:

1. Alice deposits 1 ETH using `GalleonDeposit.deposit(7 days)`.
2. After 1 week, Alice's 1 ETH deposit has vested and she can unlock the pool tokens.
3. However, Alice deposits another 2 ETH using `GalleonDeposit.deposit()` before calling `GalleonDeposit.unlockVestedDeposit()`.
4. Alice's vested pool tokens corresponding to the 1 ETH deposited a week prior is locked again and cannot be withdrawn since existing `deposits[address(alice)]` is updated based on the timestamp of the new 2 ETH deposit.

Recommendation: Consider adding a check to prevent deposits before unlocking a vested deposit, or automatically unlocking and minting vested LP tokens upon new deposits.

Update: This behavior is intended. To avoid such issues users should always unlock deposits before making a new deposit.

QSP-13 Missing Input Validation

Severity: Low Risk

Status: Acknowledged

File(s) affected: `GalleonPool.sol`, `GalleonExchangeInterface.sol`

Description: The following instances of this vulnerability were identified:

1. `GalleonPool.constructor()` does not validate the `initialExchangeInterface` input parameter value.
2. `GalleonPool.upsertAsset()` does not validate the `marketShare` input parameter value.
3. `GalleonPool.modifyExchangeInterfaceContract()` does not validate the `newContract` input parameter value.
4. `GalleonPool.modifyDepositContract()` does not validate the `newContract` input parameter value.
5. `GalleonPool.modifyTriage()` does not validate the `newTriageAddress` input parameter value.
6. `GalleonPool.balancesAndMultipliers()` does not validate if the `inputToken` and `outputToken` addresses are different.
7. `GalleonPool.syncAndTransfer()` does not validate if the `inputToken` and `outputToken` addresses are different. It also does not check if the `recipient` is different from `address(0)` or if the `amount` is higher than `0`.
8. `GalleonExchangeInterface.constructor()` does not validate any of its input parameters.
9. `GalleonExchangeInterface.setPoolAddress()` does not validate if the `poolAddress` complies with `GalleonPool`.

Recommendation:

1. Use EIP-165 to check if `initialExchangeInterface` complies with the `GalleonExchangeInterface`.
2. **[Fixed]** Check if the `marketShare` value is between `0` and `100%`. Also document how many decimal digits the value is represented with. It is unclear if `100 == 100%` or if `10000 = 100%`.
3. Use EIP-165 to check if `newContract` complies with the `GalleonExchangeInterface`.
4. Use EIP-165 to check if `newContract` complies with the `GalleonDeposit`.
5. Check that `newTriageAddress` is different than `address(0)`.
6. Check that the `inputToken` and `outputToken` addresses are different.
7. Check that the `inputToken` and `outputToken` addresses are different. Check that `recipient` is different from `address(0)` and the `amount` is higher than `0`.
8. Use EIP-165 to check if `initialApprovalContract` complies with the `ApprovalInterface`.
9. Use EIP-165 to check if `poolAddress` complies with the `GalleonPool`.

Update: From dev team:

> Some additional validations have been added, but others will not be fixed:
>
> - Adding EIP-165 checks for functions that will only be called by admins is, we believe, superfluous
> - `marketShare` is an inverse weighting calibrated so that 100 = ETH, 50 = twice the weight of ETH, and 200 = half the weight of ETH.
> - Checking to see if **inputToken** and **outputToken** are different addresses is unnecessary.

QSP-14 Strong Assumption About Number of Decimals Returned by Oracle

Severity: Informational

Status: Fixed

File(s) affected: `GalleonPool.sol`

Description: We understand that the intention of Galleon DEX is to use ETH as the numeraire asset and we can also see that in the current list of Chainlink oracles all oracles of the form `<ERC20>/ETH` return 18 decimals. However, there is no guarantee that this will be the case for such Chainlink oracles in the future. The require statement on L151 in `GalleonPool.sol`: `require(oracle.decimals()==18, "Galleon: Invalid oracle");` prevents adding Chainlink price feed contracts which do not have exactly 18 decimals digits. One Chainlink price feed contract which has a value different from 18, namely 8 decimals is the ETH/USD price feed contract. Many other Chainlink price feed contracts, which do not have 18 decimals can be found here.

The code inside `GalleonPool.findBalanceAndMultiplier()` assumes that the `latestAnswer()` function of the Chainlink price feed contract always returns a result with 18 decimal digits. However, this is not guaranteed as described above. We also refer to the FAQ section of the Chainlink documentation where the following question and answer are listed:

> **Why is latestAnswer reported at 8 decimals for some contracts, but for other contracts it is reported with 18 decimals?**
> For crypto quotes, 18 decimals is typically used because they usually require more precision. For FX quotes, 8 decimals are used because that is the precision data sources typically report them at.

If at some point in time oracles that return a number of decimals different than 18 will be needed and included, it will have a significant impact on all trading activities.

Recommendation: We recommend accepting oracles with any number of decimals and using a state variable that keeps track of how many decimals each oracle returns. This state variable must then be used to adjust the value of `weiPerInput` (and all other values that represent token amounts) according to the number of decimal digits returned by the oracle.

QSP-15 Important State Changes Are Difficult To Monitor

Severity: Informational

Status: Fixed

File(s) affected: `GalleonPool.sol`

Description: The `GalleonPool` pool smart contract allows changes to the exchange interface, deposit contract, and the triage address. However, the functions for these changes do not emit events.

Recommendation: Consider emitting events in functions `modifyExchangeInterfaceContract()`, `modifyDepositContract()`, and `modifyTriage()`.

QSP-16 Ambiguous Naming

Severity: Informational

Status: Acknowledged

File(s) affected: `UniERC20.sol`

Description: `UniERC20.uniTransferFromSender` does not transfer from the sender when the token address is equal to zero (Ether address) this can be subject to implementation error where tokens will be transferred from the wrong address. As an example `GalleonExchangeInterface.sellEthForToken` uses `UniERC20.uniTransferFromSender` but it is not sending ether from the msg.sender instead it is sending ether from GalleonExchangeInterface contract to the GalleonPool.

Recommendation: We recommand to use UniERC20.uniTransfer as best practice and change UniERC20.uniTransferFromSender implementation.

Update: From dev team:

> Since there's no such thing as "transferring from" with ETH, the analogue to "transferring from" is forwarding on whatever is attached to the transaction as value. The transfer from the ExchangeInterface to the Pool contracts is intentional and is designed to handle ETH swaps.

QSP-17 PLP Compliance

Severity: Informational

Status: Acknowledged

File(s) affected: GalleonExchangeInterface.sol

Description: The following functions implemented in GalleonExchangeInterface are not compatible with the PLP interface and yet the declarations of the same functions are used:

```
function sellTokenForToken(address inputToken, address outputToken, address recipient, uint256 minBuyAmount, bytes calldata auxiliaryData) external returns (uint256 boughtAmount);
function sellEthForToken(address outputToken, address recipient, uint256 minBuyAmount, bytes calldata auxiliaryData) external payable returns (uint256 boughtAmount);
function sellTokenForEth(address inputToken, address payable recipient, uint256 minBuyAmount, bytes calldata auxiliaryData) external returns (uint256 boughtAmount);
```

Depending on which address is going to be added to 0x, GalleonPool or GalleonExchangeInterface, the functions in the latter contract will not handle the asset sent to the contract by the 0x protocol correctly.

Recommendation: We recommend changing the function naming in GalleonExchangeInterface to be different than the PLP API, to avoid any possible misuse.

Update: From dev team:

> We don't see a reason to change these function names. Our own router code will operate directly on the ExchangeInterface contract.

QSP-18 Custom Square-Root Function Does Not Work For Input Value 2

Severity: Informational

Status: Acknowledged

File(s) affected: Sqrt.sol

Description: According to the code comment on L8 and one of the unit tests, the Sqrt.sqrt() function does not work correctly for input value 2, because instead of returning 1, it returns 2. This function should typically truncate the decimals of the square root result for any given number. In the case of input value 2 the square root result is incorrectly returned as being equal to 2. Since this function is involved in invariant computations both during deposits and withdrawals it might affect the computations to some extent. The reason why we classify the severity of this issue as Informational is that it is unlikely to ever occur in the current version of the code since we can expect the invariant() function to operate on number orders of at least 1e18 and the invariantSwap() function to operate on number orders of 1e36. Those are the only 2 functions where the sqrt() function is currently being called.

Recommendation: To avoid any miscalculations we recommend adjusting the implementation such that it correctly handles the input value 2.

Update: From dev team:

> This was intentionally introduced. The gas cost of checking if a number is 2, a highly unlikely input, was deemed to be too severe.

QSP-19 Blocked Addresses Can Still Deposit & Withdraw

Severity: Undetermined

Status: Fixed

File(s) affected: BlacklistAndTimeFilter.sol, GalleonDeposit.sol, GalleonExchangeInterface.sol

Description: Blocked addresses cannot swap, but they can still deposit into the pool, receive LP tokens, and withdraw assets.

Recommendation: Clarify whether this is intentional or not. Otherwise, modify BlacklistAndTimeFilter.sol, GalleonDeposit.sol, and GalleonExchangeInterface.sol to prevent blocked addresses from interacting with the smart contract in these ways.

Update: From dev team:

> - **Deposit**: Example filter contract has been amended to filter based on msg.sender, providing a device to block OFAC deposits.
> - **Withdraw**: As a non-custodial mechanism, we cannot in any way block withdrawals.

QSP-20 The GalleonPool.constructor() Mints 10x More Tokens

Severity: Undetermined

Status: Acknowledged

File(s) affected: GalleonPool.sol

Description: The instruction on L111: _mint(msg.sender, msg.value*10); inside the GalleonPool.constructor() mints 10 times more Galleon Pool Tokens (GLNPL) than the amount of ETH it is receiving. This behavior is not documented and it is unclear if it is correct/intended or not.

Recommendation: Either document this behavior in the specification and user-facing documentation, or fix the code such that the same amount of GLNPL is minted as the amount of ETH received.

Update: From dev team:

> This was an intentional choice made to sever any implicit link between ETH and Galleon Pool tokens. Pool tokens will not, in general, directly correspond to any number of ETH in the pool.

Adherence to Best Practices

1. Some functions have no comments. We recommend that each function have at least a short description of its purpose, its input parameters and output values, if any.
2. Some functions and contracts which do have comments right before the body are not using the Solidity NatSpec format, which is recommended as a best practice.

3. Magic numbers should be replaced by named constant and the names of such constants should provide the semantics of the value. The following instances of magic numbers have been encountered in the code:

   - `2000` on L85 in `contracts/GalleonDeposit.sol`
   - `86400` on L113 in `contracts/GalleonDeposit.sol`
   - `18` on L151, L153, L163 in `contracts/GalleonPool.sol`
   - `100` on L242 in `contracts/GalleonPool.sol`
   - `10` on L111, L153, L163 in `contracts/GalleonPool.sol`
   - `432000` on L176 in `contracts/GalleonPool.sol`
   - `500` on L67 in ` contracts/GalleonExchangeInterface.sol
   - `10000` on L122, L222, L229 in `contracts/GalleonExchangeInterface.sol`
   - `1e4` on L197 in `contracts/GalleonExchangeInterface.sol`
   - `1e10` on L207, L221, L228 in `contracts/GalleonExchangeInterface.sol`

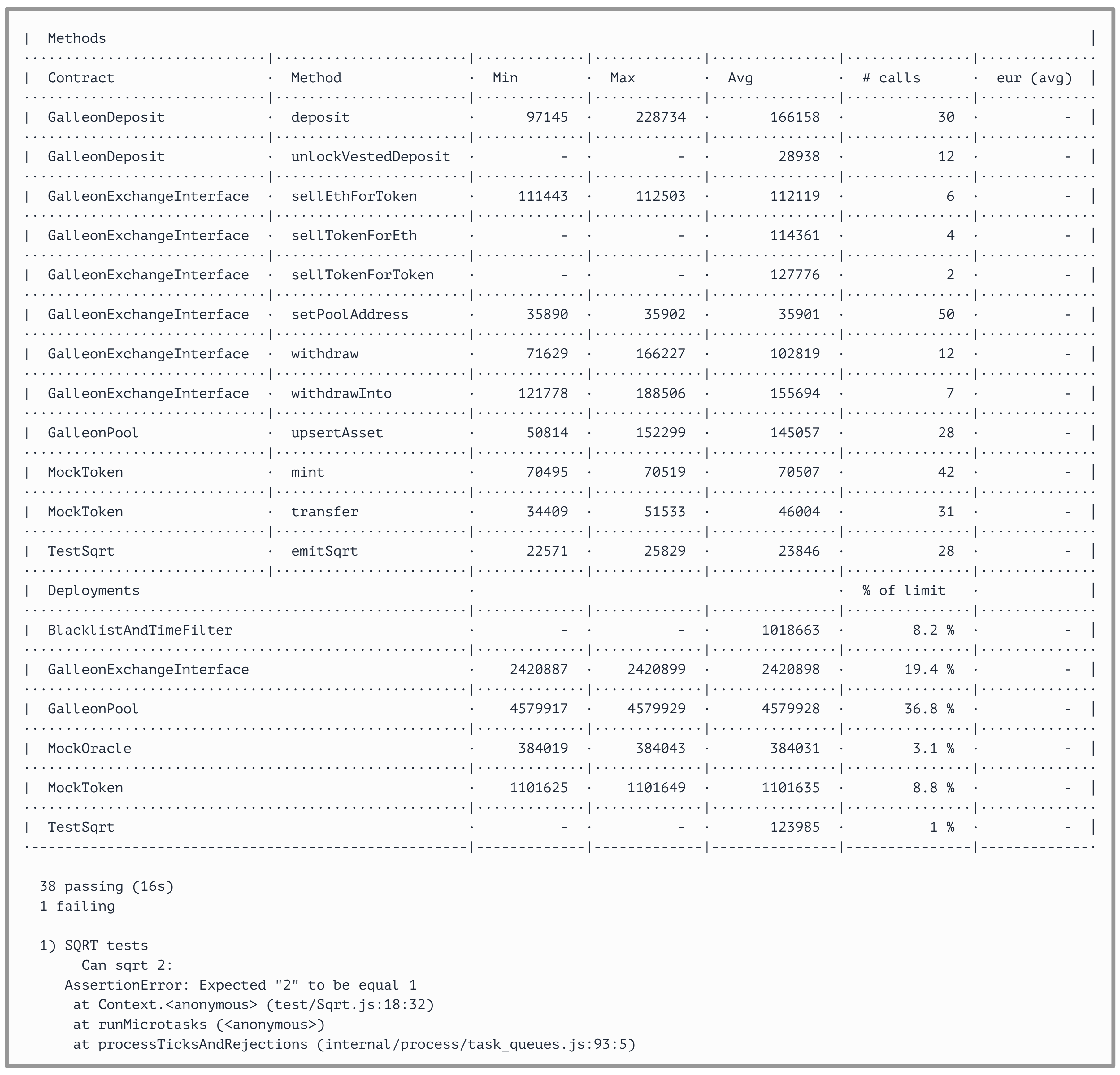
4. Use a `mapping` instead of `EnumerableSet.AddressSet` for the `assetSet` state variable inside the `GalleonPool` contract if you want to reduce gas costs for the functions that operate on `assetSet`.
5. Inconsistent use of `uint` vs `uint256`. Replace all usages of `uint` to `uint256`.
6. Wrap L182 of `GalleonPool.sol`: `delete assets[token].removalTime` in an `if (assets[token].removalTime > 0) {}` in order to reduce gas costs as indicated here.
7. The function `GalleonPool.recordUnlockedDeposit()` might be better re-named as `mintUnlockedDeposit()`.
8. There is no check corresponding to the comment on GalleonPool::L154 `// Don't deposit a token before upserting it`... However, this should not be a very serious issue since the token balances can be escaped before calling `GalleonPool.upsertAsset()`.

Test Results

Test Suite Results

We note that 38 tests are passing and 1 test is expected to fail by design.

```
% npx hardhat test

 ·-----------------------------|-------------·
 |  Contract Name              ·  Size (Kb)  |
 ······························|··············
 |  SafeERC20                  ·       0.08  |
 ······························|··············
 |  Address                    ·       0.08  |
 ······························|··············
 |  EnumerableSet              ·       0.08  |
 ······························|··············
 |  Sqrt                       ·       0.08  |
 ······························|··············
 |  UniERC20                   ·       0.08  |
 ······························|··············
 |  TestSqrt                   ·       0.32  |
 ······························|··············
 |  MockOracle                 ·       1.19  |
 ······························|··············
 |  GalleonEscapeContract      ·       1.73  |
 ······························|··············
 |  ERC20                      ·       2.88  |
 ······························|··············
 |  GalleonDeposit             ·       3.33  |
 ······························|··············
 |  BlacklistAndTimeFilter     ·       3.87  |
 ······························|··············
 |  MockToken                  ·       4.38  |
 ······························|··············
 |  GalleonExchangeInterface   ·      10.33  |
 ······························|··············
 |  GalleonPool                ·      13.70  |
 ·-----------------------------|-------------·

  Exchange Function Test
    Deployment
       ✓ Can deploy
       ✓ Starts with correct invariant
    ETH alone operations
       ✓ Can deposit ETH alone correctly
       ✓ Can deposit ETH for 1 day correctly
    Add Token
       ✓ Cannot swap to OFAC blocked address
       ✓ Cannot ETH->REP with too high min buy amount
       ✓ Can ETH->REP with ETH attached and transferred
       ✓ Cannot REP->USDC if USDC not valid token
       ✓ Cannot REP->ETH if nothing transferred
       ✓ Can ETH->REP even if selling too much ETH
       ✓ sellTokenForEth: Can trade REP into a pool with 0 REP
       ✓ Can ETH->REP after attaching ETH to call
       ✓ Can swap between two tokens
       ✓ Can swap from ETH after adding tokens
       ✓ withdrawInto correctly from 3-token pool
       ✓ withdrawInto correctly from 2-token pool
       ✓ deposit ETH alone for 1 day correctly in 3-token pool that has both ETH and REP
       ✓ deposit ETH alone for 1 day correctly in 2-token pool that has both ETH and REP
       ✓ Can add demonstration token
       ✓ Can modify demonstration token
       ✓ Has same invariant prior to adding demonstration tokens
       ✓ Has correct invariant after adding some demonstration tokens
       ✓ Can add a second token
       ✓ Can deposit ETH alone correctly into a pool with 2 tokens
       ✓ Can deposit ETH alone for 1 day correctly into a pool with 2 tokens

  SQRT tests
     ✓ Can sqrt 0
     ✓ Can sqrt 1

    1) Can sqrt 2
     ✓ Can sqrt 3
     ✓ Can sqrt 4
     ✓ Can sqrt 9
     ✓ Can sqrt 1000000000000
     ✓ Can sqrt 1000000000000000000
     ✓ Can sqrt 100000000000000000000
     ✓ Can sqrt 1000000000000000000000000
     ✓ Can sqrt 1000000000000000000000000000000
     ✓ Can sqrt 2000000000000000000
     ✓ Can sqrt 2000000000000000001
     ✓ Can sqrt 1999999999999999999

·-----------------------------------------------------|---------------------------|--------------|-----------------------------·
|                Solc version: 0.8.3                  ·  Optimizer enabled: true  ·  Runs: 10000  ·  Block limit: 12450000 gas  |
······················································|···························|··············|······························
```



```
|  Methods                                                                                                                  |
·······························|··························|·············|·············|················|···············|··············
|  Contract                    ·  Method                  ·  Min        ·  Max        ·  Avg           ·  # calls      ·  eur (avg)  |
·······························|··························|·············|·············|················|···············|··············
|  GalleonDeposit              ·  deposit                 ·      97145  ·     228734  ·        166158  ·           30  ·          -  |
·······························|··························|·············|·············|················|···············|··············
|  GalleonDeposit              ·  unlockVestedDeposit     ·          -  ·          -  ·         28938  ·           12  ·          -  |
·······························|··························|·············|·············|················|···············|··············
|  GalleonExchangeInterface    ·  sellEthForToken         ·     111443  ·     112503  ·        112119  ·            6  ·          -  |
·······························|··························|·············|·············|················|···············|··············
|  GalleonExchangeInterface    ·  sellTokenForEth         ·          -  ·          -  ·        114361  ·            4  ·          -  |
·······························|··························|·············|·············|················|···············|··············
|  GalleonExchangeInterface    ·  sellTokenForToken       ·          -  ·          -  ·        127776  ·            2  ·          -  |
·······························|··························|·············|·············|················|···············|··············
|  GalleonExchangeInterface    ·  setPoolAddress          ·      35890  ·      35902  ·         35901  ·           50  ·          -  |
·······························|··························|·············|·············|················|···············|··············
|  GalleonExchangeInterface    ·  withdraw                ·      71629  ·     166227  ·        102819  ·           12  ·          -  |
·······························|··························|·············|·············|················|···············|··············
|  GalleonExchangeInterface    ·  withdrawInto            ·     121778  ·     188506  ·        155694  ·            7  ·          -  |
·······························|··························|·············|·············|················|···············|··············
|  GalleonPool                 ·  upsertAsset             ·      50814  ·     152299  ·        145057  ·           28  ·          -  |
·······························|··························|·············|·············|················|···············|··············
|  MockToken                   ·  mint                    ·      70495  ·      70519  ·         70507  ·           42  ·          -  |
·······························|··························|·············|·············|················|···············|··············
|  MockToken                   ·  transfer                ·      34409  ·      51533  ·         46004  ·           31  ·          -  |
·······························|··························|·············|·············|················|···············|··············
|  TestSqrt                    ·  emitSqrt                ·      22571  ·      25829  ·         23846  ·           28  ·          -  |
·······························|··························|·············|·············|················|···············|··············
|  Deployments                                            ·                                            ·  % of limit   ·             |
························································|·············|·············|················|···············|··············
|  BlacklistAndTimeFilter                                 ·          -  ·          -  ·       1018663  ·        8.2 %  ·          -  |
························································|·············|·············|················|···············|··············
|  GalleonExchangeInterface                               ·    2420887  ·    2420899  ·       2420898  ·       19.4 %  ·          -  |
························································|·············|·············|················|···············|··············
|  GalleonPool                                            ·    4579917  ·    4579929  ·       4579928  ·       36.8 %  ·          -  |
························································|·············|·············|················|···············|··············
|  MockOracle                                             ·     384019  ·     384043  ·        384031  ·        3.1 %  ·          -  |
························································|·············|·············|················|···············|··············
|  MockToken                                              ·    1101625  ·    1101649  ·       1101635  ·        8.8 %  ·          -  |
························································|·············|·············|················|···············|··············
|  TestSqrt                                               ·          -  ·          -  ·        123985  ·          1 %  ·          -  |
--------------------------------------------------------|-------------|-------------|----------------|---------------|-------------·

  38 passing (16s)
  1 failing

  1) SQRT tests
       Can sqrt 2:
     AssertionError: Expected "2" to be equal 1
      at Context.<anonymous> (test/Sqrt.js:18:32)
      at runMicrotasks (<anonymous>)
      at processTicksAndRejections (internal/process/task_queues.js:93:5)
```

Code Coverage

Branch coverage of some of the core contracts in this project is too low. We strongly recommend writing a more comprehensive test suite in order to ensure that all the functionality and use-cases of this project are properly tested.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	81.73	45.45	61.76	80	
BlacklistAndTimeFilter.sol	41.18	0	23.08	35	… 65,69,73,77
GalleonDeposit.sol	100	70	100	100	
GalleonEscapeContract.sol	33.33	0	50	33.33	19,20
GalleonExchangeInterface.sol	93.9	55	85.71	94.05	61,62,66,67,68
GalleonPool.sol	73.97	40	62.86	72.5	… 306,310,313
contracts/libraries/	63.16	83.33	83.33	65	
AggregatorInterface.sol	100	100	100	100	
ApprovalInterface.sol	100	100	100	100	
Sqrt.sol	0	100	50	14.29	… 26,27,28,30
UniERC20.sol	92.31	83.33	100	92.31	39
contracts/mocks/	55.56	100	50	55.56	
MockOracle.sol	66.67	100	50	66.67	20
MockToken.sol	75	100	75	75	29
SqrtMock.sol	0	100	0	0	15,19
All files	**79.11**	**51.28**	**62.2**	**77.82**	

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

```
08a16b0896bfc8e089c5d59e41ac311442041e3cf289b2d4bf66cad6b1be4745 ./contracts/BlacklistAndTimeFilter.sol
9a70be44f34cf4835c673e03ed83bf3601ded3915f10a4d17084138e6fe5322d ./contracts/GalleonExchangeInterface.sol
681005dc920a9722fb3458b7c9011150fe72b1683ffbe8d6cb981b53a5d30c89 ./contracts/GalleonEscapeContract.sol
7b9091c7bce767aa0b61c98da9ae9d118ea43ea585d436104b889679085ef489 ./contracts/GalleonDeposit.sol
a915ef3b19f059c4d3410727cf636c6946999d533f3d37b3c1c386decf7276ea ./contracts/GalleonPool.sol
1356eab4d685c1bf929493e45551554ed060363577633fbe11c462553bc801d6 ./contracts/libraries/UniERC20.sol
e31f421ce08e14689caeeb60e0071dd0fa45d4e8bec4799d909b42cf99a78b33 ./contracts/libraries/AggregatorInterface.sol
9e9f4ae154b1b668d5d1475382cb0d85013d4d9f3b85c13a6c76bebda8c2d0c9 ./contracts/libraries/Sqrt.sol
8d266ca9b20f910ec460380b9dfc4622fa5d50b9e7f972810b7b765ecc40595c ./contracts/libraries/ApprovalInterface.sol
10b2b7713cfa02502811c539b2ec61bdd49590ea51f2134d6e89c3f1538fc743 ./contracts/mocks/MockToken.sol
a8fd3a0936a228d2298e1f7b355dd0dc726a5eb8c014b14779a138ff1a7d3972 ./contracts/mocks/SqrtMock.sol
2ff2257c3027ae517e5fda65fc3c494dbf53a9f4f7b844a32e1401839b0ffdce ./contracts/mocks/MockOracle.sol
```

Tests

```
09a55815ab003e7a3feef9956d02b002820e16070869f5d52d514e39e9341f53 ./test/Exchange.js
dab5c2271cb5b011411d33e3d8a4f892404ea58777e085f2d8cb4e055a0eb901 ./test/Sqrt.js
```

Changelog

- 2021-05-05 - Initial report based on commit hash `0dff47e`
- 2021-05-25 - Report updated according to commit hash `72f2dad`

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.

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp, Inc. (Quantstamp). Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on the website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.